Introduction to Machine Learning	**Fall Semester, 2013**
Lecture 1: October 13	
Lecturer: Lior Wolf	*Scribe: ym*

1.1 Introduction

The first question posed was *what is computer science?* And we concluded that it is about what happens between the *input* and *output* of a program. In computer vision the input can be a picture and the output could be an identification of the person in the picture.

The second question posed was *what is Machine Learning?* The initial input (the training examples) guide the building of the machine. The goal is to act "correctly" on new unseen examples. For an example from computer vision: The input (training examples) could be the photos of the individual students in the class and the test could be a joint picture of the class. The task could be an identification of the individual students in the picture.

The third question was: *what are tasks in Machine learning.* We have covered many topics

1. **Document or photo classification**. The motivation can be a Google car that drives and needs to distinguish between pedestrians, trucks and trees. The "regular" way to implement this task would be as follows: Have a detector for each *object* (say, pedestrian). The detectors covers the photo with small windows, and for each window it determines if there is a pedestrian in that window. For a window the task can be learned from examples. There is an issue how do we represent the photo. Can be raw pixel values or higher level attributes (say, lines).

2. **Speech recognition** can be either *speaker recognition* task or *transcribing* the speech to words.

3. **Language Translation** a classical problem in NLP. Today the successful automatic tools use machine learning and based on a lot of data. The input for translation can be translated books, or transcription of committee meeting (in the EU or Canada).

4. **Recommendation systems** and **Collaborative Filtering**. The great example is the Netflix challenge, where they offered $1M for a 10% increase in their prediction accuracy. The input is a huge matrix, with users as rows and movies as columns, and the value is the rating a user gave to a movie. Naturally the matrix is very sparse and the goal is to predict the missing values.

5. **Fraud detection**

6. **Driving a car**. This is a control system where the action depends on the input receives from the environment. Such issues are studied in *Reinforcement Learning*. The major difference is that the system has a state, and the action depends not only on the observations but also on the state.

7. **Spam Filter**. One can generate examples from users actions (classifying certain e-mails as SPAM or reading and replying) How do we learn such a task. We have the raw e-mails, and we like to build from them vectors. One natural way is called a *bag of words* where the vector has as entries all possible words in the language, and maintains the word counts. Note that two different e-mails might have the same word counts. The goal is to build a classification function f that given a new email decides if it is SPAM or not. More concretely we have pairs (E_i, y_i) where E_i is an e-mail and $y_i = +1$ if it is SPAM and $y_i = -1$ if it is NOT SPAM. The input (training examples) is $\{(E_i, y_i)\}_{i=1}^n$. The output of the learning algorithm is a function $f : E \to \{-1, +1\}$. This is an example of a binary supervised classification problem.

In machine learning, when building a classifier, there is an inherent tradeoff between accuracy and interpretability. The most accurate classifier is many times complex and hard to interpret. In cases where human interpretability is important, like in medical applications, it comes at the price of reduced accuracy.

1.2 Important challenges in machine learning

How important is the actual learning algorithm and its tuning. The answer depends on the level of accuracy required. It is rare to find examples where one algorithm does fantastically and all others fail miserably. However, in some tasks a fraction of a percentage difference (like in predicting the probability of a click on an ad) can make a huge difference (billions of dollars in revenue).

Simple versus complex algorithm. Simple algorithms sometimes win, since they run faster and can, at the same amount of time, use much more training data. However, there is a new trend of running complex algorithms (such as deep learning) on huge data sets. Sometimes the combination of huge amounts of data with a complex algorithm (where the training can take weeks) is the best we can do today.

An important philosophical question is whether the past really represent the future. Russel example of a chicken is great to illustrate how the past cannot predict the future. We would normally assume that the past and future examples are drawn from the same distribution. Although it is rarely the case, it is a good approximation in most cases.

Over-fitting is the phenomena of fitting the classifier too much to the training data, at the cost of lowering its prediction ability (the task we really like to do!).

Model Selection. One classical solution is Occam's Razor, which says that if we have multiple hypothesis which have the same accuracy, we should prefer the simpler one.

Regularization adds an explicit penalty to the hypothesis as a function of its complexity (beyond its error rate) and thus favors simpler hypotheses.

As an example consider points in the plane. We can try to fit them with a line (and assume that the errors are noise). At the other extreme we can fit them perfectly with a high degree polynomial and get zero training error (but very weak on generalization). The model selection would say, take a polynomial of degree at most k. The regularization might add a penalty that would depend on the norm of the coefficient (large coefficients can make the function change very fast).

1.3 Examples for supervised and unsupervised learning

1.3.1 Nearest neighbor

A very intuitive algorithm, that classifies the new point by the training example which is most similar to it. The training examples are $\{(x_i, y_i)\}_{i=1}^n$. The nearest neighbor (NN) algorithm, given a point z sorts the points according to their distance from z. Let the sorted points by their distance from z be $x_{[1]}, x_{[2]}, \ldots$. The prediction for z is the label of $x_{[1]}$, i.e., the label of the closest point to z in the training set.

A simple variation on the NN is the k-NN algorithm. In this algorithm we take a majority of the k nearest examples. Again, we sort the points by their distance from z. For examples, 3-NN we return $majority(y_{[1]}, y_{[2]}, y_{[3]})$.

The training error of 1-NN is always zero (the nearest point is the point itself).

If the classification function is deterministic (each point has one true label) then with an infinite sample we will converge to a correct classification. If the classification function is stochastic (can be due to the fact that we map many inputs to the same x_i) then the error rate, with an infinite sample, would be $2R^*(1 - R^*)$, where R^* is the optimal error rate.

1.3.2 k-Means

This is unsupervised learning. The examples have no classification, and we like to partition them to clusters. First note that the goal is not well-define, and this is inherent in most of the unsupervised algorithms.

In the k-means we are given a set of examples x_i and a parameter k and like to partition them to k sets, and with each set we associate a "center" μ_i. The goal is to minimize the

objective function

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

The minimization problem is NP-hard, and therefore unlikely to have an efficient algorithm. Here is the k-means algorithm.

K-Means

Initialize Set $t = 1$ and select k values $\mu_1^t, \ldots, \mu_k^t$.

Assign Assign each x_j to the closest out of $\mu_1^t, \ldots, \mu_k^t$. Formally, $C_j^t = \arg\min_i \|x_j - \mu_i^t\|^2$ and $S_i^t = \{x_j | C_j^t = i\}$.

Update Given the sets $S_1^t, \ldots, S_k^t$ re-compute $\mu_1, \ldots, \mu_k$, by setting μ_i^{t+1} to the average in S_i^t, i.e., $\mu_i^{t+1} = (1/|S_i^t|) \sum_{x_j \in S_i^t} x_j$.

- While there are changes return to Assign.

Initialize: Need to pick k values μ_i. One common solution is to pick k points x_j at random. Note that different initializations might give you different solutions.

In each iteration it is clear that the objective function cannot increase. There is a theoretical possibility that it will alternate without an actual decrease, but nearly impossible to occur in practice. When the k-means changes configurations without reducing the cost, we can clearly stop once we repeat a configuration.

The number of iterations of the k-means algorithm can be bounded by k^n, since this is the number of partitions. A better bound is $n^{O(kd)}$ which is computed from the geometry of the problem, considering the number of Veronoi diagrams [1]. The lower bound can be shown to be exponential even for the plane ($d = 2$) [2]. On the line ($d = 1$) there is a lower bound of $\Omega(n)$, even for $k = 2$, and a polynomial upper bound [3].

Bibliography

[1] Inaba, M., Katoh, N., and Imai, H. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In Proc. 10th Annu. ACM Sympos. Comput. Geom., pages 332–339, 1994.

[2] Andrea Vattani k-means Requires Exponentially Many Iterations Even in the Plane Discrete & Computational Geometry 45(4): 596-616 (2011)

[3] Sariel Har-Peled and Bardia Sadri How Fast Is the k-Means Method? Algorithmica 41(3): 185-202 (2005)