Introduction to Machine Learning **Fall Semester, 2016/7**

Lecture 1: October 30, 2016

Lecturer: Yishay Mansour *Scribe: ym*

1.1 Course Administration

- Lecturers: Amir Globerson (gamir@post.tau.ac.il) and Yishay Mansour (Mansour@tau.ac.il)
- Teaching Assistance: Regev Schweiger (schweiger@post.tau.ac.il). Note there are THREE recitations (a new one added).
- Homeworks: 20% of the grade. Done in pairs.
- Final Exam: February 1, 2017. 80% of the grade. (Need to pass exam to pass the course!)

1.2 Machine learning is everywhere

1. **Web Search:** Finding information. Classically part of Information Retrieval, but today uses may machine learning tools and methodologies.
2. **Speech recognition** can be either *speaker recognition* task or *transcribing* the speech to words. Becoming standard on smart phones.
3. **Language Translation** a classical problem in Natural Language Processing (NLP). Today the successful automatic tools use machine learning and based on a lot of data. Again, significant adaptation, starting to become useful in real life.
4. **Image recognition:** Part of computer vision, but is using more Machine Learning tools to reach higher performance. Deep learning is transforming the way we do image recognition.
5. **Recommendation systems**. For example Amazon buyer suggestions. Another 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 in order to give users movie recommendations.

6. **Driving a car**. This is a control system where the action depends on the input receives from the environment. Again, another example of an area which was science fiction a decade ago, and soon to be a reality.

   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. **Drone** Another application whose success depends on reinforcement learning.

Machine learning offers a *basic technology* which is utilized today in many application. Collecting data, and using data to improve performance.

1.3 Typical ML tasks

1.3.1 Supervised learning: classification

From each domain we collect labeled data. For example: In spam detection, we might have emails and their classification as spam or not spam. In health related issues, we might have patient data and the classification might be whether the patient is ill or not. For credit cards fraud, we might have information about the user and the purchase with a classification whether the purchase was legitimate or fraud.

Labels can come in various forms. We will mainly discuss binary label, as the most basic classification task. There are labels from large domains, for example selecting the topic of the document. Another issue about labels is whether a label is unique or can for an instance have multiple labels. For example a news article might have multiple topics.

The input to the machine learning algorithm is a data set. A set of instances from the domain and their classification. We will assume that the instances are selected randomly, and future test instances will come from the same distribution.

The output of the machine learning algorithm is a predictor. The predictor gets as an input an instance (without the label) and predict the label.

We will discuss may types of predictors:

1. Linear classifiers
2. Decision Trees
3. Neural networks
4. Nearest Neighbor

This is called supervised learning since we are getting the instances with labels and our main goal is to predict the label on new unseen instances.

1.3.2 Unsupervised learning: clustering

In unsupervised learning we are receiving instances but without labels. There are really two different cases. The first is when there are labels, and it is costly to do the classification. (In some applications you need to pay people to be do the labeling by hand!). The second is cases where there is no clear label. For example user in a web site. You like to partition users to groups, but unclear how to do it and what will the grouping mean. In this case the goal is mainly to discover a hidden structure.

The input is the training data which include instances with no labeling. The output would be the partition of the space to clusters. The goal is somewhat fuzzy. Many times we set a mathematical objective criteria, and optimize it. But the real goal is to hopefully discover a hidden structure which is natural and informative.

1.3.3 Reinforcement learning: control

In reinforcement learning we have an agent that interacts with the environment. The classical example is a robot, other examples are driving cars and flying drones. The main difference here is that the observation we make depend on the actions we take. For example, if the robot does not change the head direction it will not see other information. This leads to the important tradeoff between exploration and exploitation. Trying new alternatives, to discover new things, versus exploitation, which implies taking the best looking actions.

1.3.4 Why do we need Machine Learning?

- Tasks in which it is difficult to define precisely how they should be executed: When driving a car, one uses human generalization capabilities. We would like the machine to act as the driver would. Many times it is much simpler to give examples rather then rules for how to act (drive).
- Tasks that are beyond human capability: The analysis of huge database or discovering interesting and unknown phenomena.
- What is machine learning? Here is one classical definition.

  Arthur Samuel (1959): "Field of study that gives computers the ability to learn without being explicitly programmed".

1.4 Building a ML model

First we have the instances from the domain. The instances have both attributes and labels. The attributes can be either real-valued, say $\mathbb{R}^d$, or discrete $\{0,1\}^d$, etc. The label can be

binary, i.e., $\{0, 1\}$, discrete, e.g., $\{1, \ldots, k\}$, or real value.

There is no replacement to having the "right" attributes. Bad attributes will mean that the ML task is impossible. The art of finding the right attributes, while extremely important, is not part of this class. (Unfortunately, it is more of an art then an established methodology.)

For the most part we will assume that there that there is a distribution that generates the examples. For the most part we will assume that the example are sampled i.i.d. An important implication is that the training and testing examples are generated from the same distribution. Conceptually, this implies that we believe that the future and past are identical. All this is an assumption regarding the environment.

The output of the learning algorithm is a predictor. Even if the labels are binary, we can have the output be either $\{0, 1\}$ or $[0, 1]$. Outputting $q \in [0, 1]$ can be interpreted as the probability that the true label is 1.

We need to decide how to compare the classifications of different algorithms. Obviously we would like to minimize the number of errors, but perhaps many small errors are preferable to a few big errors. Also, we might want to weight differently false positive and false negative. For example, in a health setting there is a significant difference between making a mistake by predicting that a patient sick when he is not, in which case additional tests will probably find the correct classification, and a mistake by predicting that a patient healthy when he is sick, in which case we risk missing the correct identification.

1.5 Machine learning model: Complete Information

We are given a set of points which has been classified into two groups: '+' and '-'. Notice that the '+' and '-' here are simply labels given to each point.

Given a new point x, we would like to be able to classify it, that is, give it one of the two labels. In this scenario, we assume that we are provided with **Complete Information** namely we know the distribution. The distribution that generates the '+'s is D_+, the distribution that generates the '-'s is D_-, and λ is the probability of $'+'$. The joint distribution is $D(x) = \lambda D_+(x) + (1 - \lambda)D_-(x)$.

Given a point x we would like to label it. The probability that x has a '+' label is $\Pr[+|x]$. Using Bayes rule we have:

$$\Pr[+|x] = \frac{\Pr[x|+] \cdot \Pr[+]}{\Pr[x]} = \frac{\lambda D_+(x)}{D(x)} = p$$

1.5.1 Prediction and loss model

Mistake (0-1 loss)

We are now interested in classifying the point x. Which classification shall we choose? The obvious choice would be minimizing the expected mistakes. This implies that we predict $'+'$

if $p \geq 1/2$, or equivalently, if $D_+[(x,y)] > D_-[(x,y)]$.

Two remarks. First, while this would minimize the expected mistakes, it does not discriminate between $p = 0.99$ and $p = 0.52$. Second, this is called $0-1$ loss, since we either have a loss of 1 if we make a mistake or 0 if we do not.

Absolute Loss

We will output a real value number $q \in [0,1]$. We identify $y = 0$ with the label $'-'$ and $y = 1$ with the label $'+'$. When the true label is y we lose $|y - q|$.

This implies that if the true label is 1 we lose $1 - q$, and it is 0 we lose q. Our expected loss is $p(1-q) + (1-p)q = p + (1-2p)q$. This implies that is $1 - 2p > 0$, i.e., $p < 1/2$ we would set $q = 0$ and if $1 - 2p < 0$, i.e., $p > 1/2$, we set $q = 1$. Essentially we got the same predictions as in the $0-1$ loss.

Quadratic Loss

In this model, the machine outputs a real number q. When the true label is y we lose $(y-q)^2$. Namely, the loss is $(1-q)^2$ for label '+', and q^2 for label '-'. The expected loss is $l(q) = p \cdot (1-q)^2 + (1-p) \cdot q^2$. What would be the optimal q? To find the minimal expected loss, we take the derivative: $\frac{dl(q)}{dq} = 2q(1-p) - 2p(1-q) = 2q - 2p = 0$ which implies that $p = q$. Taking the second derivative we verify that this is indeed a minimum: $\frac{d^2 l(q)}{d^2 q} = 2p + 2(1-p) = 2 > 0$ Therefore, using the quadratic loss function we minimize the loss by setting $p = q$.

Logarithmic Loss

We output a real number $q \in [0,1]$ and the loss is as follows:

- $-\log q$ for label '+'
- $-\log(1-q)$ for label '-'

Our total expected loss will be $l(q) = p(-\log q) + (1-p)(-\log(1-q))$. What would be the optimal q? To find the minimum loss, we take the derivative:

$$\frac{dl(q)}{dq} = \frac{-p}{q} + \frac{1-p}{1-q} = 0 \quad \Rightarrow \quad p = q$$

Again, setting $q = p$ we will achieve minimal expected loss. Note the difference between quadratic loss and logarithmic loss: Although both result in a choice of $q = p$, in logarithmic loss it is possible to experience infinite loss (when setting $q = 0$ or $q = 1$), whereas quadratic loss will result in the loss of at most 1 for any $q \in [0,1]$.

1.6 Estimating hypothesis error

Consider a specific hypothesis $h(x) \in \{0, 1\}$. We define the error of h to be $\Pr[h(x) \neq y]$. How can we estimate the error?

The most natural approach is to take a sample $S = \{(x_i, y_i)\}$ from the distribution D. We can now define the observed error:

$$\frac{1}{|S|} \sum_{i=1}^{m} I(h(x_i \neq y - i)$$

We would like to quantify how accurate is the observed error. We know that from the Laws of large numbers, that the observed error will converge to the true error in the limit when the sample size goes to infinity. In contrast, we want accuracy for a finite sample. An important issue for us would be the tradeoff between the approximation accuracy and the sample size. We will review a few classical concentration bounds from probability theory.

Markov Inequality

Let X be a non-negative random variable (r.v.) and $E[X]$ the *expected value* (mean) of X, then:

$$Pr[X \geq \alpha] \leq \frac{E[X]}{\alpha}$$

The proof follows since $\alpha Pr[X \geq \alpha] \leq E[X]$.

Chebyshev Inequality

Suppose that X is a random variable with variance $Var(X) = E(X - E[X])^2$, then:

$$Pr[|X - E[X]| \geq \beta] \ \leq \ \frac{Var(x)}{\beta^2}$$

The proof follows since $Pr[|X - E[X]| \geq \beta] = Pr[|X - E[X]|^2 \geq \beta^2]$ and applying Markov inequality.

Chernoff Inequality

Let $Z_1, \ldots, Z_m$ be a sequence of independent identically distributed (i.i.d.) Bernoulli random variables with $Pr[Z_i = 1] = p$. Then:

$$Pr[|\frac{1}{m} \sum_{i=1}^{m} Z_i - p| \geq \lambda] \ \leq \ 2e^{-2\lambda^2 m}$$

How to think about the bound. Suppose we set the sample size to be m. The desired accuracy parameter is ϵ. We want it to hold with high probability, namely, $1-\delta$. This implies that we have $\delta = e^{-2\lambda^2 m}$ or alternatively, $m = \frac{1}{2\epsilon^2}\log\frac{2}{\delta}$.

The proof is done by noting that for any $\eta > 0$,

$$Pr[\frac{1}{m}\sum_{i=1}^{m} Z_i - p \geq \lambda] = Pr[e^{\eta\sum_{i=1}^{m} Z_i - pm} \geq e^{\eta\lambda m}] \leq \frac{E[e^{\eta\sum_{i=1}^{m} Z_i - pm}]}{e^{\eta\lambda m}} = \left(\frac{E[e^{\eta Z_i - p}]}{e^{\eta\lambda}}\right)^m$$

where we are using the Markov inequality for the first inequality and the independence for the last identity. We now can minimize the expression as a function of η.

1.7 Hypothesis class

A hypothesis class is the set of hypothesis H that we will consider for the learning algorithm. We will distinguish between the *realizable* case, where we are guaranteed that the target function f is in H, while the *non-realizable* case we do not have this guarantee.

In the realizable case, if $f(x) = \sum_{i=1}^{d} \alpha_i x_i$ and H is the set of linear function, it is enough to find a set of d independent examples and solve. In case the solution does not have zero error, it implies that our assumption is incorrect. In the non-realizable case, we will look for the best linear function, but we are not guaranteed that it has zero error.

We will discuss a few possible hypothesis classes during the course.

1. **Hyperplane** A fairly simple linear model that splits the space to two parts depending on the sign of the linear function.

2. **Neural Networks (deep learning)** A circuit composed from a sequence of gates, each gate can be a hyperplane.

3. **Decision Trees**: builds a tree whose node are splitting the space.

An important feature of the hypothesis class is that the size of the hypothesis should be small (or unrelated) to the sample size. If, for example, we simply memorize the examples. We will get zero training error, but we will not have any generalization ability. This difference, between low training error and high test error, is many times called *overfitting*.

1.7.1 Probably Approximately Correct (PAC) model

We have an unknown distribution D over domain X (this means that every $x \in X$ is seen with probability $D(x)$), set of samples i.i.d. $S \subset X$, an unknown target function $f(x)$ that classifies each point $x \in X$, and a set of hypotheses H.

The Goal: We are looking for a hypothesis $h \in H$ that minimizes the **true error**:

$$\varepsilon(h) = \Pr_D[h(x) \neq f(x)] \stackrel{def}{=} \sum_{x \in X} D(x) \cdot I(h(x) \neq f(x))$$

Note: $\varepsilon(h)$ may be non-zero even for the optimal h, since we are not guaranteed that $f \in H$.

Problem: Had we known D, we could simply go over all $h \in H$ and find the hypothesis that minimizes $\varepsilon(h)$. However, we do not have this information, but only a *sample* S drawn i.i.d from X according to D.

Solution: we can minimize only the **observed error**:

$$\hat{\varepsilon}(h) = \frac{1}{|S|} \sum_{x \in S} I(h(x) \neq f(x))$$

Complexity versus Generalization

When choosing a model we have a tradeoff between hypothesis complexity and observed error. More complex hypotheses have lower observed errors, but may have higher true errors.

A complex hypothesis that has many degrees of freedom would use all of them in order to reduce the observed error. By doing this, it will adapt to noise or to the specific sample structure and not to the structure of the true underlying distribution. For a concrete example consider a domain of of $x \in [0, 1]$ and the hypothesis can be any finite set of intervals. Clearly, for any sample we can get zero training error by having a positive interval over each positive example. Now consider a joint distribution such that for $x \leq 1/2$ we have that with probability 0.9 the example is positive and with probability 0.1 it is negative. Similarly, for $x \geq 1/2$ we have that with probability 0.1 the example is positive and with probability 0.9 it is negative. The best hypothesis will predict positive iff $x \leq 1/2$. If we restrict to a single change, we will most likely find a very similar hypothesis. However, if we allow any number of changes, we will start increasing the error rate, as we add more intervals.

In order to avoid the "over-fitting" problem we would like to limit the complexity of the model and try to find a simple model even if it has a slightly higher observed error.

To compensate a simple model for its (possibly) higher observed error, we can place a penalty on complexity. We can do this by using one of the following criteria:

- **Minimum Description Length**

$$\hat{\varepsilon}(h) + \frac{|\text{code length of h}|}{m}$$

- **Structural Risk Minimization**

$$\hat{\varepsilon}(h) + \sqrt{\frac{\log |h|}{m}}$$

We can now minimize the error plus the penalty.

1.8 Bayesian Inference

In this model, our chosen function h is part of a (possibly infinite) class of functions (hypothesis) H, one of which *may* be f. Like before, given a sample S and a new point $x \in \mathbb{R}^d$ we would like to predict the correct label. Putting this in mathematical terms, we would like to calculate

$$\Pr[f(x) = 1|x, S]$$

Choosing to compute for the '1' label is arbitrary, knowing the probability for one label automatically gives us the probability for the other (why?).

Now, assuming $f \in H$ we can write this as

$$\Pr[f(x) = 1|x, S] = \sum_{h \in H} h(x) \Pr[f = h|S]$$

and using Bayes rule we have

$$\Pr[f = h|S] = \frac{Pr[S|f = h] \cdot \Pr(f = h)}{\Pr[S]}$$

Consider the terms in the equation.

$\Pr[S|f = h]$ denotes the probability of seeing the set S (instances and their labels) if h is indeed the correct classifier.

$\Pr[S]$ is independent of the classifier and therefore often acts as a normalizer and can generally be ignored.

$\Pr[f = h]$ is our assumption that a certain $h \in H$ is the correct classifier *before* we have seen the set S. This is called the *prior* distribution on H, and it should incorporate all our prior knowledge.

Sometimes we can assign probabilities to certain hypotheses even before we see the data, for instance, we will prefer simpler hypotheses to more complex ones. Many times we like to select a non-informative prior, such as a uniform distribution over H. The basic idea is that the sample will be much more influential than the prior. Clearly we can compute the exact probability, which is the best, but many times it is computationally infeasible. Two popular Bayesian methods that replace it are:

1. Maximum Likelihood (ML) In this method we select the $h \in H$ which maximizes the probability of seeing the set S; i.e., $\text{argmax}_{h \in H} \Pr[S|h]$. The *prior* distribution is ignored here.

2. Maximum *a-posteriori* (MAP) In this method we wish to find the most probable function h, given the set S. Namely, $\max_{h \in H} \Pr[h|S] = \max_{h \in H} \Pr[S|h] \Pr[h]$.

These two methods are identical when the *prior* distribution is uniform.

1.9 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 $majority(y_{[1]}, y_{[2]}, y_{[3]})$. Note that 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.

For nearest neighbor the actual values of attributes is critical, for this reason it is important to normalize the attribute. The actual normalization would depend on the marginal distribution of the attribute. The goal is to have the value "unit less". A few examples.

1. Gaussian: replace x by $(x-\mu)/\sigma$ where μ is the average and σ is the sandars deviation.

2. Uniform: replace x by $\frac{x-x_{min}}{x_{max}-x_{min}}$ where x_{min} and x_{max} are the minimum and maximum value.

3. Exponential: replace x by x/λ.

1.10 Unsupervised learning

Unsupervised learning deal with the case where there are no labels. It can be either that the labels do not exists (e.g., user's behavior) or not observable. Mathematically the goal is ill define: You can cluster in many ways.

In general the goal is to better understand the data. Many times this is done by segmenting the data to informative clusters. Less emphasis is stress on generalization.

Some of the methods is to assume a generative model of the data and to try and recover the parameters of the model under that assumption. We will review one such algorithm, k-means.

1.11 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

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|_2^2$$

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, $C_j^t = \arg\min_i \|x_j - \mu_i^t\|_2^2$ 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].

1.12 Singular value Decomposition (SVD)

Consider a recommendation setting, where each user is identify with a short vector u from $\mathbb{R}^d$ and each item is identify also with a vector v from $\mathbb{R}^d$. The fitness of item v to user u is simply the inner product $u \cdot v$.

This implies that we have a user matrix U, which the rows are the user vectors, and an item matrix $V^\top$ where the columns are item vectors. The matrix $M = UV^\top$ is the fitness of items to users.

We observe M (or actually only a fraction of M). We can perform an SVD, which rewrites M as $UDV^\top$, where D is a diagonal matrix. By reducing D (eliminating low values) we get a low rank matrix and the desired U and $V^\top$.

1.13 Principal Component Analysis (PCA)

Our high level goal is to reduce the dimensionality of the problem. The goal is to find a basis with the most informative basis vectors. This is done by taking the highest eigenvalues and their corresponding vectors. Let U be the matrix of the d' highest eigenvalues vectors. Then we map x to $U^\top x$. We would like to minimize the reconstruction error. From $z = U^\top x$ we reconstruct Uz. This implies that we like to minimize $\|x - UU^\top x\|$ such that U is an orthonormal matrix.

The PCA can be many times helpful in presenting the data and understanding the data.

1.14 Structure of the Course

1. Introduction
2. PAC: Generalization
3. PAC: VC bounds
4. Perceptron: linear classifier
5. Support Vector Machine SVM
6. Kernels
7. Stochastic Gradient Decent
8. Boosting
9. Decision Trees
10. Regression and PCA
11. Generative Models, EM
12. Clustering, k-means
13. TBD

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)