Foundations of Machine Learning (CS725)

Instructor: Saketh

Contents

Chapter 1

Basic Concepts and Definitions

Machine learning aims at developing algorithms that mimic the ability in humans to *learn* i.e., improve their "performance" with experience. By performance, we mean their various cognitive abilities. A short note about this is presented below. It is easy to observe that machine learning algorithms will have far reaching consequences in all aspects of living and hence are worth developing.

1.1 Human Learning

Humans seem to have various cognitive abilities. Some of which[1] are: i) finding similarities or patterns or groupings in data, ii) categorizing objects not seen before as novel iii) sequence completion iv) recognition abilities including speech and object recognition v) Summarizing or abstracting text/images/multi-media vi) Decision making vii) Problem solving etc. A little thought will convince that all these abilities improve[2] with increasing experience.

Above discussion convinces that associated with learning there is always a target/unknown concept/ability. Hence-forth, we will call this as the **unknown-concept**. For e.g., in the phenomenon of learning to group data, the unknown concept is the relationship between data/objects and group-ids. In the phenomenon of speech recognition, it is the relationship between speech utterances and English transcriptions etc.

Needless to say, learning happens through experience. Hence **experience** is an important aspect of learning. It is easy to see that **experience** is simply a finite-sized realization (sampling) of the truth in the **unknown-concept**. Depending on

[1]Examples are picked based on the settings that machine researchers have formally studied.

[2]Not necessarily strictly monotonically improving.

the need/application humans express[3] the **unknown-concept** through some **input-output** pairs. For e.g., in the grouping/clustering of data example, the input is the datum and the output is the group-id etc. This clarifies what we mean by **input** and **output**.

Also, how well or fast will a person learn will definitely depend on his current state/capacity/bias/background. We will refer to this as **background** hence-forth. Given this terminology, one could say that the objective in human learning is to determine the **unknown-concept**, but it seems enough to say that the goal is to be able to provide the "right" output for *any* input (because this is how usually humans express their ability/unknown-concept that has been learnt). Summarizing, here are the five important aspects in human learning:

1. **Input** and **Output**
2. **Unknown-concept**
3. **Experience**
4. **Background**
5. **Objective of learning**

1.2 Machine Learning

Though humans possess very many abilities, they are currently far from understanding how they learn/acquire/improve these abilities. So the idea in machine learning is to develop mathematical models and algorithms that mimic human learning rather than understanding the phenomenon of human learning and replicating it.

The formal study of machine learning begins by restricting oneself to certain limited aspects in human learning and postponing the mimicing of human learning in entirety to a later stage. Accordingly we study the follow basic types of learning, which are categorized based on the type of supervisor:

Supervised Learning: This concerns the cases of human learning where the supervisor/supervision is specific/specialized to the goal at hand.

Passive version: mimics learning that happens in babies when taught by *maata-pita*. i.e., learning through specific examples (and non-examples):

[3]If humans could express the unknown-concept directly, rather than in terms of input-output pairs, then perhaps, humans would also understand how they learn!

Classification: E.g., the parents show the baby pictures of various objects while clarifying the name of each object. After being shown a few, the baby starts identifying those objects.

Regression: E.g., everyday (based on few environmental clues) the parents make a prediction about the chance that it will rain (based on which they may advice their kids not to go out to play :). After few days, the kids themselves start making these predictions.

Active version: This concerns the learning that happens in a *shishya* when taught by an *aachaarya*. E.g., After some passive supervised learning, the *shishya* asks questions about the most confusing examples to be clarified by the *aachaarya*. Note that here the choice of example is with the learner rather than the supervisor. This is more popularly known as "Active Learning".

Un-supervised Learning: This concerns the cases of human learning where the supervisor/supervision is not specific/specialized to the goal at hand.

Passive version: E.g., Based on various sentences (spoken in mother tongue) that various people speak to a kid, the kid forms a idea of his language. Then he can predict whether a word, which was never heard by him earlier, to belong to his language or not. Note that here the supervisor is not specific and more importantly, the supervision (the sentences spoken) is not explicitly intended to teach the kid what is his language's formal definition/grammar. Other examples are problems of support/mean/density estimation.

Active version: E.g., Based on actually touching hot and cold water multiple times (and perhaps getting hsi fingers burnt sometimes), the baby figures out the right temperature range of water that is "safe" for him. This kind of learning is popularly known as "Reinforcement Learning". This will not be covered in this course[4].

Now we shall write down the mathematical concepts by which we represent each of the five aspects in human learning mentioned earlier.

1. **Input** by $x \in \mathcal{X}$ ($\subset \mathbb{R}^n$, usually) and **Output** by $y \in \mathcal{Y}$ ($\subset \mathbb{R}$, usually). $\mathcal{X}, \mathcal{Y}$ are called as input space and output space respectively.

2. **Unknown-concept** by a Probability distribution [Ross, 2002] over the inputs (hence-forth denoted by F_X^U) or over the input-output pairs (hence-forth

[4]There are many other learning settings that are formally studied by ML researchers and will not be studied in this course.

denoted by F^U_{XY}). For e.g., in case of support/mean/density estimation or sequence filling problems the probability distribution is over the inputs alone and in case of object recognition, language identification applications described above, the probability distribution is over the input-output pairs.

3. **Experience** by:

   (a) set of input-output pairs. This is the case for e.g., in **supervised learning**.

   (b) set of inputs. This is the case for e.g., in **unsupervised learning**.

   In either case, we call this set as the **training set**. Most importantly, the training set and the unknown distribution have a relation: we assume that the training set is an instantiation of a set of iid random samples from the unknown distribution. We denote the set of iid random variables generating the training set as $D = \{X_1, \ldots, X_m\}$ in un-supervised case and as $D = \{(X_1, Y_1), \ldots, (X_m, Y_m)\}$ in the supervised case. The training set is an instantiation of D (hence-forth denoted by $\mathcal{D}$).

4. **Background** by (mathematical) Model. We will give examples later.

5. **Objective of learning** by some mathematical statement. For e.g. contruct $f : \mathcal{X} \mapsto \mathcal{Y}$ such that f satisfies certain mathematical condition(s). Examples later.

Chapter 2

Examples of Machine Learning Models/Algorithms

We then began by giving examples of some machine learning models/algorithms. We began with the simplest, and yet perhaps the most powerful and generic, nearest neighbour classifier, which perhaps models the way in which humans identify objects.

2.1 Nearest Neighbour Classifier

Please refer section 13.3 in Hastie et al. [2009], which gives a nice overview of this method.

Here the model is extremely simple: the idea is to remember/store the entire training data and when a (new) input is given, search for the nearest input in the training data and assign the label of the (new) input as that of this nearest input.

We began by analyzing this model/classifier formally. The formal analysis is due to Cover and Hart [1967][1]. The key result from this work is, under mild conditions and as $m \to \infty$ (m is number of training examples), we have

$$0 \leq R^* \leq R^{NN} \leq R^* \left(2 - \frac{c}{c-1} R^*\right),$$

where R^{NN} denotes the expected misclassification error of the NN classifier, in the limit and c is the number of classes. We also commented on two extreme cases: i) if $R^* = 0$, then the bounds are tight and $R^{NN} = 0$. Algorithms that achieve Bayes

[1] Available at `http://web.stanford.edu/~montanar/TEACHING/Stat319/papers/cover_nn.pdf`

error are said to be Bayes consistent. Moreover, if $R^* = 0.5$ (supervisor is clueless), then $R^{NN} = 0.5$ (so will be the learner).

We then thought about an improvement for reducing the chance of picking a low probability label from the neighbour. The obvious idea was to look for some k nearest neighbours instead of one and take the majority vote. And because for large k, we expect majority vote to be the same as the more probable class, the error might reduce to R^*. Hence we started analysing so called k nearest neighbour classifier. We intuitively argued that $m \to \infty, k \to \infty, \frac{k}{m} \to 0$ will be the unconditions under which error will reduce to R^*. This intuitive result is formally stated in the following theorem (proof skipped) due to Devroye and Gyorfi [1985][2]:

Theorem 2.1.1. *If $\mathcal{X} \subset \mathbb{R}^n, \mathcal{Y} = \{-1, 1\}$ and under the conditions $k \to \infty, \frac{k}{m} \to 0$, we have with atleast $1 - \delta$ probability,*

$$R_m^{kNN} - R^* \leq \sqrt{\frac{72\gamma^2 \log \frac{2}{\delta}}{m}},$$

where R_m^{kNN} denotes the expected probability of misclassification with k-NN classifier trained with m examples and $\gamma \leq \left(1 + \frac{2}{\sqrt{2-\sqrt{3}}}\right)^n - 1$.

More interestingly, we were able to say something about the finite k binary classification case, which is formalized in the following bound[3]:

$$0 \leq R^* \leq \ldots R^{(2k+1)NN} \leq R^{(2k-1)NN} \leq \ldots \leq R^{3NN} \leq R^{NN} \leq 2R^*(1 - R^*),$$

where R^{kNN} denotes the limiting value of the expected misclassification error with the k-NN classifier, as $m \to \infty$.

Motivated with the ideology of kNN (of estimating $F_{Y/X}$), we study other (parametric) methods for estimating distributions in section 2.5.

2.2 Decision Trees

After modelling the simplest case of human learning via the nearest neighbour algorithm, we then wanted to model human learning where humans try to estimate a single threshold beyond which inputs belong to one category and below which they belong to the other category. For example, estimating the critical threshold for the

[2]Please refer chapter 11, theorem 11.1, in Devroye et al. [1996] for a detailed proof.

[3]Please refer chapter 5, theorem 5.4, in Devroye et al. [1996] for an insightful proof

amount of water-vapour rising out (in unit time) from a bucket of water, beyond which the water is too hot to touch. We then began by modelling this situation as follows:

Let $\phi(x) \in \mathbb{R}$ denote the representation of input datum x. In the above example, x is the bucket of water and $\phi(x)$ is the amount of water-vapour rising out. We represented the human by the set of positive reals (the candidates for the threshold). This is what is the "model" in this case. Then the learning algorithm should essentially figure out what is the '$b \in \mathbb{R}^+$' such that $\{x \mid \phi(x) \geq b\}$ gives the cases where bucket of water is hot (and vice-versa).

We then tried to imagine how a human would estimate such a threshold b and the obvious answer was b will be the optimal solution of the following optimization problem[4]:

$$\min_{b \in \mathbb{R}^+} \frac{1}{m} \sum_{i=1}^{m} 1_{\{y_i(\phi(x_i)-b) \geq 0\}}.$$

We also discussed a simple algorithm to solve the above optimization problem that has a computational complexity $O(m)$.

Now came the question why does this algorithm work? In the sense, why (or in which cases) will the b picked by this algo be the "best" b in the sense that it achieves the least (true but unknown) probability of misclassification. The obvious justification was that the objective in the above optimization will approach the true probability of misclassification as $m \to \infty$. And hence, as argued in section 3, the optimal solution of the above problem will approach the "best" threshold.

We then naturally motivated that there could be multiple features $\phi_1, \ldots, \phi_n$ available. One way to utilize all of them while staying close to the above algorithm is as follows; more commonly known by the name "decision trees":

1. The "best" b_1 corresponding to ϕ_1, as per the training set is obtained by solving the optimization problem.

2. The dataset is then partitioned into two, one where ϕ_1 exceeds b_1 and one otherwise. If ϕ_1 is not "good enough", then none of these partitions may be "pure" (i.e., it may have inputs of more than a single class).

3. Then the same procedure of finding threshold and partition is repeated.

It is easy to see that this can be nicely represented as a tree (refer fig.9.5 in Hastie et al. [2009]). While this constitutes the training phase, the inference phase is pretty simple: just run the example through the decision tree and assign label as

[4]Assuming $\mathcal{Y} = \{-1, 1\}$ (binary classification).

the majority label in the leaf node reached. It is easy to see that decision tree simply combines the various features in the form of conjunctions over propositions involving the features. Each path from root to leaf node gives a conjunction that implies a particular label.

While this seems intuitive, again the question is why will this work? It is easy to see that this would work as long as there are enough examples at every stage to estimate the corresponding b_i. Since the training set size for the threshold decreases with increasing depth of the tree, we desire shallow trees giving good purity in the leaf nodes. In practice, (if one does not derive careful learning bounds), the number of levels should be pre-fixed before seeing the dataset (for reasons discussed in section 3).

2.3 Support Vector Methods

2.3.1 Binary Classification

The main reference for this section is chapter 4 in Mohri et al. [2012].

A mathematically convenient way to combine the information in the various features $\phi_1, \ldots, \phi_n$ is by looking at linear models: appropriately thresholded linear combinations of features. More specifically, consider the model

$$\mathcal{F}_L = \left\{ f \mid \exists w = [w_1 \ldots w_n]^\top \ni f(x) = \text{sign}\left(w^\top \phi(x) - b\right) \ \forall\, x \in \mathcal{X} \right\},$$

where $\phi(x) = [\phi_1(x) \ldots \phi_n(x)]^\top$, and while w determines the linear combination, b determines the threshold. In this case, the problem of learning (training phase) is simply the problem of finding the right $w \in \mathbb{R}^n, b \in \mathbb{R}$. Though it appears that there are $n+1$ unknowns, a closer look will reveal that only n of them need to be optimized for. One of them can always be fixed arbitrarily. This is because, for a given w, b pair, any $\alpha w, \alpha b$ will also give the same classification.

Given this, training can simply be posed as this optimization problem:

$$\begin{aligned} \min_{w \in \mathbb{R}^n, b \in \mathbb{R}} \quad & \tfrac{1}{m} \textstyle\sum_{i=1}^m 1_{\left\{ y_i \left(w^\top \phi(x_i) - b \right) \geq 0 \right\}}, \\ \text{s.t.} \quad & \|w\| = 1 \end{aligned} \tag{2.1}$$

Note that the constraint $\|w\| = 1$ simply fixes one of the $n+1$ variables as discussed above. It so happens that unless the above optimization problem has an optimal value of zero (i.e., there exists a hyperplane that clearly separates the two kinds of inputs), it is computationally hard to solve it. Please see Feldman et al. [2009] for

further details. In the case, there is clear separation, the above problem can infact be written as the following linear program and hence can be efficiently solved:

$$\begin{aligned} \min_{w\in\mathbb{R}^n, b\in\mathbb{R}} \quad & 0, \\ \text{s.t.} \quad & y_i\left(w^\top\phi(x_i) - b\right) \geq 1 \;\forall\; i = 1,\ldots,m. \end{aligned} \tag{2.2}$$

We then noted that the separation/margin in the above set up is the distance between the hyperplanes $w^\top\phi(x) = b-1$ and $w^\top\phi(x) = b+1$, which is equal to $\frac{2}{\|w\|}$. The above optimization problem will invariably have many solutions (once there is some separation, there is another solution with lesser separation). Of these, the one that achieves the maximum margin is given by[5]:

$$\begin{aligned} \min_{w\in\mathbb{R}^n, b\in\mathbb{R}} \quad & \tfrac{1}{2}\|w\|^2, \\ \text{s.t.} \quad & y_i\left(w^\top\phi(x_i) - b\right) \geq 1 \;\forall\; i = 1,\ldots,m. \end{aligned} \tag{2.3}$$

Once this is in place, it is easy to (re)write the original problem (where no. errors need not be zero; the generic case) as follows:

$$\begin{aligned} \min_{w\in\mathbb{R}^n, b\in\mathbb{R}, \xi\in\mathbb{R}^m} \quad & \tfrac{1}{m}\textstyle\sum_{i=1}^m 1_{\xi_i>0}, \\ \text{s.t.} \quad & y_i\left(w^\top\phi(x_i) - b\right) \geq 1-\xi_i, \xi_i \geq 0 \;\forall\; i = 1,\ldots,m. \end{aligned} \tag{2.4}$$

We already know that the above problem is computationally hard and hence one way out is to "relax" it to the following:

$$\begin{aligned} \min_{w\in\mathbb{R}^n, b\in\mathbb{R}, \xi\in\mathbb{R}^m} \quad & \tfrac{1}{m}\textstyle\sum_{i=1}^m \xi_i, \\ \text{s.t.} \quad & y_i\left(w^\top\phi(x_i) - b\right) \geq 1-\xi_i, \xi_i \geq 0 \;\forall\; i = 1,\ldots,m, \end{aligned} \tag{2.5}$$

which can be re-written as the following by eliminating ξ variables:

$$\min_{w\in\mathbb{R}^n, b\in\mathbb{R}} \quad \frac{1}{m}\sum_{i=1}^m l\left(y_i\left(w^\top\phi(x_i) - b\right)\right), \tag{2.6}$$

where $l(z) \equiv \max(0, 1-z)$, and is known as the hinge loss function[6]. We then noted that this function is convex and is infact an upper bound on $1_{y\neq g(x)}$, that simply records a misclassification. The later is also called as the 0-1 loss. We then noted the following convex surrogate losses for the 0-1 loss:

Hinge-loss: $l(z) \equiv \max(0, 1-z)$.

[5]This is popularly known as hard-margin Support Vector Machine

[6]Measures the "loss" incurred on replacing true label with estimated one

Square-Hinge-loss: $l(z) \equiv (\max(0, 1-z))^2$.

Logistic-loss: $l(z) \equiv \log(1+e^{-z})$.

Exponential-loss: $l(z) \equiv e^{-z}$.

While all of them are convex and upper bounds for 0-1, the first two do not penalize "correct" classifications, whereas the later two are more "pessimistic" and demand very high value of $yg(x)$. Except the first, all are differentiable. In plain English words, the problem (2.6) is simply minimizing the average loss over the training set[7]. Average loss over the training set is sometimes referred to as the empirical (or sample) risk. Hence (2.6) is minimizing the empirical risk.

Finally, in case we wish to write down something like maximum margin classifier that minimizes loss over the training set, then we noted that these two objectives oppose each other (for sensible loss functions) and hence the model should pre-specify what the trade-off parameter is:

$$\min_{w \in \mathbb{R}^n, b \in \mathbb{R}} \frac{1}{2}\|w\|^2 + \frac{C}{m}\sum_{i=1}^{m} l\left(y_i\left(w^\top \phi(x_i) - b\right)\right), \tag{2.7}$$

where C is the trade-off parameter. More popularly, C is called as the "hyper-parameter" or "regularization parameter". We will hence-forth denote (2.6) by LC and (2.7) by MMLC.

MMLC with hinge-loss is popularly known as Support Vector Machine (SVM). MMLC with squared-hinge-loss is known as l_2-SVM. MMLC with Logistic-loss is known as Logistic regression (also refer probabilistic models).

We then began studying the characteristics of these two optimization problems i) minimize empirical risk (average loss over training set) ii) minimize empirical risk (average loss over training set) while maximizing margin. We began with an easy to prove, but insightful, theorem called the representer theorem:

Theorem 2.3.1. *All optimal solutions, w^*, of MMLC i.e., (2.7) satisfy the following property: there exists $\alpha = [\alpha_1 \ \ldots \ \alpha_m]^\top \in \mathbb{R}^m$ such that $w^* = \sum_{i=1}^m \alpha_i y_i \phi(x_i)$. Also, there exists an optimal solution of LC i.e., (2.6) satisfying the same property.*

While this was easy to prove (see Theorem 5.4 in Mohri et al. [2012]), it gives the following insights already:

[7]This is "fine" because as $m \to \infty$, the average loss will approach the true (but unknown) expected loss. Hence (2.6) will approximately solve the problem of minimizing expected loss. Expected loss is sometimes referred to as "risk".

- LC and MMLC provide modified nearest neighbour classifiers.
- When used with "sparse" losses like hinge and square hinge loss, they become "smart" nearest neighbour classifiers in the sense that the prediction depends on few training examples alone[8].
- During training as well as inference, we do NOT explicitly need ϕ. It is enough if an oracle for computing $\phi(x)^\top \phi(z)$ exists for all $x, z \in \mathcal{X}$.

We analyzed the optimality conditions and the dual[9] of MMLC in more detail. With the notation $l_i(z) = l(y_i z)$ and l_i^* denoting the conjugate (or Fenchel dual or Legendre transform[10]) of l_i, we have the dual problem as:

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^m} \quad & -\tfrac{1}{2}\alpha^\top G\alpha - \tfrac{C}{m}\textstyle\sum_{i=1}^m l_i^*\left(\frac{-m\alpha_i y_i}{C}\right), \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^m \alpha_i y_i = 0, \end{aligned} \tag{2.8}$$

where G, called the gram matrix, whose $(i,j)^{th}$ entry is $y_i y_j k(x_i, x_j)$, and $k(x_i, x_j) \equiv \phi(x_i)^\top \phi(x_j)$. Moreover, the optimality conditions, with hinge-loss[11], turn out to be (w^*, b^* is optimal solution of MMLC and α^* is that of its dual):

- $w^* = \sum_{i=1}^m \alpha_i^* y_i \phi(x_i)$
- $\alpha_i^* = 0 \Rightarrow y_i\left((w^*)^\top \phi(x) - b^*\right) \geq 1,\ y_i\left((w^*)^\top \phi(x) - b^*\right) > 1 \Rightarrow \alpha_i^* = 0$
- $0 < \alpha_i^* < \frac{C}{m} \Leftrightarrow y_i\left((w^*)^\top \phi(x) - b^*\right) = 1$ (such are called as non-bound support vectors)
- $\alpha_i^* = \frac{C}{m} \Rightarrow y_i\left((w^*)^\top \phi(x) - b^*\right) \leq 1,\ y_i\left((w^*)^\top \phi(x) - b^*\right) < 1 \Rightarrow \alpha_i^* = \frac{C}{m}$ (such are called as bounded support vectors)

Hence we expect that with hinge loss, most of the training examples will have $\alpha_i^* = 0$ at optimality. Infact, one can show theorem 4.1 in Mohri et al. [2012].

[8] So there is no need to store the entire dataset

[9] Please refer sections 5.1.6 and 5.2 in Boyd and Vandenberghe [2004] for the generic methodology for writing the dual in terms of Conjugate. The same sections give many examples too. More specifically, example 5.4 in the same book gives back the exact derivation we did in the lectures.

[10] See section C 6.3 in Nemirovski [2005] for details about the Conjugate of a function.

[11] The very same conditions can also be derived by employing KKT conditions. Refer section 4.3.2 in Mohri et al. [2012] for such a derivation. Note that in lectures we came up with an alternate argument (based on conjugate), but ofcourse giving exactly the same optimality conditions. Also, the dual for the hinge-loss can also be derived as shown in section 4.3.3 in Mohri et al. [2012].

Exploiting this sparsity in the optimal solution is the key idea behind state-of-the-art methods for solving MMLC (when hinge-loss kind of "sparse losses" are employed). Broadly there are two (related) methods:

- Co-ordinate descent: minimization at every iteration is done only wrt. one (or few) variable(s). Rest are fixed at previously updated values. Since solution is known to be sparse, we expect to converge in few iterations.

- Active-set method: minimization at every iteration is done wrt. variables in active set. Rest are fixed at zero (non-active). Active set is updated to better guesses for the non-zero variables at optimality. Again, because of sparsity we expect to converge even before seeing all the variables once!

For the case hinge-loss with $b = 0$, the dual becomes an unconstrained problem[12] and the state-of-the-art for solving it is indeed co-ordinate descent. Please refer Hsieh et al. [2008] for details. For the case b is not fixed at zero (and is a variable), a slight modification of co-ordinate descent is employed, which is called as sequential minimal optimization (SMO). This is because the problem has a constraint and hence the iterates will become infeasible if vanilla co-ordinate descent is employed. Please refer to Keerthi et al. [2001] for details.

2.3.2 Ordinal Regression

The main reference for this section is Chu and Keerthi [2005]. Ordinal regression or ordinal classification is the setting where the output space is a set of multiple classes that form a total order. Hence the idea is to look for k thresholds in the linear combination score $w^\top \phi(x)$, if $k+1$ ordinal classes exist; everything else remains the same as in the binary classification case. Refer my quiz-1 solutions[13] for details on modifying nearest neighbours, decision trees and SVMs to handle this.

2.3.3 Regression

We then went on to look at the extreme case of ordinal regression, where the output space in the special total order called the real numbers. This setting is known as Regression and the main reference for the following is section 10.3 in Mohri et al. [2012]. More specifically:

[12] Infact, if one uses the feature vector as $[\phi(x)^\top \ 1]^\top$ instead of $\phi(x)$, then one can safely assume $b = 0$. Since then b is a part of w itself.

[13] Available at `http://www.cse.iitb.ac.in/saketh/teaching/cs725Quiz1Sols.pdf`.

Least-square Regression: Section 10.3.1 in Mohri et al. [2012]. Eqn (10.9), (10.11) respectively provide the formulation and optimality condition. This is LC with square-loss.

Ridge-Regression: Section 10.3.2 in Mohri et al. [2012]. Eqn (10.15), (10.17) respectively provide the formulation and optimality condition. This is MMLC with square-loss.

Support Vector Regression: Section 10.3.3 in Mohri et al. [2012]. Eqn (10.23), (10.26) respectively provide the formulation and its dual. This is MMLC with ϵ-insensitive loss (defined in the book).

Alternative loss functions are summarized in Fig 10.5.

Performance measures for regression appear here 3.4.3.

2.3.4 Structured Prediction

We then took the more generic setting where the classes need not have a total order relation defined over them. For example, there could be a partial order defined over the classes inducing a tree or lattice structure on the output space. An example for this is the case where the classes form a Taxonomy (e.g., taxonomy of CS subjects). Another example is the problem of sequence labeling, where the goal is to label every term in the input sequence (e.g., speech recognition). The machine learning setting where the output space has a well-defined (but perhaps complex) structure (as in above examples) is popularly refered to as the problem of Structured Prediction. The main reference[14] for this section is Tsochantaridis et al. [2005] (`struct-SVM`)[15]. The formulation we studied in lecture is given by eqn. (7) in the paper. Its dual is given in proposition 5. Note that this dual is very similar to that of regular SVM and shares similar "sparsity" properties. Hence one can again use co-ordinate descent algorithm[16]. However, later on when we cover sequence labeling problem in section 2.3.4.1, we will realize that co-ordinate descent is not good enough and one may need something more smarter, which does not even need to loop through all the dual variables. Such an algorithm is described in Algorithm 1 in the paper.

We covered a few simple examples where we applied `struct-SVM` to multi-class classification. Refer sections 4.1 and 4.2 for those examples.

[14] You may also refer to section 19.7.2 in Murphy [2012].

[15] Section 8.5 in Mohri et al. [2012] gives a broad overview of alternative structured prediction methodologies (rather than struct-SVM) for the multi-class classification problem.

[16] Nevertheless one may need to update atleast two dual variables in each iteration

2.3.4.1 Sequence Labeling

We now focus on the problem of sequence labeling, where the input space as well as output space is a set of sequences [17]. Again, for simplicity sake we assume elements of the sequence are Euclidean vectors.

For e.g., consider the problem of character recognition or speech recognition. The input in the earlier case is an image, which can be thought about as a sequence of images segmented at character level. In turn these character level sub-images can be represented in some vectorial form using image processing transforms. The input in the later case is a speech signal, which can be further segmented and processed into sequences of MFCC vectors[18]. In either case, it seems convinient to represent the input as an arbitrary lengthed sequence of Euclidean vectors. The output in both cases is sequence of characters that are represented in the corresponding input sequence. Let us denote an input sequence x of length T by $\left(x^{(1)}, \ldots, x^{(T)}\right)$, where each $x^{(i)} \in \mathbb{R}^n$ and the corresponding output sequence by $y = \left(y^{(1)}, \ldots, y^{(T)}\right)$, where each $y^{(i)} \in \mathcal{A}$. Here $\mathcal{A}$ is the set of alphabets of the language in question. Let us denote the set of all input sequences of all possible lengths by $\mathcal{X}$ and that of output sequences by $\mathcal{Y}$.

In the above applications, the machine learning problem is to induce a function $f : \mathcal{X} \mapsto \mathcal{Y}$, given a training set containing m input-output pairs[19], which when evaluated on an input sequence of length T will evaluate to the "best" output sequence of the same length. At this point we refer the reader to section 2.5.2.4 and present a specific choice for $\phi(x, y)$ in struct-SVM, which will mimic the discriminative model for sequence labeling given in (2.13).

This specific choice for $\phi(x, y)$ is discussed in detail in section 4.3 in Tsochantaridis et al. [2005]. While this seems fine, it is clear that the number of dual variables with struct-SVM for sequence labeling will be $O(\sum_{i=1}^{m} c^{T_i})$, where c is the number of values each label in the sequence can take and T_i is the length of the i^{th} sequence. Hence employing the usual co-ordinate descent method will pose computational challenges.

The alternative is to employ an active set algorithm. The idea in an active set algorithm is to restrict the non-zero dual variables to those in the active set, which itself is updated at every iteration. The hope is that the final active set size is not very large compared to the actual number of non-zeros at optimality. Hence we

[17] of finite but arbitrary length, as opposed to fixed length in case of Euclidean vectors

[18] Please see `http://en.wikipedia.org/wiki/Mel-frequency_cepstrum` for a primer on MFCC.

[19] sampled from the unknown distribution

never need to solve an optimization problem larger than the active set[20]. The details of such a clever active set algorithm are presented in algorithm 1 of Tsochantaridis et al. [2005]. Theorem 18 in the same paper, proves that the algorithm's complexity is polynomial. Notice that this algorithm would require one to solve an argmax problem. This could be done using a modified Viterbi algorithm[21]. The details are provided in section 4.3.2 of the paper.

2.3.5 Unsupervised Learning

There are various unsupervised learning problems that can be handled using SVM-kind of ideas: density estimation, high density region estimation (and hence clustering and novelty detection).

2.3.5.1 Density Estimation

One way of parameterizing density functions (f) is by using $f(x) = \sum_{i=1}^{m} \alpha_i f_i(x)$, where $\alpha_i \geq 0, \sum_{i=1}^{m} \alpha_i = 1$ and $f_i(x)$ are "basic density functions" centered at the i^{th} training example. For instance, $f_i(x) \propto \exp\{-\|x_i - x\|^2\}$ (see also section 2.5.1.2). Please read Vapnik and Mukherjee [2000] for details of how the problem of optimizing/learning the parameters α can be posed as an SVM-like formulation (given by equations (17)-(19) in the paper).

2.3.5.2 High Density Region Estimation: Clustering & Novelty Detection

Sometimes, one is not interested in estimating the full density function, but only interested in estimating regions in the domain that have high density. Such an estimation problem is useful for various applications:

Clustering: If clusters are defined as regions of high density, the classical problem of clustering is the same as that of high density region estimation.

Novelty Detection: If rare/novel datapoints are annotated as those being sampled from low density regions, then again novelty detection problem can be solved by estimating the high density regions (complement of whom will give low density regions).

[20] And the sice of active set is potentially far less than $O(\sum_{i=1}^{m} c^{T_i})$.

[21] Viterbi algorithm is introduced in section 2.5.2.4 below.

The key idea is to let $w^\top \phi(x)$ represent a score proportional to the unknown density and then estimate w. Since training set is nothing but a sample from this unkown density function, the information we have for estimating w is that: $w^\top \phi(x)$ is high, say ≥ 1,[22] for most of the training datapoints[23]. To this end, we have the following:

$$\begin{aligned} \min_{w \in \mathbb{R}^n, \xi \in \mathbb{R}^m} \quad & \tfrac{1}{2}\|w\|^2 + \tfrac{C}{m}\textstyle\sum_{i=1}^m \xi_i, \\ \text{s.t.} \quad & w^\top \phi(x_i) \geq 1 - \xi_i,\ \xi_i \geq 0,\ \forall\ i. \end{aligned} \tag{2.9}$$

The above formulation is popularly known as the one-class SVM (for obvious reasons).

Alternatively, one can also come up with a ν-SVM variant of the above, which is detailed in equation (4)-(5) in Scholkopf et al. [2001]. Interstingly this is also equivalent to the tightest enclosing hypersphere problem, detailed in equation (13) of the same paper.

Once training is done, inference is very simple, a datapoint x is from high density region if $w^\top \phi(x) \geq 1$ and else otherwise.

Finally, Ben-Hur et al. [2001] provides details of algorithm for using the inference from a one-class SVM to identify datapoints belonging to different clusters (see section 2.2 in the paper).

2.4 Kernel Methods

Chapter 5 in Mohri et al. [2012] is a good reference for this section.

In the previous section, we assumed features ϕ were given. But thanks to representer theorem, we know that both for training, as well as prediction, we only require dot products between the feature vectors $k(x, z) \equiv \phi(x)^\top \phi(z)$.

We then studied what properties should $k : \mathcal{X} \mapsto \mathcal{X}$ satisfy if it indeed represents the dot product between feature vectors[24]. Given a set of m points $Z_m = \{z_1, \ldots, z_m\}$, let K denote the matrix whose $(i, j)^{th}$ entry is $\phi(z_i)^\top \phi(z_j)$. If k represents a dot product between feature vectors, then it is clear that $K_m \succeq 0\ \forall Z_m,\ \forall m \in \mathbb{N}$ (i.e., K_m is symmetric and positive semi-definite).

[22] Ofcourse, $w^\top \phi(x) \geq 1$ makes sense only if the scale of w is fixed, which can be done by restricting $\|w\|$.

[23] we do not insist on density score being high for all because we know that there is a small chance that the samples are actually points of low density.

[24] Here the question is NOT what are the properties of dot-product, which we know.

We then asked if the converse were true and to that end, we discussed in detail Theorem 5.2 in Mohri et al. [2012] and its proof. The theorem not shows guarantees the existence of a Hilbert space and a feature mapping onto it, but also illustrates a special one called as the Reproducing Kernel Hilbert Space (RHKS). The definition of RKHS is in the theorem statement itself.

Encouraged by these observations we define: a function $k : \mathcal{X} \mapsto \mathcal{X}$ satisfying the condition that all the gram-matrices (of all sizes) are symmetric and positive-semi definite, is called as a kernel over $\mathcal{X}$.

We discussed many properties of kernels:

- Conic combinations of (finite number of) kernels is a kernel.
- Product of (finite number of) kernels is a kernel.
- If a sequence of kernels $k_1, k_2, \ldots, k_r, \ldots$ converges to a function k, then k is a kernel.
- If k is a kernel, then so is $\hat{k}(x, z) \equiv \frac{k(x,z)}{\sqrt{k(x,x)k(z,z)}}$.

Using the above we gave many examples of kernels over $\mathcal{X} \subset \mathbb{R}^n$:

Linear Kernel: $k(x, z) = x^\top \Sigma z, \Sigma \succeq 0$

Polynomial Kernel: $k(x, z) = (1 + x^\top \Sigma z)^d, d \in \mathbb{N}, \Sigma \succeq 0$

Gaussian Kernel: $k(x, z) = e^{-\frac{1}{2}(x-z)^\top \Sigma (x-z)}, \Sigma \succeq 0$

The key advantages of employing a kernel are as follows:

- Domain experts from various application fields of machine learning feel that specifying similarity between objects is easier than giving a vectorial description/representation for the objects. Since kernels measure similarity, it means that many feel it is easier to specify kernel rather than ϕ.
- Needless to say, because of the above advantage, MMLC can be used with arbitrary input spaces as long as a kernel is available.
- Since the prediction function $g(x)$ can be written interms of the kernel values, $g(x) = \sum_{i=1}^{m} \alpha_i y_i k(x_i, x)$, the kernel determines the characteristic of the prediction function. For example, prediction function is a linear function of x, if one employs a linear kernel and so on.

- Infact, any algorithm/methodology that employs dot-products of points rather than the point themselves may have the same benefits by employing kernels. For e.g., kernelized nearest neighbour, where the distance is specified by a kernel: $d(x,z) = \sqrt{k(x,x) + k(z,z) - 2k(x,z)}$.

We then noted a speciality of the Gaussian kernel, which is that all gram matrices induced by it (over distinct points) are (strictly) positive definite! We call such kernels are strictly positive kernels. The immediate consequence is that the dimensionality of the space spanned by the mappings of m distinct points in the RKHS will be m. We commented that this condition is a necessary condition for a kernel to induce all possible prediction functions (asymptotically). Interested students may read up this insightful manuscript Sriperumbudur et al. [2011].

2.5 Probabilistic Models

Probabilistic modelling[25] is necessary if the prediction function is desired to be a pmf/pdf. For e.g., predict the chance of rainfall today etc. Probabilistic modelling can also be used for standard ML problems like classification etc. For e.g., estimate $f_{Y/X}$ (conditional likelihood at each $X = x$) using probabilistic modelling, then define prediction function as $g(x) \equiv \text{sign}(f_{Y/X}(1/x) - f_{Y/X}(-1/x)))$. In the following, we define likelihood function as the pmf for discrete rvs and likelihood function as the pdf for continuous rvs.

Broadly there are two methodologies for building probabilistic models: non-parametric (refer section 2.5.1) and parametric (refer section 2.5.2.1). Further, for supervised learning problems, there is a complementary categorization for the methods:

Discriminative Modelling: This is a direct method where the idea is to model the (conditional) likelihood functions $f_{Y/X}$ at every $x \in \mathcal{X}$.

Generative Modelling: This is an indirect method where the idea is to model the entire joint distribution F_{XY} and then derive the prediction function from it.

Yet another complementary cateogrization of methods is[26]:

Bayesian Modelling: Unlike the classical objective of picking the "best" model parameters, here the idea is to view each parameter as a plausible candidate,

[25] Model is set of pmfs or set of pdfs rather than set of linear functions etc.

[26] All terms in this list are defined in section 2.5.2.1.1

seek opinion from each and compute the final output as a weighted average of individual opinions. In summary, the key characteristic in Bayesian modelling is that, associated with each parameter value, there is a likelihood and the estimate of the original likelihood is simply a marginalization over these. Eg. Bayesian Average Model.

non-Bayesian Modelling: Unlike Bayesian modelling, here we believe there is only one plausible candidate for model parameter value and seek for it (the "best" model parameter value). E.g., Maximum Likelihood Estimate.

Hybrids: A combination of the above. More specifically, estimate the (posterior) distribution for the model parameters (i.e., like in Bayesian methods assume that every model parameter value is plausible) and then pick the "best" parameter value (like in non-Bayesian methods) based on this distribution. Eg. Maximum Aposteriori (MAP) estimation.

2.5.1 Non-parametric Methods

2.5.1.1 Supervised Learning

It is easy to modify the k-NN model to output the probability of $x \in \mathcal{X}$ belonging to class $y \in \mathcal{Y}$: simply estimate this probability as the fraction of neighbours in training set belonging to this class y.

It is easy to modify the decision tree model to output the probability of $x \in \mathcal{X}$ belonging to class $y \in \mathcal{Y}$: simply estimate this probability as the fraction of examples in training set belonging to this class y in the leaf node that x landed-up in.

The above two methods, as described, are discriminative methods.

2.5.1.2 Unsupervised Learning

If one needs to estimate the likelihood function (either density estimation or probability estimation) of the input data itself[27], then one non-parametric way is to perform what is called as parzen-window estimation. Please refer section 14.7.2 in Murphy [2012] for details.

Also, in section 14.7.3, it is shown how to use this parzen-window estimator in generative modelling for supervised learning. It is interesting to note that this generative method recovers the k-NN algorithm described above for probabilistic

[27]Recall that this is problem is an example of an unsupervised learning problem.

outputs, which is an example of a discriminative method (and is described in section 2.5.1.1). Hence this is an example where both discriminative and generative modeling give exactly the same prediction function.

2.5.2 Parametric Methods

2.5.2.1 Modelling with Exponential Family

Typically we deal with models belonging to the exponential family[28] like, Bernoulli model[29], Gaussian model[30].

The main reference for this section is chapter 9 in Murphy [2012][31]. Definition of exponential family is given in section 9.2.1. Examples are illustrated in sections 9.2.2.1-9.2.2.3.

2.5.2.1.1 Unsupervised Learning Here we consider the problem of estimating the likelihood function given samples from it. We start with a model belonging to the exponential family[32]. Then, the only unkown is(are) the canonical parameter(s), which we denote as $\theta \in \mathbb{R}^n$. Hence the learning problem can simply be posed as an optimization problem with θ as the variable and the objective implying closeness of the distribution picked from this model to the true (but unknown) distribution. Also, it will be convinient if the measure of closeness will involve expected values rather than anything else (so that it will be easy to apply something like law of large numbers). Hence we came up with the method of moments:

$$\min_{\theta \in \mathbb{R}^n} \quad \sum_{i=1}^{\infty} \left(\mathbb{E}_\theta \left[\psi_i(X) \right] - \frac{1}{m} \sum_{j=1}^{m} \psi_i(x_j) \right)^2, \tag{2.10}$$

where $\psi_i(X) = X^i$ and $\mathbb{E}_\theta$ represents the expectation computed with f_θ as the likelihood.

This is expected to do well because as $m \to \infty$, $\frac{1}{m} \sum_{j=1}^{m} \psi_i(x_j)$ will converge to the true (but unknown) $\mathbb{E}\left[\psi_i(X)\right]$. Please refer Hansen [1982] for more details about consistency of method of moments.

[28] As will be clear later, this is the right analogue for linear models in deterministic modelling.

[29] Bernoulli model is the set of all Bernoulli distributions. Each distribution in this model can be identified with the "probability of heads".

[30] Gaussian model is the set of all Gaussian distributions. Each distribution in this model can be identified with the "mean" and "covariance/variance".

[31] Only relevant content upto section 9.3.

[32] i.e., given a set of distributions with a fixed h, Z, ϕ

While consistency-wise this algorithm seems good, in terms of computation it seems discouraging because there are (possibly) infinite terms in the objective. However atleast for Bernoulli and Gaussian models it is clear that only a finite number of them will determine the rest. Fortunately, this is a result that is applicable to any model belonging to the Exponential family:

Theorem 2.5.1. *Given a model belonging to exponential model*[33]*, the distributions in it can be either parameterized by θ, the canonical parameters, or by using $\mathbb{E}[\phi(X)]$, the expected sufficient statistics.*

Please refer Section 9.2.6 in Mohri et al. [2012] for a proof in the discrete distributions case. And for the continuous distributions case please refer Section 6.3.1 in`https://web.stanford.edu/class/stats311/Lectures/lec-07.pdf`.

In other words, for the exponential family, the method of moments reduces to the following:

$$\min_{\theta \in \mathbb{R}^n} \quad \sum_{i=1}^{n} \left(\mathbb{E}_\theta \left[\phi_i(X) \right] - \frac{1}{m} \sum_{j=1}^{m} \phi_i(x_j) \right)^2, \tag{2.11}$$

where $\phi_i(X)$ is the i^{th} sufficient statistic. More importantly, the proof for the theorem shows that there will exist a θ such that the objective in (2.11) will be zero. i.e, θ^* will be optimal iff $\mathbb{E}_\theta \left[\phi_i(X) \right] = \frac{1}{m} \sum_{j=1}^{m} \phi_i(x_j) \ \forall \ j = 1, \ldots, n$.

For Bernoulli model, this simply means that the estimate of true probability of heads is the fraction of heads in m tosses and for the Guassian model, the estimate of true mean and true covariance are the sample mean and sample covariance.

In statistics, there is an alternate intuitive method for estimating parameters of a model, which is called as Maximum Likelihood Estimation (MLE):

$$\max_{\theta \in \mathbb{R}^n} \quad \sum_{i=1}^{m} \log \left(f_\theta(x_i) \right), \tag{2.12}$$

where the objective is log of the likelihood of the training data computed using the distribution in the model corresponding to θ. Though this algorithm is very intuitive and needs no detailed motivation, it is not apparent why the solution provided by MLE will (atleast asymptotically) be close to the (Bayes) optimal. Please refer Section 1.2 in `http://www.win.tue.nl/~rmcastro/AppStat2011/files/MLE_partI.pdf`, which shows that the only difference between method of moments and MLE is that the former uses distance between Expectations (moment-generating-function

[33] Without loss of generality we assume $h(x) = 1$.

in general) as the notion of distance between distributions, whereas the later uses KL-divergence as the notion of "distance"[34].

Interestingly, MLE gives the exact same solution as method of moments for exponential models. Please refer section 9.2.4 in Murphy [2012] for details. MLE for Gaussian distribution is detailed[35] in section 4.1.3.

Bayesian Methodology As usual, we assume a model belonging to the exponential family is given and let the model parameters be represented by θ. In Bayesian techniques, we assume that every θ is plausible[36]. In particular, let the likelihood function generating the model parameters be f_Θ. Then in Bayesian methods, the idea is to estimate the final likeliood function as an average over all the parameters: i.e., $f_X(x) = \sum_i f_{X/\Theta}(x/\theta_i) f_\Theta(\theta_i)$ if Θ is a discrete rv and $f_X(x) = \int f_{X/\Theta}(x/\theta) f_\Theta(\theta)\, \mathrm{d}\theta$ if Θ is a conts. rv. Needless to say, here $f_{X/\Theta}(x/\theta)$ is nothing but the distribution indexed by θ in the given model[37].

Now while this clear, the key/only unknown in the above description is the likelihood function for model parameters f_Θ. The key step in Bayesian learning is hence estimating f_Θ from samples[38] of f_X ie., $\mathcal{D} = \{x_1, \ldots, x_m\}$. Needless to say, MLE/MM cannot be directly applied to estimate f_Θ (as samples from this distribution are not given). Hence we need to write down some equation that captures the relation between this and the f_X, whose samples are given. To this end, let us first realize that the correct notation for the estimate of $f_\Theta(\theta)$ after the samples $\mathcal{D}$ are shown is given by $f_{\Theta/D}(\theta/\mathcal{D})$, where D is the random variable representing the training set. Note that in the non-Bayesian case, the MLE estimate of θ was a fixed function of D; however, in the Bayesian set-up, even given $D = \mathcal{D}$, every θ is plausible and the relation between Θ, D is stored in $f_{\Theta/D}(\theta/\mathcal{D})$. This likelihood is usually refered to as the "Aposteriori likelihood of model parameters"[39]. In summary, the

[34]Section 2 in the same article (`http://www.win.tue.nl/~rmcastro/AppStat2011/files/MLE_partI.pdf`) shows that KL-divergence (though it is itself not a distance) upper bounds a valid notion of distance between distributions, known as Bhattacharyya/Hellinger distance. Also, refer section 1 of this article itself for a revision of properties of KL divergence.

[35]Note that the method of moments is definetely "easier" in this case as it directly says true mean, covariance estimates are sample mean, covariance. Whereas section 4.1.3 detials a complicated procedure for realizing the same. In summary, I encourage students to always keep in mind both these methods so that one can use whichever is "easier".

[36]Think about each distribution in the model, i.e., each model parameter, being an expert.

[37]Since θ was not a random variable in non-Bayesian methods, the notation we used earlier for $f_{X/\Theta}(x/\theta)$ was $f_\theta(x)$.

[38]Note that till now we discussed the problem of estimating a likelihood function for which some samples (the training set) are given. Here the problem is slightly different. We need to estimate likelihood of a rv that is related to another likelihood and the samples given are for the other related likelihood and NOT the original likelihood.

[39]because it represents the distribution of parameters after seeing the training set.

key/only unknown is $f_{\Theta/D}(\theta/\mathcal{D})$ and needs to be estimated from the samples $\mathcal{D}$.

The obvious way to procede now is to call the Bayes rule: $f_{\Theta/D}(\theta/\mathcal{D}) = \frac{f_{D/\Theta}(\mathcal{D}/\theta)f_{\Theta}(\theta)}{f_D(\mathcal{D})}$. Note that in this expression, $f_{D/\Theta}(\mathcal{D}/\theta)$ can be computed easily in terms of θ because we assume iid samples: $f_{D/\Theta}(\mathcal{D}/\theta) = \Pi_{i=1}^{m} f_{X/\Theta}(x_i/\theta)$. And, the denominator is simply the integration/summation of the numerator over all values of θ. Hence the aposterior distribution of θ and hence the estimate of the orginal likelihood as given by the averaging formula: $f_{X/D}(x/\mathcal{D}) = \int f_{X/\Theta,D}(x/\theta,\mathcal{D}) f_{\Theta/D}(\theta/\mathcal{D})\, \mathrm{d}\theta = \int f_{X/\Theta}(x/\theta) f_{\Theta/D}(\theta/\mathcal{D})\, \mathrm{d}\theta$, can be computed if an "apriori" likelihood f_Θ is given. The above averaged likelihood is usually refered to as the "Bayesian Averaged Model (BAM)", as we employed Bayes rule to build the aposterior and then used it in the averaging.

Though at first look, it may appear that again some "apriori" likelihood f_Θ needs to be supplied, and hence the problem remains unsolved; on a closer look we can infact say that the ability to utilize the "apriori" likelihood is one of the strengths of employing Bayesian methods. This is because, f_Θ simply represents the common knowledge one might have about the true/unknown parameter (even before showing a training set). So it is OK if we assume that the "apriori" likelihood must be specified by the model and which apriori distribution to employ is then a model selection issue.

It is usually customary to refer to such Bayesian models by using a name that contains two parts, the first one specifying the name of the aprior distribution for the model parameters and the second part specifying the name of the original likelihood function being modeled. For e.g., if the values x takes are binary, then a Bayesian model named "Beta-Bernoulli model" would specify that the f_X is being modeled by Bernoulli distributions and the apriori likelihood f_Θ for its parameter θ (i.e., the prob. of "heads") is given by a specific Beta distribution. The "parameters" for the specific apriori distribution are usually called as the hyper-parameters (since they determine the model; hyper-parameter selection then is the same as model selection). Please refer chapter 2 in Murphy [2012] for definitions/details of common distributions like beta.

Hybrids A popular hybrid method is to obtain the posterior of model parameters like in Bayesian methods and then use this likelihood to pick the "best" parameter value. And then ofcourse, use this best value alone to index for the best estimate of the original likelihood from the model. One of the criteria for picking the best is: by looking at where the aposterior likelihood is maximized (mode of aposterior). This method is called as Maximum Aposterior (MAP) estimation. Alternatively, one might go for mean/median of the aposterior likelihood etc.

Refer section 3.3, section 3.4, section 4.6.3 in Murphy [2012] [40] for details of the Beta-Bernoulli, Dirichlet-Multinoulli, GIW-Gaussian models (GIW stands for Gaussian-inverse-Wishart distribution; and is the aprior distribution in this case; and further, Gaussian models the original likelihood).

In all the above examples, you would have noticed that the form of the distribution of the aprior and the aposterior are exactly the same. For e.g., if original likelihood is given by Bernoulli and the aprior is a Beta distribution, then the posterior will be another Beta distribution. Such a special aprior distribution for a given likelihood function is called as "the conjugate prior". For e.g., GIW is the conjugate prior for Gaussian, Dirichlet is the conjugate prior for Multinoulli etc.

Please refer section 9.2.5.2 in Murphy [2012] for the conjugate prior for a given exponential distribution. Also, please read entire section 9.2.5 for details of conjugateExponential-Exponential model.

2.5.2.1.2 Supervised Learning

Non-Bayesian Generative modeling: We begin with generative modelling for binary classification with input space $\mathcal{X} \subset \mathbb{R}^n$. Firstly, note that the unknown distribution, F_{XY}, in this case cannot be described by a likelihood function (as XY jointly is neither conts. nor discrete). So an alternative is to model the likelihoods $f_{X/Y}$ for all $y \in \{-1, 1\}$ and also model f_Y. For e.g., one may employ (multivariate) Gaussian models for $f_{X/Y}$ and employ Bernoulli model for f_Y. The algorithms for estimating these likelihoods are described in section 2.5.2.1.1. Once these three likelihoods are modelled, one can write down the formula for F_{XY} and more importantly for $f_{Y/X}(y/x) = \frac{f_{X/Y}(x/y)f_Y(y)}{\sum_{y' \in \{-1,1\}} f_{X/Y}(x/y')f_Y(y')}$, using Bayes rule.

Further, if one obtains a classifier using $f_{Y/X}$ (by labeling an x by the class of highest probability), such a classifer is called as a "Bayes classifier". If a Bayes classifier is obtained by modeling class-conditionals with Gaussians, then this is usually refered to as "Gaussian Discriminant Analysis (GDA)". Please refer section 4.2 in Murphy [2012] for details of GDA. Also, refer formula (4.38) for an expression of $f_{Y/X}$ using GDA.

Complementary to this, if one assumes that the input features are independent given the class, i.e. the class conditionals $f_{X/Y}$ factorize as $f_{X/Y}(x/y) = f_{X_1/Y}(x_1/y) \dots f_{X_n/Y}(x_n/y)$ then, the corresponding "Bayes classifier" is called as the Naive-Bayes classifier. Please refer sections 3.5.1.1 for examples of a Naive-Bayes classifier. Note that typically the naive assumption of class-conditional feature in-

[40] you may skip sections 4.6.3.8, 4.6.3.9

dependence is done for the sake of computational convinience.

Bayesian Generative modeling: Given the above, one can easily think about Bayesian version of Bayes classifier: instead of MLE/MM for estimating the class-conditionals and class-prior, now employ BAM for all. All other steps remain the same. Refer sections 3.5.1.2&3.5.2 for Bayesian version of Naive-Bayes classifier.

Generative modelling for regression with input space $\mathcal{X} \subset \mathbb{R}^n$ is straightforward. For e.g., directly estimate the density f_{XY} by using a $n+1$ dimensional multivariate Gaussian. And then from this f_{XY}, one can always compute $f_{Y/X}$ using $f_{Y/X}(y/x) = \frac{f_{XY}(x,y)}{\int f_{XY}(x,y)\ \mathrm{d}y}$. For this case of Gaussian, the details of the formula are given in theorem 4.3.1 in Murphy [2012].

Non-Bayesian Discriminative Modeling: Recall that in discriminative modeling, the goal is to model $f_{Y/X}$ at every $X = x$. The novelty in this situation however is that we are not given (many) samples from this distribution at even perhaps a single $X = x$. Thus we need to seek for an alternate re-parametrization of these models for $f_{Y/X}$. We then said, lets look at how $f_{Y/X}$ looks like for a Bayes classifier, then that would suggest a suitable re-parametrization.

Infact, as refered earlier, formula (4.38) in Murphy [2012] gives this for GDA. This formula directly motivates the Logistic Regression model detailed in sections 8.1-8.3.1, 8.3.7. The corresponding MLE problem turns out to be convex and hence can be solved efficiently. Infact, the MLE problem is exactly same as a linear model with logistic loss (variant without maximizing margin).

Now one can think about a discriminative model for regression too based on the result in theorem 4.3.1 in Murphy [2012]. This leads to the linear regression model detailed in sections 7.1-7.3.2. Infact this is same as the linear model with square-loss (variant without maximizing margin).

Bayesian Discriminative Modeling: Given the non-Bayesian discussion, atleast methodology-wise, this should be straight-forward. Instead of MLE estimate of logistic regression or linear regression parameters, perform a BAM/MAP estimate. For details of Bayesian linear regression[41] refer sections 7.6-7.6.3.1.

Hybrid variants for Discriminative Modeling: The details of MAP estimate for Linear regression is presented in sections 7.5 in Murphy [2012]. Infact this is same as the linear model with square-loss (variant with maximizing margin).

[41] Interested students may refer section 8.4 for Bayesian Logistic regression.

The same with logistic regression is given in equation (8.45). Infact this is same as the linear model with logistic-loss (variant with maximizing margin).

2.5.2.2 Generative vs. Discriminative

Given the above discussions, it is clear that the discriminative models are direct methods and more importantly, involve lesser number of parameters to obtain the very same model as the corresponding generative models. Ofcourse generative models seem to play a key role in choosing the right re-parametrizations suitable for the discriminative ones. However, at the first look, it may seem that beyond this role in re-parametrization, generative models have no practical use. However this is incorrect. For example, suppose you have a situation where some input feature values are missing at prediction stage. Then since generative models model the entire joint, one can still come up with the estimate of likelihood of output given the non-missing (observed) values. Infact, one can even obtain likehoods of the missing ones given the observed ones. So in this sense generative models are more "powerful" than discriminative ones.

While the above is true, one should keep in mind that "powerful" need not mean more accurate. The intuitive reason being the fact that finite information being stored in the training dataset. In plain words, generative models can handle many questions but each answer has less confidence; while discriminative models can answer only one question (what is output given input), and hence may answer with more confidence. In summary, if the goal is purely predicting output given all inputs, then ofcourse discriminative is the way; else generative is perhaps the way. Please refer section 8.6 in Murphy [2012] for a detailed discussion on Generative vs. Discriminative.

More importantly, sometimes a complete feature may be missing during all stages of training and prediction, then in such situations one can still employ generative modelling. Such completely missing features are usually called as latent/hidden variables. The case of a single hidden/latent discrete variable is discussed in section 2.5.2.3.

2.5.2.3 Mixture Models

Refer sections 11.1 to 11.2.4.1 in Murphy [2012] for motivations/definition/examples of mixture model. Refer 11.3-11.4.2.5 for details of the Expectation Maximization (EM) algorithm[42] that reaches a stationary point (that has better likelihood than the intial iterate) of the likelihood function (this is itself impressive as the likelihood

[42]Interested students may refer section 11.4.2.8 for a MAP variant of EM.

function is non-convex). Refer section 11.4.7.1-11.4.7.2 for details of convergence of EM.

Please refer section 11.6.1 for details of EM algorithm for missing values under Gaussian model.

2.5.2.4 Models for Sequence Labeling

We refer readers to section 2.3.4.1 for notation and set-up for the sequence labeling problem.

As usual, we begin with generative models first. The obvious step is to work out details of a Bayes classifier for this problem. The key question hence is to ask which distribution can model sequences (of arbitrary length)? Surely, all multivariate distributions we know will not work as they can generate fixed length vectors. Here, the requirement is to generate sequences of vectors of arbitrary length. Since each element in the sequence is still a vector, which can be generated using multivariate distributions, we are motivated to present distributions that "employ" multivariate distributions.

The most "simplistic" distribution then is as follows: $f_X(x) = f_{X^{(1)},\ldots,X^{(T)}}\left(x^{(1)},\ldots,x^{(T)}\right) = f_{X^{(1)}}\left(x^{(1)}\right)\ldots f_{X^{(T)}}\left(x^{(T)}\right)$. Each $f_{X^{(i)}}$ can simply be modeled by any multivariate distribution[43]. Note that T is arbitrary. This kind of a model essentially says that each position in the sequence is independent of the other. Infact, there exists an even "simpler" model: $f_X(x) = f_{X^{(1)},\ldots,X^{(T)}}\left(x^{(1)},\ldots,x^{(T)}\right) = g\left(x^{(1)}\right)\ldots g\left(x^{(T)}\right)$. Here g is some particular distribution. This model, in addition to independence, imposes a "homogenity" assumption that distributions centered at every location is the same (homogeneous).

Ofcourse, we rarely use the above models because in practice the independence (and homogenity) assumptions are too strong. The next higher level of complex models are known as Markov chains: $f_X(x) = f_{X^{(1)},\ldots,X^{(T)}}\left(x^{(1)},\ldots,x^{(T)}\right) = f_{X^{(1)}}\left(x^{(1)}\right) f_{X^{(2)}/X^{(1)}}\left(x^{(2)}/x^{(1)}\right) f_{X^{(3)}/X^{(2)}}\left(x^{(3)}/x^{(2)}\right)\ldots f_{X^{(T)}/X^{(T-1)}}\left(x^{(T)}/x^{(T-1)}\right)$. Surely, this model is NOT saying positions are independent, but it is indeed saying that they are conditionally independent. In particular, the conditional independence assumption encoded[44] in the above is: $f_{X^{(t)}/X^{(t-1)},X^{(t-2)},\ldots,X^{(1)}}\left(x^{(t)}/x^{(t-1)},x^{(t-2)},\ldots,x^{(1)}\right) = f_{X^{(t)}/X^{(t-1)}}\left(x^{(t)}/x^{(t-1)}\right)$. i.e., given the parental position, the ancestors are irrelevant.

[43] univariate if we are talking about output sequence.

[44] Ofcourse, there are other conditional independencies that are in-turn implied by this.

This is the case because $f_X(x) = f_{X^{(1)},\ldots,X^{(T)}}\left(x^{(1)},\ldots,x^{(T)}\right) =$

$$f_{X^{(1)}}\left(x^{(1)}\right) f_{X^{(2)}/X^{(1)}}\left(x^{(2)}/x^{(1)}\right)\ldots f_{X^{(t)}/X^{(t-1)},X^{(t-2)},\ldots,X^{(1)}}\left(x^{(t)}/x^{(t-1)},x^{(t-2)},\ldots,x^{(1)}\right)$$
$$\ldots f_{X^{(T)}/X^{(T-1)},X^{(T-2)},\ldots,X^{(1)}}\left(x^{(T)}/x^{(T-1)},x^{(T-2)},\ldots,x^{(1)}\right).$$

The particular conditional independence assumption made above is known as Markov assumption and hence the name Markov chain or Markov Model.

Now, we can also talk about a homogeneous Markov chain: $f_X(x) = f_{X^{(1)},\ldots,X^{(T)}}\left(x^{(1)},\ldots,x^{(T)}\right) = f_{X^{(1)}}\left(x^{(1)}\right) g\left(x^{(2)}/x^{(1)}\right) g\left(x^{(3)}/x^{(2)}\right)\ldots g\left(x^{(T)}/x^{(T-1)}\right)$. Again, the conditional distribution centered at every position is the same and hence the additional qualifier of homogeneous.

Estimating the parameters of a homogeneous Markov chain is already known to us, provided we model every g, as well as $f_{X^{(1)}}$ by distributions in exponential family. Note that we say 'every g' because g is a distribution over values at a position given the value in the previous (parent) position. In particular, if the Markov chain is over output sequences, then g given an alphabet can be modeled by multinoulli and $f_{X^{(1)}}$ can also be modelled using Multinoulli. Details of terminology used in case of Markov chains (like states, state transition matrix etc.) are given in sections 17.1-17.2.1 in Murphy [2012]. Details of MLE estimation of parameters are given in section 17.2.2.1. Ofcourse, instead of MLE, one may employ MAP or BAM for parameter estimation.

Note that in the case Markov chain is over the output sequences of alphabets (like in the eg. of speech and character recognition), then the Markov chain essentially models the language. Needless to say, such a homogeneous markov chain will be a terrible generative model for human languages like English. But they will prove useful when employed in a Bayes classifier for sequence labeling problem (as will be clear later). Other applications of language models are listed in section 17.2.2.

The above discussion provides models for f_Y (e.g. homogeneous Markov chain). Now the remiaining element in Bayes classifier is $f_{X/Y}$. We model this by a simple independence assumption: $f_{X/Y}(x/y) = f_{X^{(1)},\ldots,X^{(T)}/Y^{(1)},\ldots,Y^{(T)}}\left(x^{(1)},\ldots,x^{(T)}/y^{(1)},\ldots,y^{(T)}\right) = f_{X^{(1)}/Y^{(1)}}\left(x^{(1)}/y^{(1)}\right)\ldots f_{X^{(T)}/Y^{(T)}}\left(x^{(T)}/y^{(T)}\right)$ and further a homogenity assumption: $f_{X/Y}(x/y) = g\left(x^{(1)}/y^{(1)}\right)\ldots g\left(x^{(T)}/y^{(T)}\right)$. Every g at given $y^{(i)}$ can be modeled[45] by a suitable exponential distribution (say multivariate Gaussian[46]). Ofcourse we know how to perform a MLE/MAP/BAM estimate for these distributions. This completes our discription of the training the Bayes classifier. Section 17.5.1 in Murphy [2012] provided details of this training algorithm. Lets call this the Markov Bayes classifier.

[45] g at a given value of alphabet is called as the "emission distribution" from that alphabet.

[46] hence a Gaussian at every alphabet needs to be estimated.

Now lets focus on the issue of infering/predicting the "best" output sequence given an input sequence $x = \left(x^{(1)}, \ldots, x^{(T)}\right)$ and using this (trained) Markov Bayes classifier. One obvious way is to perform a MAP estimate: $\arg\max_{y \in \mathcal{Y}_T} f(y/x)$, where f is the posterior model with the trained parameter ($f(y/x) \propto f(x/y)f(y)$). Note that, unlike in any previously encountered situation like multi-class classification or regression, this optimization problem is non-trivial and a naive algorithm for finding this max might require $O(c^T)$ effort. Here, c is the total number of alphabets/states. Fortunately, because of the convinient conditional independences made in the Markov Bayes model, this max problems admits a poynomial time algorithm. Please refer to the Viterbi algorithm described in section 17.4.4 (17.4.4.1-17.4.4.5) in Murphy [2012] for an algorithm that requires only $O(c^2T)$ for solving this MAP problem.

The above discussion makes it clear that the decision about the output sequence is made at a "sequence level" rather than at individual position level and hence even the simplistic Markov chain language model[47], when employed in this Bayes classifier, seems to give a "rich" classifier.

Now lets think about a discriminative version of the Markov Bayes classifier. For this, we will need to look at the expression for $f_{Y/X}(y/x)$:
(2.13)

$$f_{Y/X}(y/X) = \frac{g\left(x^{(1)}/y^{(1)}\right) \ldots g\left(x^{(T)}/y^{(T)}\right) f_{Y^{(1)}}\left(y^{(1)}\right) h\left(y^{(2)}/y^{(1)}\right) h\left(y^{(3)}/y^{(2)}\right) \ldots h\left(y^{(T)}/y^{(T-1)}\right)}{\sum_{\forall \bar{y}} g\left(x^{(1)}/\bar{y}^{(1)}\right) \ldots g\left(x^{(T)}/\bar{y}^{(T)}\right) f_{Y^{(1)}}\left(\bar{y}^{(1)}\right) h\left(\bar{y}^{(2)}/\bar{y}^{(1)}\right) h\left(\bar{y}^{(3)}/\bar{y}^{(2)}\right) \ldots h\left(\bar{y}^{(T)}/\bar{y}^{(T-1)}\right)}.$$

Instead of writing the posterior in the above form, we can re-write using the parameters (e.g. one for each Gaussian (g) at $y^{(i)}, i = 1, \ldots, T$ and one for each multinoulli (h) at $y^{(i)}, i = 1, \ldots, T-1$ and one multinoulli modelling $Y^{(1)}$.)

In discriminative modelling, the goal is to maximize the this conditional (posterior) likelihood wrt. to these parameters. Though the parameters are exactly the same as in Markov Bayes classifier, this maxmization is not as easy as the Markov Bayes just because of the summation in the denominator that has c^T terms! But fortunately this can be done efficiently[48]. However the details are beyond the scope of this course. Motivated by this discriminative model, one can design a struct-SVM based method for the sequence labeling problem[49] (refer section 2.3.4.1).

Finally, all of the above methods perform supervised training and hence a labeled dataset is required. In the example of speech recognition (introduced in

[47] Which seems to be a terrible generative model for language

[48] Interested students can look at CRFs, which are a generalization of this "discriminative Markov model" presented in section 19.6 in Murphy [2012]. In particular, the training algorithm is described in section 19.6.3.

[49] Infact the relationship between CRF and struct-SVM is more close and is detailed in section 8.5 in Mohri et al. [2012] and section 19.7 in Murphy [2012].

section 2.3.4.1), this means that one needs to carefully specify from what time instant to what time instant, which phoneme/alphabet was uttered. This level of supervision may not be available. In such cases, one may think about a completely un-supervised version of the above, which is popularly known as the Hidden Markov Model (HMM). The details[50] are provided section 17.3-17.5 in Murphy [2012].

[50]I expect students to be very familiar with the terminology in HMM (section 17.3), and the outline of the EM algorithm i,e., entire section 17.5.2, and viterbi i.e., section 17.4.4. The other sections/sub-sections like section 17.4.2-17.4.3 are for interested students and may be skipped.

Chapter 3

Model Selection

Many a time, we are confronted with the question of choosing the "best" among the multiple models/algorithms for a particular learning problem. If the question is a broad one inquiring which model is the best in general, then perhaps the only way is to look at asymptotic or large-sample bounds (like the ones we studied for kNN) and rank the models. However, since most of these bounds apply for any unknown distribution, they may be pessimestic for the specific data at hand. So in order to choose the best model for a dataset, one usually opts for what is known as k-fold cross validation procedure (which, when $k = 2$, mimics how a human supervisor selects his best student).

3.1 Cross-Validation

In k-fold cross validation, the idea is to partition the given dataset into k parts (randomly). Here k may be any value from 2 to m, the size of the dataset. Then given this partition, one loops over each part. In the i^{th} iteration, the i^{th} part of the dataset is called as Validation set and the remaining $k - 1$ parts are considered as training set and each model is trained. Then using the trained models, prediction of labels (inference) is performed on the validation set. Using a performance measure (like accuracy, precision, recall etc.; see section 3.4), the performance of each model on the Validation set is measured. After all the iterations end, the score for each model is simply taken as average performance score over all iterations (with the different k validation sets). The highest of these is declared the winner. Please refer section 7.10 in Hastie et al. [2009] for more details.

We then asked the following important question: while cross-validation indeed mimics how humans tend to select "best" humans, is there any theoretical justifica-

tion for this algorithm? The answer is yes, but the complete answer is beyond the scope of this course. What we gave was simply an intuitive insight into the answer. We noted that the (cross-)validation error is simply an empirical estimate or sample mean version of the true expected error. Hence, as the number of validation examples increases, we expect that the sample mean goes to the true mean because of law of large numbers. Extending this result, we expect that the minimizer of validation error also approches that of the true error[1]. Hence cross-validation "works" if there are enough number of validation examples.

An interesting observation is that the number of points used for training and the overall number of points used for validation will be maximized when $k = m$. This extreme case is called as leave-one-out-validation and the corresponding error is called Leave-One-Out (LOO) error. An interesting result, due to Luntz and Brailovsky [1969], is that the expected LOO err performed on m examples is equal to the expected error of a classifier trained with $m-1$ examples. This is a neat result because in case of LOO, the model is trained with different $m-1$ sized datasets.

While this seems fine, a related meta question is "how many models to try?". Note that the above discussion simply gives a procedure for selecting the best among a given number of models. It does NOT say whether to try 10 models (say 1-NN to 10-NN) or to try 100 models (say 1-NN to 100-NN). While it seems obvious that the latter is better, because it would definitely give a lower cross-validaton error; it is important to note that in the latter case the claim finally is about "the best in 100 models is so and so" vs. a relatively weaker claim in the former case "the best in 10 models is so and so". Naturally we expect that the in the latter case, the confidence with which we can make the stonger claim will be lower (as compared to that in the former case; because there is only limited amount of information stored in the training set regarding the unknown distribution).

Interestingly, the guarantee (upper bound) one can provide on the true (but unknown) probability of misclassification turns out to be a sum of these two terms i.e., sum of the validation error (empirical error or error on the sample) and the confidence term. More specifically, if the set of models to be tried is $\mathcal{M}$, then one can make the following statement: with probability atleast $1-\delta$,

$$P\left[Y \neq g^{best}(X)\right] \leq \frac{1}{m}\sum_{i=1}^{m} 1_{\left\{y_i \neq g^{best}(x_i)\right\}} + O\left(\mathcal{N}_{\mathcal{M}}\sqrt{\frac{\log(\frac{1}{\delta})}{m}}\right),$$

[1]It so happens that this convergence may not happen if there are "too many" models to select from. Please see the $\mathcal{N}$ complexity defined later. The key result is that as long as $\mathcal{N} < \infty$ this convergence is gauranteed to happen. In this course we will ofcourse restrict ourselves to such cases only.

where $\mathcal{N}_{\mathcal{M}}$ is a measure[2] for "complexity" of the set of models that satisfies the following relation: $\mathcal{M}_1 \subset \mathcal{M}_2 \Rightarrow \mathcal{N}_{\mathcal{M}_1} \leq \mathcal{N}_{\mathcal{M}_2}$; and $g^{best} \in \mathcal{M}$ is the best prediction function chosen by cross-validation procedure. Unfortunately, providing more insight into the nature of $\mathcal{N}$ is beyond the scope of this course.

So, in practice[3] one usually pre-decides (before the training data arrives), the set $\mathcal{M}$ i.e., pre-decides the models to be considered and then sticks to them.

3.1.1 Approximation vs. Estimation Trade-off

In light of the discussion above, it is clear that as the "size/complexity" of the model increases, estimation of the true parameters/distributions becomes difficult. On the contrary, bigger the model, more likely will it contain a given (but unknown) Bayes Optimal function. Because of this trade-off, there cannot exist a "universally best" model.

The difference between the risk with the estimated function/parameter and the minimum attainable in the model is called as estimation error. And, the difference between the Bayes optimal error and the minimum attainable in the model is called the approximation error. Thus, as model size increases, the approximation error decreases (in general); however, the estimation error increases (in general). Hence the goal of model selection should be to strike the correct balance between the two errors.

3.2 Bayesian take on Model Selection

The main reference for the basic idea is detailed in section 5.3 in Murphy [2012], which is briefed below.

CV procedure ranks models based on the performance of the "best" candidate in it. After understanding Bayesian methods, one would perhaps think about a way of ranking models based on the "average" performance of candidates in it. Such a method is called as the Maximum Marginal Likelihood (MML) method. The definition and details of computing marginal likelihood for a given model are presented in section 5.3.2 in Murphy [2012]. In MML, the idea is to compute the so-called marginal likelihood for each model. And then, rank the models by decreasing marginal likelihood. In particular, the "best" model will be that whose performance on average (across all candidates in it) is highest. Needless to say, once

[2] In the lectures, the second term in the above bound was what refered to as $h(\delta, m)$.

[3] the other alternative (which is a non-option for you) is to actually derive the bounds as above carefully.

a model a selected, the "best" in each model can be computed using MLE or MAP or BAM. There is a nice interpretation for the MML method detailed in section 5.3.1 in Murphy [2012].

Infact, a MAP or BAM version of the MML can be derived provided one is given some prior over the models themselves. The MAP is detailed in near equation (5.12) in Murphy [2012]. It is easy to write down the BAM version of MML now. This discussion now leads to another question: how do you choose the prior over the models? In other words, how would you choose the hyper-hyper-parameter determing the prior over the models? For anwsering this, we may repeat the above discussion now at hyper-hyper level rather than at hyper level. This idea of layered BAM/MAP models are called as Hierarchical Bayes models. And typically one is happy with two stages. The variants possible are listed in table on page 173 in Murphy [2012]. It is easy to see that the generic name for MML is Empirical Bayes described in the table. The discussion in sections 5.5 and 5.6 shows how Hierarchical Bayes can be employed for problems other than model selection like multi-task learning motivated in sections 5.4.4.2 and 5.5.1 of the book.

3.3 Feature Learning

A particular model selection problem is that of selecting the "best" features. Typically the setting is that some basic features describing the input are known and the goal is to pick few features that are generated by combining the given input features. We focus here on probabilistic generative models and leave the other paradigms for future courses.

The core idea in generative models paradigm for performing feature selection is to use the notion of latent variables and run some EM-like training algorithm. Then the value(s) of the latent variable(s) are used as the final features. For e.g. consider the mixture model, which is indeed an example of a model having a single discrete[4] latent variable, say y. A new feature representation for x can simply be the pmf of y/x under the estimated model. In lectures some text categorization based examples were given to show this representation is not bad and may be highly compact compared to the original features.

A generalization of mixture models where there are multiple latent variables, which my take continuous values, is factor model specified by: $f_{X/Z}(x/z) = \mathcal{N}(Wz + \mu, \Psi), f_Z(z) = \mathcal{N}(\mu_0, \Sigma_0)$. The parameters W, μ, Ψ need to be estimated and typically Ψ is restricted to be diagonal and $\mu_0 = 0, \Sigma_0 = I$. Once the parameters are estimated, the new features for x are given as the mode of posterior i.e.,

[4]infact takes on finite number of values

$\arg\max_z f_{Z/X}(z/x)$.

The main reference for this method is section 12.1 in Murphy [2012]. In particular, the details of the training/EM algorithm are detailed in section 12.1.5 in the book (you need to use $c = 1$ and $r_{ic} = 1$ in the derivations). Equation (12.20) in the book gives the formula for obtaining the new features given the old.

In section 12.2 the connection between the above and PCA is described[5]. This throws light on why the above methods is called as Probabilistic PCA (PPCA).

Supervised version of PPCA is easy: in addition to $f_{X/Z}(x/z) = \mathcal{N}(Wz + \mu, \Psi)$, $f_Z(z) = \mathcal{N}(\mu_0, \Sigma_0)$, we will have $f_{Y/Z}(y/z) = \mathcal{N}(w^\top z, \Sigma)$. Here, the last part is simply the linear regression part. The only difference between this and vanilla linear regression is that the relation between y and x is now reflected as relation between y and z. In other words, this is like encoding the input features into latent features and using the latent features to solve the supervised problem. Ofcourse, if we have a classification problem, then we will have $f_{Y/Z}(y/z) = Ber(\frac{e^{-yw^\top z}}{1+e^{-w^\top z}})$. This is like logistic regression on latent variables.

Note that in the supervised cases, one may finally be only interested in predicting y given x and hence one may not even bother to recover the latent representation z used. Nevertheless, we are indeed performing a feature learning/selection step implicitly.

Another point to note is that, either in factor model based linear regression or factor model based logistic regression, the final relation between y and x is linear. One may now think about multiple layers of latent variables (with perhaps different no. variables in each layer). In this case, the relationship between y and x may be non-linear. Such layered factor models are known as Neural networks, a discussion of which is beyond the scope of this course.

3.4 Performance Measures

A simple and "universal" performance measure is likelihood of data. However, there are alternatives as briefed below.

[5]I expect the students to be familiar with PCA, which was also described in midsem exam. An explanation of it is given in this section too.

3.4.1 Binary Classification

Most of the performance measures can be derived given the so-called confusion matrix. Refer page-1 in `http://www.damienfrancois.be/blog/files/modelperfcheatsheet.pdf` for details.

3.4.2 Ordinal Regression

The following is one reference `http://www.inescporto.pt/~jsc/publications/journals/2011JaimeIJPRAI.pdf`.

3.4.3 Regression

Refer page-2 in `http://www.damienfrancois.be/blog/files/modelperfcheatsheet.pdf` for details. The most popular measure however is not mentioned in this pdf. It is called as explained variance. It is given by: $1 - \frac{MSE}{var(y_i)}$, where $var(y_i)$ denotes the variance in the y values in the training set. If this quantity is ≥ 0 for a regression algorithm, then it means that the algorithm is better than a simple baseline of always predicting y as the average y across the training set.

Bibliography

A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. .Vapnik. Support Vector Clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Wei Chu and S. Sathiya Keerthi. New Approaches to Support Vector Ordinal Regression. In *Proceedings of International Conference on Machine Learning*, 2005.

T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

L. Devroye and L. Gyorfi. *Nonparametric Density Estimation: The L1 View*. John Wiley, New York, 1985.

L. Devroye, L. Gyorfi, and G. Lugosi. *A Probabilitic Theory of Pattern Recognition*. Springer, 1996.

V. Feldman, V. Guruswami, P. Raghavendra, and Yi Wu. Agnostic learning of monomials by halfspaces is hard. *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 385–394, 2009.

Lars Peter Hansen. Large sample properties of generalized method of moments estimators. *Econometrica*, 50:1029–1054, 1982.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A Dual Coordinate Descent Method for Large-scale Linear SVM. 2008.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/089976601300014493.

A. Luntz and V. Brailovsky. On estimation of characters obtained in statistical procedure of recognition. *Technicheskaya Kibernetica (Russian)*, 3, 1969.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012.

Kevin Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 1 edition, 2012.

A. Nemirovski. Lectures on Modern Convex Optimization. `http://www.isye.gatech.edu/faculty-staff/profile.php?entry=an63`, 2005.

S. M. Ross. *A First Course in Probability*. Pearson Education, 6 edition, 2002.

B. Scholkopf, J. C. Platt, J. C. Shawe-Taylor, A. j. Smola, and R. .C. Williamson. Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, 13(7):1443–1471, 2001.

Bharath K. Sriperumbudur, Kenji Fukumizu, and Gert R.G. Lanckriet. Universality, Characteristic Kernels and RKHS Embedding of Measures . *Journal of Machine Learning Research*, 12:2389–2410, 2011.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *JOURNAL OF MACHINE LEARNING RESEARCH*, 6:1453–1484, 2005.

V. N. Vapnik and Sayan Mukherjee. Support Vector Method for Multivariate Density Estimation. In *Advances in Neural Information Processing Systems*, 2000.