Mathematics Department of Aristotle University of Thessaloniki

Information Theory's Relation and Application to Machine Learning

A Theoretical Approach and Implementation

Dimitriadis Spyridon
Kofterou Maria
Supervisor Professor: Antoniou Ioannis

Contents

Abstract

The main purpose of this essay is to examine and introduce the use of Information Theory in Machine Learning. In the beginning of this essay the reader shall be introduced to the basic concepts of Machine Learning and Information Theory. Once the main definitions are established, the relation between Machine Learning and Information Theory is discussed and further explored. Proceeding to a deeper analysis, the main application and mathematical algorithms that form this connection are presented and proved. Heading from theory to practice, an implementation of Multinomial Logistic Regression using Cross Entropy algorithm in a Neural Network is used for the Classification of Mnist handwritten digit dataset. The results are presented and compared to the ones obtained from other MLP classification using MSE as objective function for the same dataset, discussed and analyzed.

Introduction

Information Theory was first introduced by Shannon's publication "A mathematical Theory of Communication" in 1948, as an effort to cover the needs of electrical engineers   to design efficient yet reliable communication. Few years later, Machine Learning is approached by Alan Turning's proposal of a 'learning machine' that could learn and become artificially intelligent. As a scientific endeavor, machine learning grew out of the quest for artificial intelligence. Already in the early days of Artificial Intelligence as an academic discipline, some researchers were interested in having machines learn from data.

Although there are a lot of similarities between Machine learning's mathematical foundation that lies on statistics and probabilities, and Information Theory that is mathematically formalized using the same tools, as well as information is a concept that rules Machine Learning procedures, it was not until the late 70's that their connection started to be investigated.

The last 35 years the Information Theory's relation and application to Machine Learning was widely examined and numerous theories and effective algorithms have developed, with lots of them proved to be more efficient than the traditional ones proposed. This essay aims to present the elements of both Machine Learning and Information Theory and to develop the relationship between them, present efficient entropy-based algorithms to accomplish different Machine Learning tasks and stimulates one of them in real data to present and discuss the results.

An introduction to Machine Learning

Machine learning (ML) is the study and construction of systems that can learn from data, that is computer algorithms that improve automatically through experience. These systems are called learning machines. However, the ultimate goal of ML study is insight, not machine itself. The term insight accounts for learning mechanisms in descriptions of mathematical principles. In a loose sense, learning mechanisms can be regarded as the natural entity.

Machine learning draws on concepts and results from many fields, including statistics, artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity, and control theory. Among the tools developed under the umbrella of Machine Learning are: Decision Trees, Neural Networks, Genetic Algorithms, Bayesian Learning.

Let's try to clarify the above definition by approaching ML with an example from real life, the Instructor-Student example, a parallelism that will be widely used to demonstrate the concepts of ML in this chapter: When a instructor teaches reading to his student, the student is not taught to read specific words and texts, but is taught the rules and concept of reading thus he can read every text that comes to his hands. This is the concept of ML, forming a mathematical algorithm -and transform it to a computer one- that will use a certain and finite dataset, called **Train set**, to learn how to accomplish a given task, and then generalize this knowledge to accomplish the same task while given new, unknown data.

Introducing a first and more empirical approach of "learning", we could describe the learning process as: Form groups from training samples-Extract main features-Generalize- Make Decision. How the main features will be extracted and the decision rule depend on the task that is to be accomplished. The following theorem, **The inductive learning hypothesis**, defines the learning process formally:

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.(Mitchell, 1997)

Technically speaking, a Learning Machine follows the following basic 4 steps:

1) **Choose the basic algorithm to perform the task** by analyzing the training set, in order to form the model.
2) **Learning**: Optimize model's parameters based on the data.
3) **Complete the task**

4) **Measure model's performance**: Using the **Test set**, a set of new data, the algorithm was not trained on, in order to compute how efficiently the task was accomplished and whether modifications have to be made. There are a lot of **evaluation measures**, according to the specific task to be accomplished.

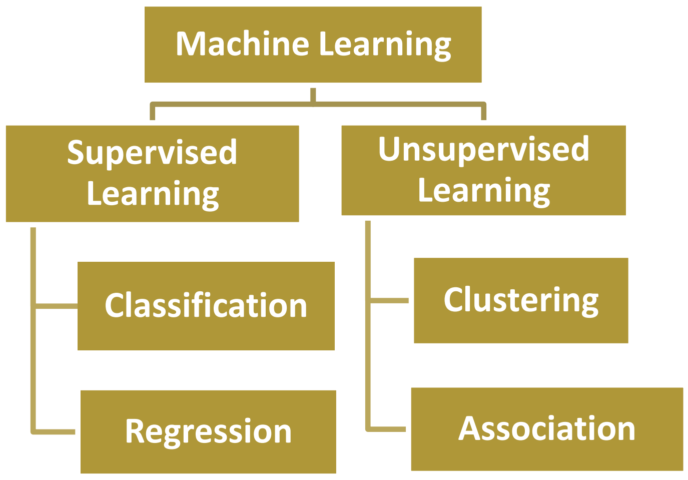
To reassure that the concepts of Train and Test set in ML are concrete, we will use the Instructor-Student example once more: While teaching multiplication the instructor uses examples and exercises to demonstrate the principles of multiplication to his students, these exercises are the train set. In order to ensure though, that the students understood and got familiar with multiplication and not just learned by heart the solved exercises, the teachers provides them with homework exercises to test whether or not the lesson should be repeated, and these are the test set.

Just as there are different ways in which we ourselves learn from our own surrounding environments, so it is with learning machines. In a broad sense, we may categorize the learning processes through which learning machines function as follows: learning with a teacher and learning without a teacher. Learning with a teacher is also referred to as **supervised learning**. In conceptual terms, we may think of the teacher as having knowledge of the environment, with that knowledge being represented by a set of input–output examples (S. Haykin, 2008). **In unsupervised, or self-organized, learning**, there is no external teacher or critic to oversee the learning process. Rather, provision is made for a task-independent measure of the quality of representation that the model is required to learn, and the free parameters of the machine are optimized with respect to that measure. For a specific task-independent measure, once the learning machine has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically (Becker, 1991).Reinforcement learning and semi-supervised learning are subcategories created by the main two above.

The Instructor- Student example will be used again, to attempt a clearer demonstration of the above learning categories: In the supervised learning procedure, during foreign languages class, the student is provided with a list of words and its translation, like book and store and then he is asked to form or translate new words, like bookstore. In unsupervised learning the student is given a set of test in different languages and, without prior knowledge of those languages, he is asked to find common features, like repeated words or same syntax, and group the texts of same language.

Moreover, we should define **overfitting or overtraining**. It stands for the phenomenon of a learning machine learning too many input–output examples, ending up memorizing the training data. It may do so by finding a feature (due to noise, for example) that is present in the training data, but not true of the underlying function that is to be modeled. When the machine is overtrained, it loses the ability to generalize between similar input–output patterns.(S. S. Haykin, 2009)

In the following graphic it is presented how the two basic learning categories divide machine learning and the basic tasks to be accomplished in each category.



We shall briefly define:(Theodoridis, Koutroumbas,2003)

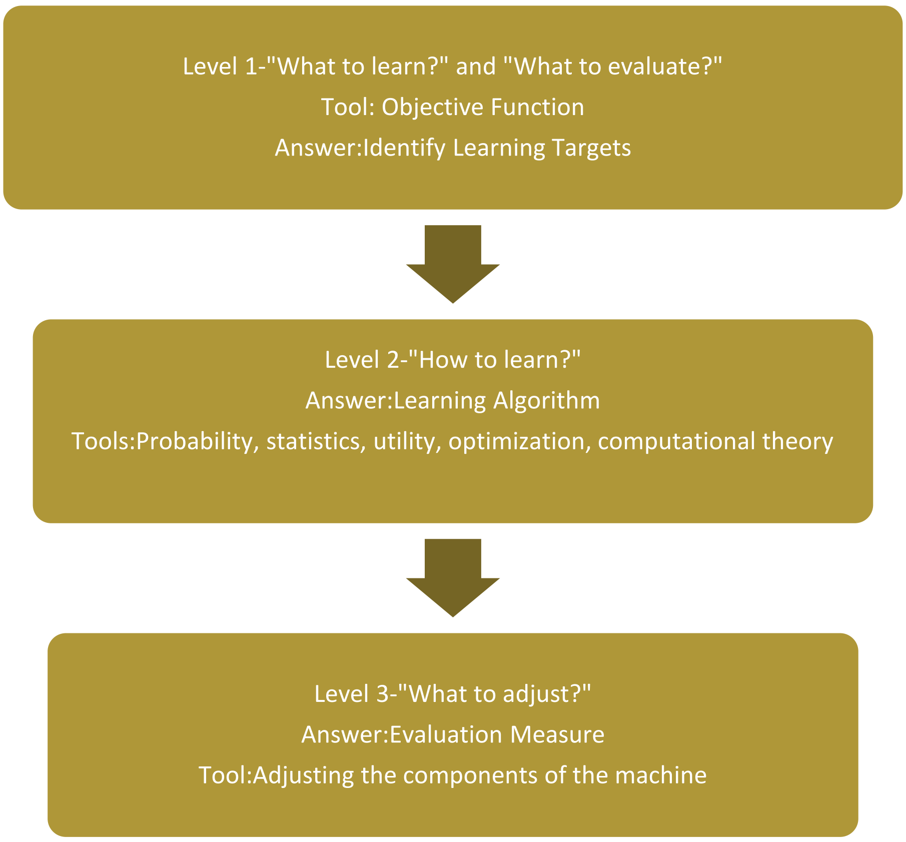
Classification: Such problems, in which the task is to classify examples into one of a discrete set of possible categories.
Regression: The task of approximating a real-valued target function.
Clustering: The task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups.
Association: The task is discovering interesting relations between variables in large databases.

As all the basic definitions and concepts of ML are now set, the logical following question to arise is how information is processed in a learning machine. In his paper "Information Theory and its relation to Machine Learning" (Bao-Gang Hu,2015),Hu proposed a three-level approach for information processing by a machine, formed by the following four questions:

1) **"What to learn"** is a study on identifying learning target(s) to the given problem(s), which will generally involve two distinct sets of representations, linguistic and computational.
2) **"How to learn?"** is a study on learning process design and implementations. Probability, statistics, utility, optimization, and computational theories will be the central subjects. The main concerns are generalization performance, robustness, model complexity, computational complexity/cost, etc. The study may include physically realized system(s).

3) **“What to evaluate?”** is a study on “evaluation measure selection” where evaluation measure is a mathematical function. This function can be the same or different with the objective function defined in the first level.
4) **“What to adjust?”** is a study on dynamic behaviors of a machine from adjusting its component(s). This level will enable a machine with a functionality of “evolution of intelligence”.

The following flow chart will demonstrate how these four questions form the three levels of ML as well as what these levels are constituted by:


Level 1-"What to learn?" and "What to evaluate?"
Tool: Objective Function
Answer:Identify Learning Targets
Level 2-"How to learn?"
Answer:Learning Algorithm
Tools:Probability, statistics, utility, optimization, computational theory
Level 3-"What to adjust?"
Answer:Evaluation Measure
Tool:Adjusting the components of the machine

The first level is also called “learning target selection”. The three levels above are neither mutually exclusive, nor collectively exhaustive to every problem in machine learning. We call them basic so that the extra problems can be merged within one of levels. The problems within four levels are all interrelated, particularly for “What to learn?” and “What to evaluate?” . “How to learn?” may influence to “What to learn?”, such as convexity of the objective function or scalability to learning algorithms from a computational cost consideration. Structurally, “What to adjust?” level is applied to provide the multiple closed loops for describing the interrelations.

At this point, the crucial remark is that whenever a learning target is wrong in the computational representation, one is unable to reach the goal from Levels 2 and 3.Up to now, we have missed much studies on learning target selection if comparing to a study of feature selection. When “What to learn?” is the most primary problem in machine learning, we do need a systematic, or comparative, study on this subject. Furthermore, the subject of “What to learn?” will provide a strong driving force to machine learning study in seeking “the fundamental laws that govern all learning processes” . Starting from these one starts to realize that, when contemplating learning targets selection and feature extraction in the learning process, the concept of information carried from each target or feature, and its correlation to the learning algorithm and the task to be accomplished, pops up. And this is the point that Information Theory makes its appearance.

Information Theory in Machine Learning

In his paper that was meant to be a classic and set the foundation for a whole new area of studying, Claude Shannon in 1948 introduced and formalized the elements of Information Theory (IT). The origins of IT were purely practical: the need of communication systems that are both efficient and reliable to be designed was what emerged Shannon's original. However, information theory if currently formed as a deep mathematical theory contemplating no more than the very essence of the communication process. IT carves out a framework for the study of fundamental issues such as the efficiency of information representation and the limitations involved in the reliable transmission of information over a communication channel.

Moreover, the theory encompasses a multitude of powerful theorems for computing ideal bounds on the optimum representation and transmission of information-bearing signals. These bounds are important because they provide benchmarks for the improved design of information-processing systems.(S. Haykin, 2008)

In the following part of this essay we will present information-theoretic models that lead to self-organization in a principled manner. In this context, a model that deserves special mention is the **maximum mutual information principle** due to Linsker (Linsker,1988). This principle states the following:

The synaptic connections of a multilayered neural network develop in such a way as to maximize the amount of information that is preserved when signals are transformed at each processing stage of the network, subject to certain constraints.

Entropy Functions

Given a discrete random variable $Y = \{y_k \mid k = 0, \pm 1, \pm 2, ..., \pm K\}$, with probability mass function $p(y)$, Shannon defined entropy(Shannon, 1948) as:

$$\textbf{Shannon's Entropy: } H(Y) = -\sum_{y} p(y) \log_2 p(y)$$

Entropy is an expression of disorder to the information and a measure of the average amount of information conveyed per message. Moreover, given that Y is a discrete, $H(Y)$ is bounded as $0 \leq H(Y) \leq \log(2K+1)$. This boundary allows the following two observations:

1) This lower bound on entropy corresponds to no uncertainty, as $H(Y) = 0 <=> p_k = 1$ for a k and zero for every other observation.

2) This upper bound on entropy corresponds to maximum certainty, as $H(Y) = \log(2K+1) <=> p_k = 1/(2K+1), \forall k \ni (2K+1)$.

In case of a continuous random variable Y, with probability mass function $p(y)$, entropy is defined as:

$$\textbf{Differential Entropy: } h(Y) = -\int_{-\infty}^{+\infty} p_Y(y) \log p_Y(y) dy = -\mathrm{E}[\log p_Y(y)]$$

From this basic concept, the other information measures (or entropy functions) can be formed. Given p(t, y) is the joint distribution for the target random variable T and prediction random variable Y , and p(t) and p(y) are called marginal distributions. We call them measures because some of them do not satisfy the metric properties fully, like KL divergence. The other entropy functions are given in the table below:

Name	Formula	(Dis)similarity
Joint Information	$H(T,Y) = -\sum_t \sum_y p(t,y) \log_2 p(t,y)$	Inapplicable
Mutual Information	$I(T,Y) = -\sum_t \sum_y p(t,y) \log_2 \frac{p(t,y)}{p(t)p(y)}$	Similarity
Conditional Entropy	$H(T\mid Y) = -\sum_t \sum_y p(t,y) \log_2 p(t\mid y)$	Dissimilarity
Cross Entropy	$H(T;Y) = -\sum_z p_t(z) \log_2 p_y(z)$	Dissimilarity
KL Divergence	$KL(T,Y) = \sum_z p_t(z) \log_2 \frac{p_t(z)}{p_y(z)}$	Dissimilarity

But, there are also other entropies beside Shannon's Entropy that have interesting applications such as:

$$\textbf{Renyi's Entropy: } H_a(Y) = \frac{1}{1-a} \log \int p^a(y) dy$$

Taken into account the "mathematical principles", learning machines can be distinguished into two groups. Those employing empirical formulas, like error rate or bound, cost (or risk), utility, or classification margins to achieve their task and those based on information theory .(Principe, 2010)Therefore, a systematic study seems necessary to answer the two basic questions below (Yao YY,2003):

1) When one of the principal tasks in machine learning is to process data, can we apply entropy or information measures as a generic learning target for dealing with uncertainty of data in machine learning?

2) What are the relations between information learning criteria and empirical learning criteria, and the advantages and limitations in using information learning criteria?

An answer to the first question comes from Watanabe (Watanabe, 1981), who proposed that “learning is an entropy-decreasing process” and pattern recognition is “a quest for minimum entropy”. The principle behind entropy criteria is to transform disordered data into ordered one (or pattern). Watanabe seems to be the first “to cast the problems of learning in terms of minimizing properly defined entropy functions”, and throws a brilliant light on the learning target selection in machine learning.

Regarding the second question, In 1988 Zellner completed a theoretical proof that Bayesian theorem can be derived from the optimal information processing rule (Zellner, 1988) , a proof that draws the first lines to reveal the roots of Bayesian theory on information and optimization concepts. Just to make a brief mention of it, we quote as it has been concluded “it has been shown that learning or information processing rules can be derived analytically by optimizing information criterion functions. Bayes’ theorem or information processing rule was shown to be an optimal rule both in static and dynamic learning contexts”. On Furthermore, Principe and his collaborators made a significant contribution, proposing Information Theoretical Learning (ITL) as a generic learning target in machine learning.

By this the introductory part of this essay is completed, as the relationship between ML and IT is now established and all the important concepts defined. We shall now focus on two basic facts of ML, one belonging to Unsupervised Learning and one to Supervised, Clustering and Classification. In the remaining of this essay we will furtherly discuss methods to accomplish these tasks, analytical mathematical entropy-based algorithms will be presented and discussed for both tasks and practical stimulations will be provided. In addition, an innovative approach that enables learning rate update in Neural Networks back-propagation learning rule will be presented in the end.

Classification

As defined above, classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known. This prior knowledge of the category assigned to each observation of the data set refers to Supervised Learning.

Often, the individual observations are analyzed into a set of quantifiable properties, known variously as explanatory variables or features. These properties may be categorical (e.g. gender), ordinal (e.g. "large", "medium" or "small"), integervalued (e.g. age) or real-valued (e.g. ratios). Extracting the right features according to categories distinction is crucial, as each classification is dependent in deferent parameters. For example having a group of people, if the classification criterion is gender, age is an irrelevant feature that may gain more confusion that efficiency to the algorithm. **Principle Component Analysis (PCA)** based on cross- entropies as well as **Independent Component Analysis (ICA)** are algorithms based on IT that seek to extract these main features of train set. These algorithms will not be examined in this essay but the reader is refer to Haykin's "Neural Networks and Machine Learning", pages 510-545, for additional information.

Other classification algorithms, called classifiers, work by comparing observations to previous observations by means of a similarity or distance function. In the sense of this kind of classifiers we will introduce below **(Dis)similarity measures derived from IT** and the **Maximum Entropy Classifier.**

(Dis)similarity measures

The importance of similarity and dissimilarity measures in terms of classification is obvious, as maximizing or minimizing these measures can form the decision functions for a classification task. For example, setting as the classification rule the demand that each observation will be assigned to the set that contains the majority of observations that have similarity measures higher to a bias with it. Hu(Hu,2015) developed a theory using the entropy functions presented above as such measures, and related them with exact classification. Specifically, he introduced Mutual Information as a similarity measure, and Conditional Entropy, Cross Entropy and KL Divergence as Dissimilarity ones.

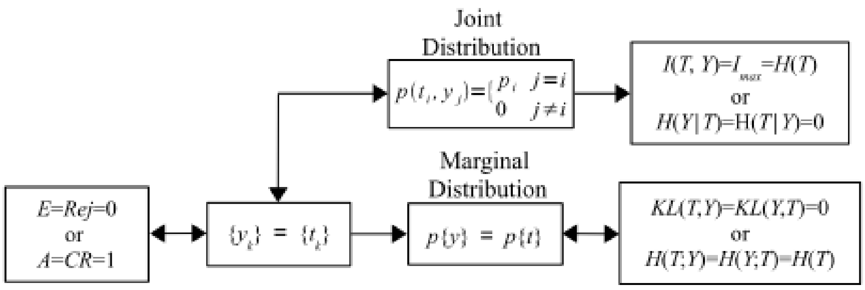
Given the variable T providing a ground truth statistically, in classification's case the Target, that is the correct category that an observation belongs to, the its entropy H(T) is the baseline of learning. So, setting Y as the variable that describes the predictions for each observation, one can declare that the measures reach the baseline of H(T) when the following relations hold:

$$I(T,Y) = H(T;Y) = H(Y;T) = H(Y) = H(T)$$
$$KL(T,Y) = KL(Y,T) = H(Y|T) = H(T|Y) = 0$$

Moreover, using the notations E:= Error, Rej:=Reject, A:=Accuracy, CR:= Correct Recognition Rates, Hu and his partners(B.-G. Hu, He, & Yuan, 2011) also prove the following relations:

$$CR + E + \mathrm{Re}\,j = 1$$
$$A = \frac{CR}{CR + E}$$

and in the sense of $\{y_k\} = \{t_k\}$ describes an equality between the label variables in every sample, that is the for each observation there is a label referring to the target-category and a label for the category it was assigned to by the classifier, the following table describes the connection between entropy measures and classification.



The following conclusion derives from these relations: The necessary condition of exact classifications is that all the information measures reach the baseline of H(T), that is an identical correspondence between Y and T. Therefore, classification criterion as well as model efficiency estimators can be fully replaced by entropy functions.

Maximum Entropy Distribution

When it comes to natural problems, features selection is not always a choice as the data can be extremely complex and the classification parameter is not always straightforward. Simple similarity or dissimilarity measures are not efficient in such case. A more advanced approach is the usage of probability distributions to define relations and accomplish classification of the data. That arises a new problem though, as probabilities are not always provided or easily estimated, leading as to cope with stochastic systems, enriched with constraints on the probability distribution of the states. The constraints can be certain ensemble average values or bounds on these values.

The problem is to choose a probability model that is optimum in some sense, given this prior knowledge about the model. We usually find that there are infinite number of possible models that satisfy the constraints. Which model should we choose? The answer to this fundamental question lies in the **Maximum-entropy Principle** given by Jaynes (Jaynes, 1957):

When an inference is made on the basis of incomplete information, it should be drawn from the probability distribution that maximizes the entropy, subject to constraints on the distribution.

So, basically entropy defines a measure on the space of probability distributions such that those distributions with the highest entropy are considered as optimal among the others. Following this statement, a constrained- optimization problem is formed. An outline of the procedure followed to cope with such problems is given below(Csiszár, 1996):

Given a random variable Y, we aim to maximize the differential entropy of Y, that is maximizing the function:

$h(Y) = -\int_{-\infty}^{+\infty} p_Y(y) \log p_Y(y) dy$, for each $p_y(y)$ with respect to these three constrains:

$$p_Y(y) \geq 0$$
$$\int_{-\infty}^{+\infty} p_Y(y) dy = 1$$
$$\int_{-\infty}^{+\infty} p_Y(y) g_i(y) dy = a_i, i = 1, 2, \ldots.., m$$

where g_i is any function of y. The 3rd constrain is the one that defines the moments of Y, according to g_i, taking into account all the available previous knowledge on Y. Obviously, this constrained- optimization problem can be solved using the method of Lagrange Multipliers. Finally, the **Maximum Entropy Distribution** is found to be:

$p_Y(y) = \exp(-1 + \lambda_o + \sum_{i=1}^{m} \lambda_i g_i(y))$, where $\lambda_i, i = 1, 2, \ldots., m$ refers to the Lagrange Multipliers, chosen with respect to the above constrains.

For a detailed solution of the problem the reader is refers to "Neural Networks and Machine Learning" (S. S. Haykin, 2009), pages 481-483.

Algorithm

In maximum entropy we use the training data to set constraints on the conditional distribution. Each constraint expresses a characteristic of the training data that should also be present in the learned distribution. Let's suppose we have a train data D , a set of classes C and a set of observations to be classified.

The first step is to set a real-valued function of each observation and its class be a feature, $f_i(d,c): \square^2 \to \square$. How this function will be defined depends entirely on the problem's particularities and data's form. Maximum entropy addresses this function in a way that permits a restriction of the model distribution, ensuring the same expected value for this feature as seen in the training data, D .

If we express this request into a mathematical form, the learned conditional distribution $P(c \mid d)$ must have the property:

$$\frac{1}{|D|}\sum_{d\in D} f_i(d,c(d)) = \sum_d P(d)\sum_c P(c\mid d)\, f_i(d,c)$$

$P(d)$ stands for the distribution of the data to be classified, which in practice is unknown. This can be overcome by using our training data, without class labels, as an approximation to the document distribution, enforcing the constraint:

$$\frac{1}{|D|}\sum_{d\in D} f_i(d,c(d)) = \frac{1}{|D|}\sum_{d\in D}\sum_c P(c\mid d)\, f_i(d,c)$$

Therefore, the first step is to identify a set of feature functions that will be useful for classification. Second step is to measure the expected value of each one of these features over the train data, forming the constraints for the model distribution.

The third step, once constraints are estimated, is to apply the Maximum Entropy Principle for the learned conditional distribution (Pietra, Pietra, & Lafferty, 1997):

$$P(c\mid d) = \frac{1}{Z(d)}\exp\Big(\sum_i \lambda_i f_i(d,c)\Big)$$

where each $f_i(d,c)$ is a feature, λ_i is a parameter to be estimated and $Z(d)$ is simply the normalizing factor to ensure a proper probability:

$$Z(d) = \sum_c \exp\Big(\sum_i \lambda_i f_i(d,c)\Big)$$

There is a basic disadvantage and several advantages in this method to discuss. The disadvantage, to begin with, is that Maximum entropy is apt to overfitting, due to its heavy reliance on the labeled training data, which forms the constrains. With too little data, the expected value of a feature in the training data may be far from the true value. Several studies propose utilizing a prior on the model to reduce overfitting and improve the performance.

On the other hand, when the constraints are estimated from labeled training data, the solution to the maximum entropy problem is also the solution to a dual maximum likelihood problem for models of the same exponential form. (Nigam, Lafferty, & Mccallum, 1999). Additionally, since the likelihood surface is convex, that guarantees a unique global maximum and rules out local maxima. Thus, there is one possible approach for finding the maximum entropy solution: set any initial exponential distribution of the correct form as a starting point and perform hill climbing in likelihood space. Since there are no local maxima, this will converge to the maximum likelihood solution for exponential models, which will also be the global maximum entropy solution.

The other major advantage is that maximum entropy has nothing to do with independence assumptions. This may be clearer if illustrated by an example: consider the phrase “Puerto Rico”. In most of the cases, these two words co-occur, rarely standing for themselves. As far as Naive Bayes, the evidence of this phrase will be double-counted. When it comes to Maximum entropy though, λi for each of these features will be discounted such that their weight towards classification will be appropriately reduced by half, thanks to the constraints work over expectations of the counts.

Applications

Maximum entropy is widely used for a variety of natural language tasks, such as language modeling, part-of-speech tagging, and text segmentation. Some of these applications shortly described are:

- Nigam, Laffery and McCallumn (Nigam et al., 1999) proposed an Maximum Entropy based model for Text Classification by estimating the conditional distribution of the class variable given to each document, and experimented on several databases, comparing their results to naïve Bayes classifier.
- Osborne (Osborne, 2002), presented a maximum entropy classifier used to extract sentences from documents and after stimulations compared classifiers efficiency with previously used classifiers.
- Berger and his collaborators (Berger, Pietra, & Pietra, 1996), describe a method for statistical modeling based on maximum entropy, presenting a maximum-likelihood approach for automatically constructing maximum entropy models and describe how to implement this approach efficiently, using as examples several problems in natural language processing.
- Zhu, Ji and Xu (Zhu, Ji, Xu, & Gong, 2005) explore correlations among categories with maximum entropy method and derive a classification algorithm for multi-labelled documents

Besides language related issues, the algorithm is also used for biological and ecological problems that require prediction models. Such examples are:

- Phillips, Anderson and Schapire (Phillips, Anderson, & Schapire, 2006) use of the maximum entropy method (Maxent) for modeling species geographic distributions with presence-only data, predicting missing or non-existing data of rare species .
- Yeo and Burge(Yeo & Burge, 2004) propose a framework for modeling sequence motifs based on the maximum entropy principle, recommending the approximation of short sequence motif distributions with the maximum

entropy distribution consistent with low-order marginal constraints estimated from available data.

Clustering

Clustering is the problem of partitioning a finite set of points (observations) in a multi-dimensional space (its dimension depends on the features of each observation) into classes (clusters) so that each clusters contains points similar to each other and dissimilar to the ones of other clusters.

So basically, clustering is the generalization of classification when training labels are not obtained therefore it belongs to unsupervised learning. Sharing the same philosophy, cluster analysis itself doesn't correspond to a specific algorithm, but a general task to be solved. There exist various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions.

Therefore, clustering can be described as a multi-objective optimization problem. It is the individual data set as well as the nature of each problem that configure the appropriate algorithm and the parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters).

Clustering has been divided into two approaches, the **partitional** and the **hierarchical** algorithms. A partitional clustering algorithm obtains a single partition of the data, often by minimizing a cost function, between patterns and cluster centroids. The particularity of this method is the requirement of a-priori knowledge of the number of clusters to be formed. A hierarchical clustering algorithm on the other hand, produces a hierarchy of clusters, enabling different sets of clusters to be obtained at different levels. Similarity between clusters can be defined in several ways, often resulting in different clustering results for different similarity measures.

In accordance to all these different approaches of a clustering problem, several entropy-based algorithms will be presented.

Error-Entropy Minimization

Given a set of observations as a training data, the entropy of the estimation error over the training data's clustering by an algorithm must be minimized. The interpretation of this is as follows. When entropy of the error is minimized, the expected information contained in the estimation error is minimized; hence the clustering model is trained optimally in the sense that the mutual information between the train data and the model output is maximized. This is the philosophy behind entropy minimization criterion.

Minimum entropy criterion faces to two sub-problems, when applied to deal with clustering problems: (i) estimating a posteriori probabilities and (ii) minimizing the entropy. Although some parametric method could be employed to estimate a posteriori probabilities, a nonparametric approach, proposed by Li, Zhang and Jiang (H. Li, Zhang, & Jiang, 2004) will be presented in this essay. A merit of the nonparametric approach is that it does not require much prior information (e.g. distribution) of the data. In order to minimize the entropy, an efficient iterative algorithm is used, which usually converges in a few steps. The basic steps of the constructing the clustering algorithm are:

1) Define Minimum Entropy Criterion
2) Prove it by Fano's Inequality
3) Use a non-parametric estimation of a posteriori probability
4) Combine to create the iterative algorithm

1) Define Minimum Entropy Criterion

As defined above, **Shannon's entropy** of a random variable $X = \{x_1, x_{2,\dots}\}$ with probability mass function $p(x)$ is:

$$H(X) = -\sum_x p(x) \log p(x) \quad (1)$$

Since entropy measures the amount of "disorder" of a system, we hope that each cluster has low entropy because data points in the same cluster should look similar. Thus, we would like to employ some form of entropy in the objective function of clustering. A **straightforward minimum entropy clustering criterion** could be defined as:

$$J = \sum_{j=1}^{m} p_j H(X \mid c_j) \quad (2)$$

where $H(X \mid c_j)$ is the entropy of cluster c_j, and p_j is the probability of cluster c_j such that $\sum_{j=1}^{m} p_j = 1$.Suppose each cluster c_j follows the d-dimensional Gaussian distribution with covariance matrix Σ_j. So,

$$H(X \mid c_j) = \log(2\pi e)^{d/2} + \frac{1}{2} \log |\Sigma_j|$$

and by discarding additive constant $\log(2\pi e)^{d/2}$ the **clustering criterion to minimize** is:

$$J = \frac{1}{2} \sum_{j=1}^{m} p_j \log |\Sigma_j| \quad (3)$$

However, it is actually not adequate for clustering because it neglects the semantic aspects of data. Note that the data contain some hidden meaning. In clustering/classification, the semantic information that we care about is the category of objects. The task of machine learning is just to infer this categorical information. Entropy was developed without the consideration of the meaning of data because the concept of entropy was developed originally for communication where the meaning of messages is irrelevant to the transmission of messages. Thereby, a good minimum entropy clustering criterion has to reflect the relationship between data points and clusters. Such relationship information helps us to reveal the meaning of data, i.e. the category of data. Besides, it also helps us to identify the components, i.e. clusters, of mixed information source. Since the concept on which entropy measures are based is similar to that of probabilistic dependence, an entropy measured on a posteriori probabilities could be such a suitable quantity, defined:

$$H(C|x) = -\sum_{j=1}^{m} p(c_j|x)\log p(c_j|x) \qquad (4)$$

Let C is the random variable of category taking values in $\{c_1, c_2, ..., c_m\}$ with probabilities $p_1, p_2, ..., p_m$. In (4), we compute a posteriori probabilities $p(c_j|x)$ to determine how much information has been gained. $H(C|x)$ is maximized when all $p(c_j|x)$ are equal. In this case, the object x could come from any source (i.e. cluster) equally probably, and thus we do not know which cluster the object x should belong to. This is also intuitively the most uncertain situation. On the other hand, $H(C|x)$ is minimized to 0 if and only if all $p(c_j|x)$ but one are zero, this one having the value unity. That is, we are certain of the cluster of x only if $H(C|x)$ vanish. Thus, $H(C|x)$ can assess the dependence between x and C well. By integrating x on the whole data space, we obtain **the minimum entropy clustering criterion:**

$$H(C|x) = -\int \sum_{j=1}^{m} p(c_j|x)\log p(c_j|x)p(x)dx \qquad (5)$$

The above quantity is actually the conditional entropy of the random variable C given the random variable X . The conditional entropy $H(C|x)$ measures how uncertain we are of C on the average when we know X.

2) Fano's Inequality

Fano's inequality provides a strong evidence that minimum $H(C|x)$ could be a good clustering criterion. Suppose we know a random variable X and we wish to guess the value of the correlated category information C. Fano's inequality relates the probability of error in guessing the random variable C to its conditional entropy

$H(C \mid x)$. Suppose we employ a function $\hat{C} = f(x)$ to estimate C. Define **the probability of error** as $P_e = P\{\hat{C} \neq C\}$. **Fano's Inequality** states that:

$$H(P_e) + P_e \log(m-1) \geq H(C \mid x) \qquad (6)$$

The proof can be found and explained in "Elements of Information Theory" by Cover and Thomas(Cover & Thomas, 2005), pages 38-40.

The explanation of how this proof makes the above criterion a strong one for clustering is simple. Note that P_e = 0 implies that $H(C \mid x)$ = 0. In fact, $H(C \mid x)$ = 0 if and only if C is a function of X. Fano's inequality indicates that we can estimate C with a low probability of error only if the conditional entropy $H(C \mid x)$ is small. In machine learning, P_e is expected to be small by the implicit assumption that X contains adequate information about C, i.e $H(C \mid x)$ is small. Thus, the minimum $H(C \mid x)$ is a natural criterion for clustering.

3) Use a non-parametric estimation of a posteriori probability

To estimate $p(c_j \mid x)$, we could employ some parametric method. However, it may not be appropriate for clustering complicated because the choice of any particular distribution could lead to a very poor representation of the data if the data has a complex structure. We therefore seek a nonparametric method for modeling the data. There are two kinds of nonparametric estimation techniques, Parzen's window density estimation[1] and k-nearest neighbor density estimate. They are fundamentally very similar, but exhibit some different statistical properties. An advanced approach to estimate the priors using a model derived from IT was proposed by Palubinskas and coworkers (Palubinskas, Descombes, & Kruggel, 1998).In what follows, we give a brief overview of the first two nonparametric density estimation methods.

Consider estimating the value of a density function p(x) at a point x. We may set up a small window R(x)(e.g. hyper-cube or hyper-ellipsoid) around x. Then, the probability mass of R(x) may be approximated by p(x) · v where v is the volume of R(x). On the other hand, the probability mass of R(x) may also be estimated by drawing a large number (say n) of samples from p(x), counting the number (say k) of samples falling in R(x), and computing k/n. Equating these two probabilities, we obtain an **estimate of the density function** as:

$$p(x) = \frac{k}{nu} \qquad (7)$$

[1] See "Pattern Recognition"(Theodoridis & Koutroumbas, 2009) pages 61-62.

If we fix the volume u and let k be a function of x, we obtain Parzen density estimate. On the other hand, we may fix k and let v be a function of x. More precisely, we extend the region R(x) around x until the k-th nearest neighbor is found. This approach is called the k-nearest neighbor density estimate. By Bayes's rule, we have

$$p(c_j \mid x) = \frac{p(c_j)p(x \mid c_j)}{p(x)} \qquad (8)$$

We may use n_j/n as the estimator of $p(c_j)$, where n_j is the number of points in cluster c_j. If **Parzen density estimate** is employed, we have:

$$p(c_j \mid x) = \frac{\frac{n_j}{n}\frac{k(c \mid c_j)}{n_j u}}{\frac{k(x)}{nu}} = \frac{k(x \mid c_j)}{k(x)} \qquad (9)$$

Thus, the estimate of $p(c_j \mid x)$ is just the ratio between the number of samples from cluster c_j and the number of all samples in the local region R(x). **If k-nearest neighbor estimate** is used, we obtain:

$$p(c_j \mid x) = \frac{\frac{n_j}{n}\frac{k}{n_j u(x \mid c_j)}}{\frac{k}{nu(x)}} = \frac{u(x)}{u(x \mid c_j)} \qquad (10)$$

4) The iterative algorithm

It is now about time to develop a clustering algorithm to optimize the criterion (5). However, the criterion (5), where $p(c_j \mid x)$ can be estimated by either of the above estimations, given by (9) or (10), is not suitable for directly clustering the data because we can minimize $H(C \mid x)$ to 0 by simply putting all data points into one cluster. Such an optimal solution is trivial and interferes with finding the practically useful partitions. Li and Ogihara proposed and proof a theorem that states : Any clusters process decreases the entropy and therefore the goal of clustering can be to find a partition C such as the increase in H(X)-H(C) is maximized, that is, again, minimize H(C) (T. Li & Ogihara, 2004)

Thus, instead of directly clustering the data, we will present an iterative algorithm to reduce the entropy of an initial partition given by any other clustering methods (e.g. k-means). That is, the algorithm searches a local optimal solution starting from a partition given by some other method. This algorithm can be easily modified for optimizing the criterion with the k-nearest neighbor estimation. In our case, an intuitive idea to update the partition is to assign a data object x to the cluster of its neighbors. This reassignment actually decreases the entropy associated with x because any change toward unequalization of the probabilities decreases the entropy.

Suppose a point x is assigned to the cluster c_i currently and most neighbors of x are assigned to the cluster $c_j = c_i$. Moreover, suppose n_i neighbors of x belong to c_i and n_j neighbors of x belong to c_j such that n_i < nj. After x is reassigned to cluster c_j, the difference between n_i − 1 and n_j + 1 is larger than that of n_i or n_j. So, we make the difference between the probabilities $p(c_j \mid x)$ and $p(x \mid c_j)$ larger, and thus reduce the entropy associated with x. Such an update, however, does not necessarily decrease the total entropy of the partition. Note that the entropy associated with the neighbors of x also changes after the reassignment. The total entropy of partition could increase although the entropy associated with x decreases. Based on this observation, the constructed iterative algorithm considers the change of entropy associated with all neighbors of x, and sums up in the following:

Input: A dataset containing n objects, the number of clusters m, and an initial partition given by some other clustering method.
Output: A set of at most m clusters that locally minimizes the conditional entropy.
Method: For every object x in the dataset:
1) If the cluster c_j containing most of the neighbors of x is different from the current cluster c_i of x, then define

$h = \sum_y H'(C \mid y) - H(C \mid y))$, where y are neighbors of x, and x is also regarded as the neighbor of itself. $H(C \mid y)$ and $H'(C \mid y)$ are the entropy associated with y before and after assigning x to the cluster c_j, respectively.

2) If h<0 then assignee x to cluster c_j.

3) Repeat until no change is done.

Renyi's Entropy

This section presents a new clustering algorithm using Renyi's entropy as similarity metric for clustering, proposed by Principe, Jenssen, Erdogmus and Eltoft

(Jenssen, Ii, Erdogmus, Principe, & Eltoft, n.d.). Renyi's entropy lends itself nicely to non-parametric estimation, overcoming the difficulty in evaluating traditional entropy metrics. It enables the utilization of all the information contained in the distribution of the data, and not only mere second order statistics as many traditional clustering algorithms are limited to.

The main idea is to assign a data pattern to the cluster, which among all possible clusters increases its within-cluster entropy the least, upon inclusion of the pattern. This procedure is introduced as **differential entropy clustering**. Before proceeding to the mathematical algorithm, a scheme of the model is cited, to assist in reader's better understanding.

1) A number of small clusters are "seeded" randomly in the data set, labeling a small subset of the data, to overcome the fact that there exists no previous knowledge of the number of clusters.
2) All remaining patterns are to be labeled by differential entropy clustering.
3) The "worst cluster" is defined by a quantity named the between-cluster entropy. Its members are re-clustered, again by differential entropy clustering, reducing the overall number of clusters by one.
4) This procedure is repeated until only two clusters remain. At each step we store the current labels, thus producing hierarchy of clusters.
5) The between-cluster entropy can be used to select our final set of clusters in the cluster hierarchy.

1) Initializing the clusters

Initially K clusters are "seeded" in the dataset. This is done by first randomly selecting K observations from the data set, each initially representing a cluster. Thereafter the point closest to any member of a cluster is included in the cluster, until a preselected value of N patterns have been assigned to each cluster. The grouping is done this way instead of just finding the N nearest neighbors to the "seed" pattern, since this makes the grouping more sensitive to the data structure.

2) Differential Entropy Clustering

Consider a set of feature vectors distributed in feature space. Initially a subset of the feature vectors have been assigned to cluster among the K initialized ones. The problem of clustering is now to decide whether a new pattern X should be assigned to each of these clusters. The proposed clustering is based on a simple observation. If X is wrongly assigned to m-th cluster, then the uncertainty or entropy of m-th cluster will increase more than the entropy of the cluster in which X truly belongs.

Hence, in the general case of having initial clusters C_k, $k=1,2,....,K$, assign X to a cluster C_i if:

$$H(C_i + x) - H(C_i) < H(C_k + x) - H(C_k), \forall k = 1, 2, ..., K, k \neq i \qquad (1)$$

Previously, entropy has been a metric difficult to evaluate without imposing unrealistic assumptions about the data distributions. Recently it was discovered that an entropy measure proposed by Renyi lends itself nicely to non- parametric estimation directly from data. **Renyi's entropy** for a stochastic variable X with probability density function (pdf) f_x is given by:

$$H(X) = \frac{1}{1-a} \log \int f_x^{\ a}(x) dx, a > 0 \qquad (2)$$

Specifically, for a=2 we obtain **Renyi's quadratic entropy**:

$$H(X) = -\log \int f_x^{\ 2}(x) dx \qquad (3)$$

This expression can easily be estimated directly from data by the use of Parzen window density estimation (see"Sergios Theodoridis, Konstantinos Koutroumbas Pattern recognition 2003,"pages 61-62), with a multidimensional Gaussian window function. Assume that cluster C_k consists of the set of discrete data points $x_i, i = 1, 2, ..., N_k$. Now, the **pdf estimate** based on the data points of C_k is given by:

$$\hat{f}_x = \frac{1}{N_k} \sum_{i=1}^{N_k} G(x - x_{i,} \sigma^2 \mathrm{I}) \qquad (4)$$

where N_k is the number of data points in C_k and we have used a symmetric Gaussian kernel with covariance matrix $\Sigma = \sigma^2 \mathrm{I}$. By substituting (4) into (3), and utilizing the properties of the Gaussian kernel, we obtain an estimate of the entropy of C_k, called **within-cluster entropy**:

$$H(C_k) = -\log(V(C_k)) \qquad (5)\text{ , where}$$

$$V(C_k) = \frac{1}{N^2_{\ k}} \sum_{i=1}^{N_k} \sum_{j=1}^{N_k} G(x - x_{j,} 2\sigma^2 \mathrm{I}) \qquad (6)$$

4)"Worst Cluster" Selection

Instead of using (6) to compute the entropy of each cluster individually, we modify it such that the double sum runs over all data points, and includes a membership function, $M(x_{i,} x_j)$ which equals to one if x_i and x_j belongs to different clusters, and zero otherwise. We name the resulting expression the **between- cluster entropy**, which is given by:

$$H(C_1, ..., C_k) = -\log V(C_1, ..., C_k) \qquad (7)\text{, where}$$

$$V(C_1,....,C_k)=\frac{2}{2\prod_{k=1}^{K}N_k}\sum_{i=1}^{N}\sum_{j=1}^{N}M(x_i,x_j)G(x_i-x_j,2\sigma^2\mathrm{I}) \qquad (8)$$

If the clusters are well separated, $V(C_1,....,C_k)$ will have a small value, consequently $H(C_1,....,C_k)$ will have a large value. Proceed by eliminating one cluster at a time, and calculate the between- cluster entropy based on the remaining clusters in each case. To be more specific, by eliminating, we mean that the membership function is set to zero whenever acting on any of the members of the eliminated cluster.

The "worst cluster" is now selected as the cluster that when eliminated, results in the largest between-cluster entropy based on the remaining clusters, because this means that the remaining clusters are the most separated clusters.

4-5) Terminating the algorithm

Note that before each time the members of a cluster are re-assigned labels, and the number of clusters is reduced by one, all patterns have been labeled. Thus at each step a set of clusters exist. We store the cluster labels at each step, thus producing a hierarchy of cluster assignments. We continue this procedure until only two clusters remain. The issue now is to decide where in the hierarchy to select our final clustering**.**

Cross-Entropy

The main idea on the application of cross-entropy on clustering problems lays on the Cross-Entropy algorithm (CE) for optimization problems. Therefore, a brief introduction to CE will first be presented, following Kroese, Rubinstein and Taimre's work (Kroese, Rubinstein, & Taimre, 2007), and then the clustering problem will be introduced as an optimization problem, so the CE will create a clustering algorithm, called CEC, according to both Kroese, Rubinstein and Taimre as well as Tabor and Spurek's developed algorithms. (Tabor & Spurek, n.d.)

The CE method

The main idea of the CE method for optimization can be stated as follows: Suppose, we wish to maximize some performance function S(x) over all states x in some set X. Let us denote the maximum by γ ∗, thus

$$\gamma^* = \max_{x\in X} S(x) \qquad (1)$$

First, we randomize our deterministic problem by defining a family of sampling probability mass functions $\{f(.;v), v\in V\}$ on the set X. We assume that this family includes the degenerate density at x∗ , say $f(.;v^*)$. Next, we associate with (1) estimation problems of the form

$$l(\gamma) = P_u(S(x) \geq \gamma) = E_u I_{\{S(X)\geq\gamma\}} \qquad (2)$$

Here, X is a random vector with pdf $f(.;u)$, for some u ∈ V and γ ∈$\Re$. This is called the associated stochastic problem (ASP).

Having defined an ASP, the goal of the CE algorithm is to generate a sequence of tuples $\{(\gamma_t, v_t)\}$ that converge quickly to a small neighborhood of the optimal tuple (γ ∗, v∗).More specifically, we choose some initial v_0 and a not too small ρ ,say $\rho = 10^{-2}$, and proceed as follows:

1. Adaptive updating of γ_t : For a fixed v_{t-1}, let γ_t be the (1 − ρ)-quantile of S(X) under . A simple estimator $\hat{\gamma}_t$ of γ_t can be obtained by drawing a random sample $X_{1,}X_2,.....,X_N$ from $f(.;v_{t-1})$ and evaluating the sample (1 −ρ)-quantile of the performances:

$$\hat{\gamma}_t = S_{([(1-\rho)N])} \quad (3),$$

Where $S_{(k)}$ denotes the k-th order statistic of {S(Xi)}.

2. Adaptive updating of v_t : For fixed γ_t and v_{t-1} ,derive v_t from the solution of the formula:

$$\max_v D(v) = \max_v E_{v_{t-1}} I_{\{S(X)\geq\gamma_\tau\}} \ln f(X;v) \qquad (4)$$

The stochastic counterpart of (4) is as follows: for fixed $\hat{\gamma}_t$ and $\hat{v}_{t-1}$, derive $\hat{v}_t$ from the following formula:

$$\max_v \hat{D}(v) = \max_v \frac{1}{N}\sum_{i=1}^{N} I_{\{S(X_i)\geq\hat{\gamma}_t\}} \ln f(X_i;v) \quad (5)$$

Thus, the main CE optimization algorithm, which includes smoothed updating of parameter vector v can be summarized as follows:

Main CE Algorithm for Optimization:

1. Choose some v_0 .Set t = 1 (level counter).
2. Generate a sample $X_{1,}X_2,.....,X_N$ from the density $f(.;v_{t-1})$ and compute the sample (1−ρ)-quantile $\hat{\gamma}_t$ of the performances according to (2).

3. Use the same sample $X_{1,} X_2, \ldots\ldots, X_N$ and solve the stochastic program (5).Denote the solution by $\overset{*}{v_t}$.

4. If for some t ≥ d, say d = 5

$$\overset{\wedge}{\gamma}_t = \overset{\wedge}{\gamma}_{t-1} \ldots = \overset{\wedge}{\gamma}_{t-d} \qquad (6)$$

Then (6) stops (let T denote the final iteration); otherwise set t = t + 1 and reiterate from step 2.

Note that the parameter v is updated in (5) only on the basis of the (1−ρ)% best samples. These are called the elite samples. The main ingredients of any CE algorithm are a) Trajectory generation: Generate random samples in X according to $f(.;v_{t-1})$ and b) Parameter updating: Update v on the basis of the elite samples, in order to produce better performing samples in the next iteration.

The clustering problem as a continuous multi-external problem

A CE approach for solving the clustering problem is now presented, by viewing it as a continuous multi-extremal optimization problem where, as in the k-means method, the centroids $c_1, c_2, \ldots., c_K$ (therefore we have K clusters) are the decision variables. In short, we consider the program **decision function:**

$$\min_{c_1, c_2, \ldots., c_K} S(c_1, c_2, \ldots., c_K) = \min_{c_1, c_2, \ldots., c_K} \sum_{j=1}^{K} \sum_{z \in R_j} \| z - c_j \|^2 \qquad (7)$$

Where $R_j = \{z : \| z - c_j \| < \| z - c_k \|, k \neq j\}$. That is, R_j is the set of data points that are closer to c_j than to any other centroid. Note that the performance (or objective) function S to minimize is a real-valued function on $X = \square^{dK}$, where d is the dimension of each observation. In order to apply the CE method, we need to specify (a) a parametric family of sampling distributions $\square^{dK}$ and (b) the updating rules for the parameters of the sampling distribution.

The latter involves the Kullback–Leibler or CE distance, and results typically in updating the parameters via the maximum likelihood estimates of the elite (i.e., best) samples. It is important to realize that the choice of the sampling distribution is quite arbitrary and has nothing to do with the (suspected) distribution of the cluster data. We choose the sampling distribution to be dK-dimensional Gaussian of a specific form (to be explained next). The reason for choosing a Gaussian distribution is that the updating formulas become particularly simple.

For better insight and easy reference we consider the program (7) with K = 2 clusters and dimension d = 2. The sampling distribution is such that, we independently sample random centroids C_1 and C_2 according to 2-dimensional normal distributions $N(\mu_1, \Sigma_1)$ and $N(\mu_2, \Sigma_2)$ respectively.

The parameter vector v in CE Algorithm now consists of the mean vectors and covariance matrices $\{\mu_1, \mu_2, \Sigma_1, \Sigma_2\}$. As in a typical CE application for continuous multi-extremal optimization, we set the initial matrices Σ_1 and Σ_2 to be diagonal (with quite large variances on the diagonals) and then, we proceed according to CE algorithm and the procedure can be generalized for K clusters as follows:

1) Choose, deterministically or randomly, the initial mean vectors and covariance matrices $\mu_i, \Sigma_i, i = 1, 2, ...K.$
2) Generate K sequences of centroids $Y_{k1}, ...Y_{kN}, k = 1, 2, ..., K$, according to $Y_{ij} \sim N(\mu_j, \Sigma_j), j = 1, 2, ..., K$, independently. For each k = 1, ... ,N calculate the objective function as in (7), with c_j replaced by Y_{ij}.
3) Complete Steps 2 and 3 of CE algorithm. In particular, update the parameters (μ_1, μ_2) and (Σ_1, Σ_2) according to the Kullback–Leibler cross-entropy.
4) If the stopping criterion is met, then stop and accept the resulting parameter vector (μ_{1T}, μ_{2T}) (at the final T-th iteration) as the estimate of the true optimal solution.

Stopping criterion: There are many possible variations to the standard stopping criterion in Step 4 of CE Algorithm. In particular, for continuous optimization, criterion (6) may not be appropriate, as $\overset{\wedge}{\gamma}_t$ may, sometimes, never be equal, thus preventing the algorithm from stopping. An alternative, for the present problem, is to stop when the maximum of the diagonal elements in Σ_1 and Σ_2 is less than some n, say 10^{-4}.

Starting positions: One advantage of the CE method is that, as a global optimization method, it is very robust with respect to the initial positions of the centroids. Provided that the initial standard deviations are chosen large enough, the initial means have little or no effect on the accuracy and convergence speed of the algorithm. In general, we choose the initial means and standard deviations such that the initial sampling distribution is fairly “uniform” over the smallest rectangle that contains the data points. Practically, this means that the initial standard deviations should not be too small, say equal to the width or height of this “bounding box.” For the KM method, however, a correct choice of starting positions is essential. A well-known data-dependent initialization method is to generate the starting positions independently, drawing each centroid from a d-dimensional Gaussian distribution N(μ,Σ),where μ is the sample mean of the data and Σ the sample covariance matrix of the data.

Applications

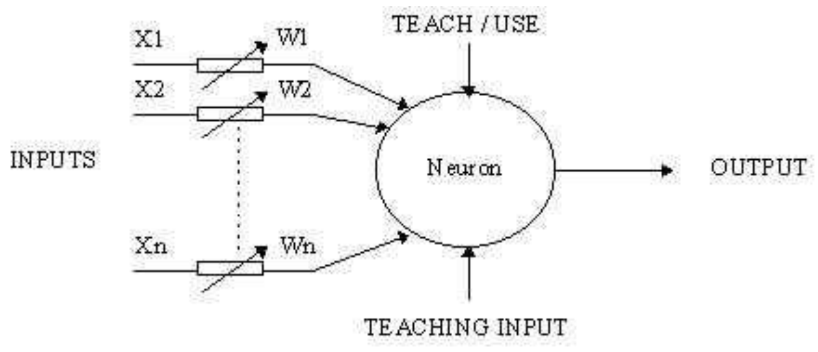
- Li, Zhang and Jiang (H. Li et al., 2004) used entropy based clustering to achieve gene expression data and analyze it. They also proposed the minimum entropy criterion, which is the conditional entropy of clusters given the observations and generalized it by replacing Shannon's entropy with Havrda-Charvat's structural alpha-entropy.
- Małyszko and Stepaniuk(Małyszko & Stepaniuk, 2008) addressed the problem of combining the concept of rough sets and entropy measure in the area of image segmentation. Segmentation presents the low-level image transformation routine concerned with image partitioning into distinct disjoint and homogenous. Simultaneous combining entropy based thresholding with rough sets results in rough entropy thresholding algorithm.
- Li, Ma and Ogihara (T. Li & Ogihara, 2004) study the entropy-based criterion in clustering categorical data, that is data in which there is no inherited distance measures between data values. They conduct experiments to verify the efficiency of these new criteria and propose partition minimizing criterion.
- Sahoo, Wilkins and Yeager(Sahoo, Wilkins, & Yeager, 1997) present a general technique for thresholding of digital images based on Renyi's entropy. The proposed method includes previously proposed well known global thresholding methods. The effectiveness of the proposed method is demonstrated by using examples from the real-world and synthetic images.
- Zimmer, Tepper and Akselrod (Zimmer, Tepper, & Akselrod, 1996) implement a new thresholding technique, known as 'minimum cross entropy thresholding' for Segmentation in medical image analysis.. They present a multivariate extension of MCE in which the segmented variable (gray level) is replaced by a weighted combination of several image parameters. The results obtained are demonstrated for ultrasound images of ovarian cysts.

Neural Networks

The study on artificial neural networks is inspired by the structure and function of the human brain (Diamantaras, 2007). The aim is to simulate the brain's learning model to implement algorithms that are designed to learn and adapt continuously. Learning is achieved through training, a repetitive process of gradually adapting the network parameters to values suitable for the problem to be solved with sufficient success. With these optimal parameters the algorithm applies to the same type of problem, if it continues to solve it, until it is characterized as efficient. Such abstract algorithmic constructs fall within the field of computational intelligence with the general aim of learning, generalizing, grouping patterns, making decisions, developing optimal strategies, and more.

The structure of an artificial neural network can be aligned with a biological neuron, as it shares the same elements. Specifically, there are neural entry gates, which receive signals from other neurons, neuron exit gates, which emit signals to other neurons and synapses, the points of association between neuronal inlets and outlets. An engagement has a reinforcing role if it irritates it from producing information or suppressing it by blocking the production of information. The percentage of information transferred is called synaptic weight and is synapse is assigned with a **synaptic weight**.

The mathematical implementation of the idea was carried out in a simple model by McCullotchs and Pitts in 1940. A neuron can work binary, y = 0 when the neuron remains inactive, and y = 1 when the neuron acts. Synapses are described by the synaptic weights w_i , which are real numbers. Let $x_1, x_2, ..., x_n$ are the inputs of the neuron, then the sum that serves as an action criterion is given by the sum of the products u = $w_i x_i$. A simple neuron's representation is presented bellow for better understanding. In the case where u is greater than the fixed threshold θ, then only the neuron acts. It is the **Objective Function** of the neural network, or of each neuron or layer of neurons that appoints this threshold. Following the rapid growth of the industry, various variations and enrichments of this basic algorithm followed. The architecture of the network is based on an input matrix involving all templates, an output layer, and one or more hidden layers in between for the gradual calculation of the requested (Haykin, 2009).



The classification of neural networks is done according to the way of calculating the synaptic weights, with supervision or without supervision, according to the definitions of supervised and unsupervised learning given previously.

Concerning the learning rule, we will focus on the **backward propagation** of errors. Back propagation, is a common method of training artificial neural networks and used in conjunction with an optimization method such as gradient descent. The algorithm repeats a two phase cycle, propagation and weight update. When an input vector is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired output, using a loss function, and an error value is calculated for each of the neurons in the output layer. The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output.

Mutual Information as an Objective function

The problem thus becomes one of adjusting the free parameters (i.e., synaptic weights) of the system so as to optimize the mutual information. Depending on the application of interest, we may identify four different scenarios described as follows:

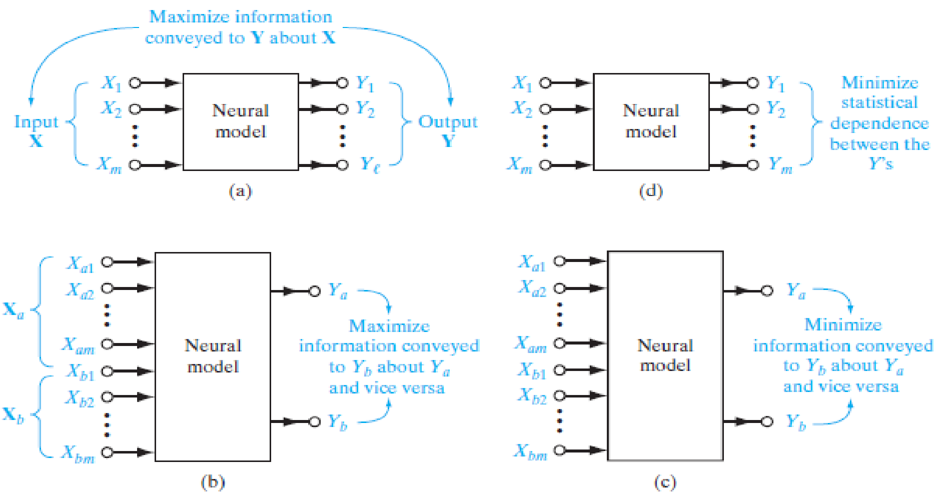
1) The input vector X is composed of the elements $x_1, x_2, ..., x_n$, and the output vector Y is composed of the elements $y_1, y_2, ..., y_l$. The requirement is to maximize the information conveyed to the system output Y about the system input X (i.e., the information flow across the system).
2) A pair of input vectors X_a and X_b is derived from adjacent, but non overlapping, regions of an image. The inputs X_a and X_b produce scalar outputs Y_a and Y_b , respectively .The requirement is to maximize the information conveyed to Y_b about Y_a and vice versa.
3) The input vectors X_a and X_b are derived from a corresponding pair of regions belonging to two separate, but related, images. The outputs produced by these two input vectors are denoted by Y_a and Y_b , respectively.

The objective is to minimize the information conveyed to Y_b about Y_a and vice versa.

4) The input vector X and the output vector Yare defined in a manner similar to the first case, but with equal dimensionality. The objective here is for the statistical dependence between the components of the output vector Y to be minimized.

In all four situations, mutual information plays a central role. However, the way in which it is formulated depends on the particular situation being considered.

The following picture, taken from Haykim's book, pg 494, illustrates the above four cases:


Maximize information conveyed to **Y** about **X**
Input **X**
Neural model
Output **Y**
(a)
Minimize statistical dependence between the Y's
(d)
$\mathbf{X}_a$
$\mathbf{X}_b$
Maximize information conveyed to Y_b about Y_a and vice versa
(b)
Minimize information conveyed to Y_b about Y_a and vice versa
(c)

Let's firstly try to contemplate how Mutual Information can be observed in a NN. Consider a Neural System where the application of a continuous random variable X to the input of the system produces a continuous random variable Y at the output of the system. By definition, the differential entropy h(X) is the uncertainty about the system input X before observation of the system output Y,and the conditional differential entropy H(X|Y) is the uncertainty about the system input X after observation of the system output Y. Therefore, the difference, H(X)-H(X|Y), is the uncertainty about the system input X that is resolved by observing the system output Y. This entropic difference is called the mutual information between the system input X and the system output Y; denoting it by I(X;Y).

The idea of designing a neural processor to maximize the mutual information I(Y; X) is appealing as the basis for statistical signal processing. This method of optimization is embodied in the **maximum mutual information (Infomax)** principle due to Linsker (Linsker, 1988), which may be stated formally as follows:

The transformation of a random vector X observed in the input layer of a neural system to a random vector Y produced in the output layer of the system should be so chosen that the activities of the neurons in the output layer jointly maximize *information about the activities in the input layer. The objective function to be maximized is the mutual information I(Y;X) between the vectors X and Y.*

The Infomax principle provides a mathematical framework for self-organization of the signal transmission system, assuming that the number of components in the output vector Y is smaller than the number of components in the input vector x. Also, this principle may be viewed as the neural network counterpart of the concept of channel capacity, which defines the Shannon limit on the rate of information transmission through a communication channel.(S. S. Haykin, 2009)

Classification in MLP[2] by minimizing Error's Entropy

Entropy and the related concepts of mutual information and Kulback-Leibler divergence have been used in learning systems (supervised or unsupervised) in several ways. Recently, Prıncipe and co-workers, proposed new approaches to using entropic criteria in particular, the minimization of the Renyi's second order entropy of the difference between the MLP output and the desired target (the error). The minimization of error entropy implies a reduction of the expected information contained in the error, which leads to a maximization of the mutual information between the desired target and the model output. This means that the network is learning the target variable. The benefits of using Shannon's entropy arise from the well-known information meaning as well as its unique information measure properties. The following algorithm is developed from Silva, Marques de Sa and Alexandre (Silva, S, & Alexandre, 2004)

Consider an MLP with one hidden layer with output y and a target variable (class membership for each example in the dataset), t. For each observation we measure the error using e(n) = t(n) − y(n), n = 1, . . . ,N where N is the total number of examples. We only consider the two-class problem; thus, as in we set t ∈ {−1, 1} and a single output unit with y ∈ [−1, 1]. The proposed **back propagation** algorithm uses a **Shannon's entropy estimator** with mean square consistency:

$$\hat{H}(E) = -\frac{1}{N}\sum_{n=1}^{N} \log \hat{f}(e(n))$$

where E is the error (difference) random variable. Note that this expression can be seen as the expected value of log f(x), thus the approximation of the integral by the mean value over a sample. Also, as we don't know the distribution of the error variable, we must rely on nonparametric estimates. For the estimation of f(x) we use the **nonparametric kernel estimator:**

[2] MultiLayer Perceptron

$$\hat{f}(e(n)) = \frac{1}{Nh}\sum_{l=1}^{N} K\left(\frac{e(n)-e(l)}{h}\right)$$

where h is the **smoothing parameter** of the standard Gaussian kernel K :

$$K(x) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{2}{2}x^2\right)$$

In order to use the steepest descent training rule and the backpropagation algorithm, we need to derive an analytic expression for the gradient. Using the usual notation where $\frac{\partial \hat{H}}{\partial w_{kj}}$ is the partial derivative of $\hat{H}$ related to the weight connecting neuron j in a previous layer to neuron k in the next layer, we have

$$\frac{\partial \hat{H}}{\partial w_{kj}} = -\frac{1}{N}\sum_{n}\frac{\partial}{\partial w_{kj}}\log(\hat{f}(e(n))) = -\frac{1}{N}\sum_{n}\frac{1}{\hat{f}(e(n))}\frac{\partial \hat{f}(e(n))}{\partial w_{kj}}$$, where

$$\frac{\partial \hat{f}(e(n))}{\partial w_{kj}} = \sum_{l=1}^{N}\frac{1}{\sqrt{2\pi}}\frac{\partial}{\partial w_{ij}}\exp\left\{-\frac{1}{2}\left(\frac{e(n)-e(l)}{h}\right)^2\right\} =$$

$$= \sum_{l=1}^{N}\frac{1}{h^2}K\left(\frac{e(n)-e(l)}{h}\right)\{e(n)-e(l)\}\left[\frac{\partial e(n)}{\partial w_{kj}} - \frac{\partial e(l)}{\partial w_{kj}}\right].$$

Thus, $$\frac{\partial \hat{H}}{\partial w_{kj}} = \frac{1}{N^2h^2}\sum_{n=1}^{N}\sum_{l=1}^{N}\frac{\frac{1}{h}K}{\hat{f}(e(n))}\left(\frac{e(n)-e(l)}{h}\right)\{e(n)-e(l)\}\left[\frac{\partial e(n)}{\partial w_{kj}} - \frac{\partial e(l)}{\partial w_{kj}}\right] \quad (1)$$

The computation of $\frac{\partial e(n)}{\partial w_{kj}}$ is as usual for the back propagation algorithm.

On ∂w_{kj} just have to take care if w_{kj} is an input-hidden or a hidden-output neuron. Having determined (1) for all network weights, the **weight update** is given, for the m-th iteration, by the **gradient descent rule**:

$$w_{kj}^{(m)} = w_{kj}^{(m-1)} - \eta\frac{\partial \hat{\mathrm{H}}_{\mathrm{E}}}{\partial w_{kj}}$$

The algorithm has two parameters that one should optimally set: the smoothing parameter, h, of the kernel density estimator and the learning rate η. As the training process evolves, it is expected that the errors get closer, which means that one should need a decreasing smoothing parameter h. Experimental results led to the conclusion that this is a highly sensitive parameter. Indeed, decreasing h as the training evolves (for example, proportional to the variance of the errors in each epoch), leads to an unstable behavior of the algorithm. In order to cope with this instability, one should run experiments with several h values and determine the best for each dataset and network configuration.

We also investigated the benefits of adjusting η along the training process. The procedure of variable learning rate not only solves the problem of choosing η, but also ensures a stable training. An **update rule for learning rate**:

$$\eta^m = u\eta^{(m-1)}, H^{(m)} \le H^{(m-1)}, 0 \overset{\wedge}{\le} d \le 1 \le u \text{ Or}$$

$$\eta^m = d\eta^{(m-1)} \text{, restart otherwise.}$$

If entropy does not increase from one epoch to another, the algorithm is in the right direction, so η is increased by a factor u in order to speedup convergence. However, if η is large enough to increase entropy, then the algorithm makes a restart step and decreases η by a factor d to ensure that entropy is being minimized. This restart step is just a return to the weights of the previous epoch.

At this point we should note that when back probagation training algorithm is implied, Shannon's entropy causes a computational complexity to the algorithm, due to the numerical evaluation of a complicated integral over the real line required. This problem can be solved by using Renyi's entropy for α=2 instead, with no loss in the efficiency of the algorithm when it comes to prediction or clustering problems. Erdogmus and Principe(Erdogmus & Principe, n.d.) developed the algorithm and applied it TDNN[3] for short-time series prediction.

Classification in MLP by Cross-Entropy Multinomial Logistic Regression

The logistic regression model used for multi-classification kind of problems, called the **multinomial logistic regression classifier (synonyms: Softmax Regrassion, Maximum Entropy Classifier or just Multi-class Logistic Regrassion),** employs **Softmax** Function , first introduced by Bridle (Bridle, n.d.) in the early 90's, and has since found a widly applied in Artificial Neural Networks, as an activation function of MLP's last layer. The algorithm presented is based on the work of Sebastian Raschka.

Let's first draw the workflow of this method, and then follow a step-by-step presentation:

[3] Time-Delay Neural Network

1) Insert Inputs=Training Data
2) Use the linear equation as Linear model
3) Logits
4) Apply Softmax Function
5) One-Hot-Encoding Target Representation
6) Utilize Cross Entropy as Cost Function
7) Parameter Optimization- Weight Update

Insert Inputs

The inputs are the features of each labeled observation of the training set X. Therefore, a single observation is just the single row values from the training set, containing all the features and the corresponding target class. It should be noted that features values should always be numerical. If not numerical, they need to be converted into numerical values using the proper categorical data analysis techniques.

Linear Model

The linear model equation is defined as the **linear equation in the linear regression model**, that is: Y=WX+b, where X is the set of inputs, b is the bias and W is the Weight Matrix, as defined above. One can easily understand after all the above familiarization with Neural Networks' structure, that this is the **Objective function** of the NN.

To make the procedure as far clear to the reader, let's give an example: Suppose that X is a matrix, containing all the features of each observation and the 1 for the bias, let's say X = [x1,x2,x3]. Then W should be another matrix including the same input number of weights W = [w1,w2,w3].Therefore, the linear model output will be the w1*x1, w2*x2, w3*x3 , and **NN learning process in nothing but optimize these weights**. For each observation in the training set will pass through the all the stages of multinomial logistic regression and the proper weights W will be computed.

Logits

Mutlinomial Logistic Regression's logits i.e. outputs of the linear model are formally named as Scores. They form the set Y, which is in 1-1 correspondence with X set, that is each observation of the training set is assigned to an output. Logits will change with the changes in the calculated weights.

Apply Softmax Function

Softmax function is the function that characterizes the logistic regression model for multi-classification(Bridle, n.d.). It is a probabilistic function that calculates the probabilities of the event over ‘N’ different events. That is, the N different classes we have. In general way of saying, this function will calculate the probabilities of each class over all possible classes. These calculated probabilities will be utilized for determining the target class for the given inputs. When used for multi-classification model, it returns the probabilities of each class and the target class will have the high probability. Concerning the creation of a NN, softmax functions used in different layer level.(Theodoridis & Koutroumbas, 2009)

As stated above, the softmax function returns the **high probability value for the high scores** and fewer probabilities for the remaining scores. That simply means that for each of NN’s logits, softmax will calculate a probability, indicating the target that suits best for each observation.

$$S(Y_j) = \frac{e^{y_j}}{\sum_{i=1}^{K} e^{y_i}}$$: Softmax Function

Where, Yj is the output assigned to Xj input, among a set of K inputs.

Properties of softmax function:
The calculated probabilities will be in ranger of 0 to 1.
The sum of all the probabilities is equals to 1.

One-Hot-Encoding:

One-Hot Encoding is a method to represent the target values into a binary representation. The creation of One-Hot-Encoding matrix is rather simple: For every input features the one-hot-encoding matrix is filled with the values of 0 and the 1 for each target class. The total number of values in the one-hot-encoding matrix and the unique target classes are the same.

An example will demonstrate the procedure clearly; supposing an observation X with 3 input features like x1, x2, and x3 and one target variable (With 3 target classes). Then the one-hot-encoding matrix will have 3 values. Out of 3 values, one value will be 1 and all other will be 0s, according to the label of the train data, pointing the class in belongs.

Utilize Cross Entropy as Cost Function

In general, defining a cost function, the input parameters for it are the calculated weights and all the training observations i.e. the logits. The cross entropy function is used to find the similarity distance between the probabilities calculated from softmax function and the target one-hot-encoding matrix.

If the loss function value is low, it indicates that there is not a big difference between NN output and target, therefore the Classification task has been optimally performed. Otherwise, the process of calculating the weights will start again and the weights will be updated, as described below. Process will continue until the cost function value is less than a threshold or after a number of iterations, set as the **termination criterion**.

At this point, IT makes its contribution to the classification task of our NN. The Cross-entropy is chosen as the cost function, forming a distance function between the calculated probabilities from softmax function and the created one-hot-encoding matrix. For the right target class, the distance value will be less, and the distance values will be larger for the wrong target class. Therefore, **Cross-Entropy Cost Function** is defined as:

$$D(S(Y);L) = -\sum_{x} p_{S(Y)}(x)\log_2 p_L(x)$$, where L is the One-Hot-Encoding Matrix.

Note: Cross-entropy is a measure but not a metric, as it fails the symmetric property.

Parameters Optimization-Weight Update

The process followed for the weight update of this algorithm is once again **Back Propagation rule**, implementing **gradient descent**, only this time the function to be minimized is the cost function above. The process will not be repeated again analytically as it is only a matter of different calculation of the derivatives, however basics will be mentioned.

1) The cost function to be minimized is defined as the average of all cross-entropies over our n training samples, that is:

$$J(W) = \frac{1}{K}\sum_{i=0}^{K} D(S(Y);L)$$

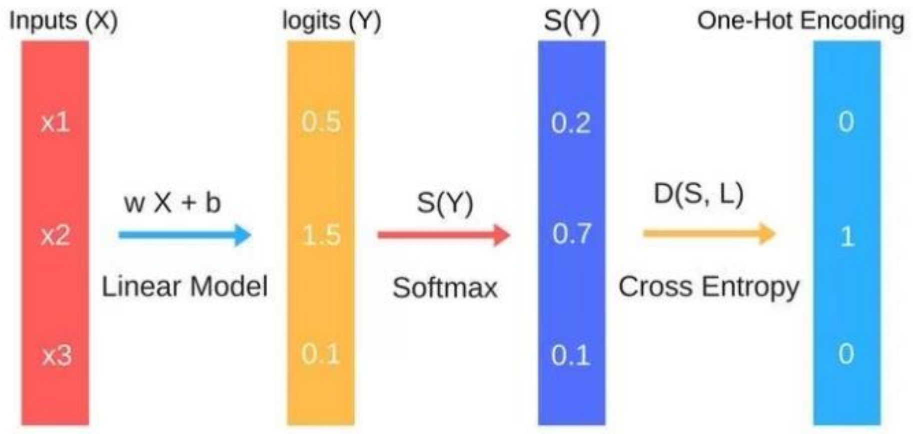
2) In order to learn our softmax model via **gradient descent**, we need to compute the derivative:

$$\frac{\partial \mathrm{J(W)}}{\partial \mathrm{w_i}} = -\frac{1}{\mathrm{K}}\sum_{\mathrm{i=1}}^{\mathrm{K}}[\mathrm{x}^{(\mathrm{i})}(\mathrm{L_i} - \mathrm{S(Y_i)})]$$

3) The **weight update rule** of m-th iteration is formed as:

$$w_{kj}^{(m)} = w_{kj}^{(m-1)} - \eta\frac{\partial J(W)}{\partial w_{kj}}$$, where η is the learning rate parameter.

All the procedure described above is simply presented in the following picture:



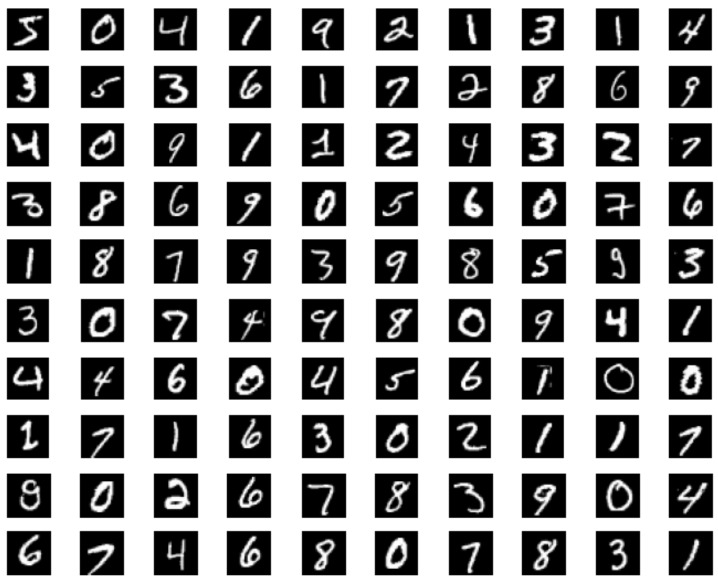
Multinomial Logistic Classifier

Application in a Neural Network- Multinomial Logistic Classifier

Introduction to the dataset and specify the problem

The MNIST Handwritten digit recognition is one of the fundamental problems in practical pattern recognition system design. Handwritten digit recognition is fundamental to many practical applications, such as bank check processing, postal mail sorting, automatic number plate recognition, etc. For these applications, the recognition accuracy and speed is crucial to the overall performance. The MNIST (Modified National Institute of Standards and Technology) data set [http://yann.lecun.com/exdb/mnist/index.html] was constructed by Yann Le- Cun from the NIST's special databases, which is a very representative data set for testing the classification performance on handwritten digit recognition. This data set contains 60000 training samples and 10000 testing samples. Each sample represents a digit ranging from 0 to 9, and is normalized and centered in a gray-scale image with size 28x28. It is used directly the 28x28 = 784 gray-scale pixel values as the features of each sample.

An example of the dataset's 100 first pictures is given in the picture below:



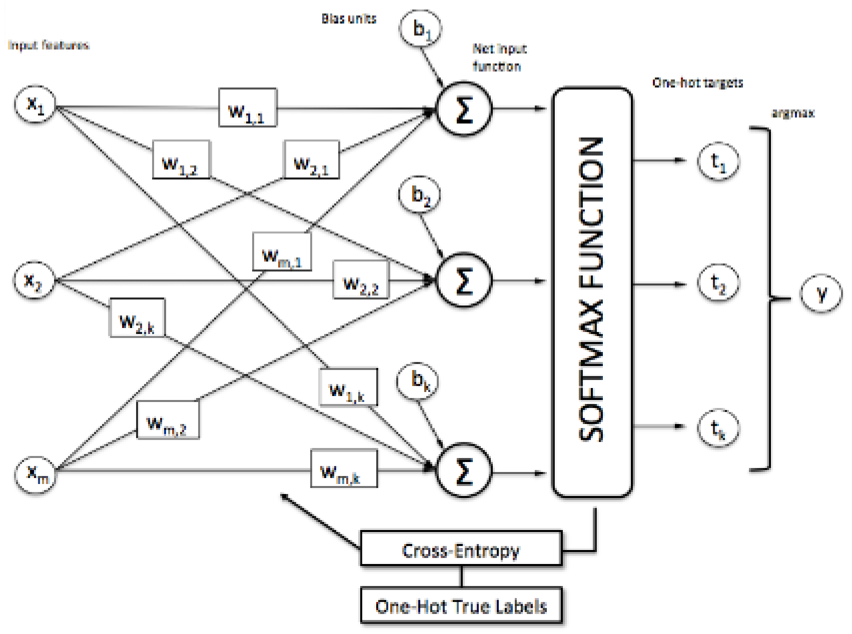
The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. The images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field. Therefore, that means that each picture, that is each observation in the dataset, forms a feature vector of **28x28=784 features**.

The MNIST training set is composed of 60,000 patterns. Our test set was composed of 10,000 patterns. The 60,000 pattern training set contained examples from approximately 250 writers. It has been made sure that the sets of writers of the training set and test set were disjoint.

The task to be accomplished is the classification of each image in one of the 10 classes of digit numbers 0-9, so that it will be assigned to the number class it belongs. That basically means creating a NN that can identify the hand writing number. Two distinct NN were developed for this purpose, an Artificial Neural Network with 3 layers, an input, an output and a hidden one, that uses an Entropy based criteria as a Cost Function, specifically the Multinomial Logistic Regression algorithm presented above, and an 3-layered MLP, utilizing Mean Square Error as its cost function. Then, the results are presented, compared and analyzed.

Methodology and Strategy

Softmax Regression (synonyms: Multinomial Logistic, Maximum Entropy Classifier, or just Multi-class Logistic Regression)is chosen to complete the task of MNIST's classification, as it is a task of multi-class classification where the class label *y* can take on more than two possible values ,under the assumption that the classes are mutually exclusive, as it happens in our case. The following pictures sums up the entire procedure:



Softmax Regression

To briefly describe how the above process was implemented in our case, we should first mention that each image among the 60.000 is inserted one after the others and there are 784+ the bias=785 Xi inputs features. We have 10 Σ, that is 10 clases of digits, therefore 10 synaptic weights are assigned to each Xi. Each one of the 10 classes' neurons is given a value that is next to be inserted in the Softmax function, in order to become a probability. The value of the softmax function is therefore the the inserted observation's probability to belong to each class. After the creation of one-hold-encoding matrix L has been completed, cost- entropy cost function is being used and finding the minimum of this function, the weights are optimized the output of our NN is resulted.

The implementation is based on professor's Andrew Ng suggestions that can be found in this site: ufldl.stanford.edu. Matlab has been used for the creation and training of the NN. The specific procedure is given in the rest of this section.

Load data

To load the data, two MATLAB helper functions have been used, for extracting the data from http://yann.lecun.com/exdb/mnist/index.html. The function loads the MNIST images and labels into inputData and labels respectively. The images are **pre-processed to scale the pixel values (gray-scaled) to the range [0,1].**

Initialize constants and parameters

Two constants, **inputSize** and **numClasses**, corresponding to the size of each input vector and the number of class labels, also initialize a, the weight decay parameter i.e. the learning rate.

Linear Regression Model

Inserting each observation in the linear model Y=WX results to the 10 logits, forming the Y vector.

Softmax Function

Softmax function is then utilized to transform each one of the 10 Yi logits into probabilities. The logit with the highest probability stands for the class that the observation is classified to.

Create One-Hot-Encoding Label Matrix

The set observation's labels forms a 60.000x1 matrix that we will now convert in a 60.000x10 matrix so that the multi-class classification can be performed. This happens as stated: first of all, for convenience reasons the class label 0 is renamed as 10. Then, for each one observation there is a row of 10 columns, each column standing for one of the 10 classes, respectively from 1 to 10. We put 1 in the column of the class the observation belongs and fill with zeros in all the other indexes of the row.

Implement the Cost Function

In softmaxCost.m, implement code to compute the cost function *J*(θ). Then train the softmax model using the function softmaxTrain.m. which uses the L-BFGS algorithm, which is nothing but the gradient descent specified for this problem, utilizing the function minFunc. Training the model on the entire MNIST training set of 60000 28x28 images. The learning parameter that was used when softmax classifier was trained is set to a = 0.0001

Testing

Now that we've trained the model, it will be tested against the MNIST test set, comprising 10000 28x28 images. To do so we used the function softmaxPredict.m, a function which generates the predictions for input data under a trained softmax model? Finally, to **evaluate the accuracy** has been used (the proportion of correctly classified images).

Comparison to Multilayer Perceptron (MLP)

In order to properly evaluate the implemented algorithms efficiency and investigate if the entropic function is making a difference in the procedure, an MLP NN was also trained to accomplish the same task. The specific details of MLP's implementation are not relevant to this essay and shall not be presented, but a brief description of MLP is needed.

To compare the classification of the handwritten digits recognition problem with Softmax Regression we used a fully connected neural network that consists of 3 layers: the input, a hidden layer and the output layer. Specifically, the number of neurons that has been used in hidden layer is 100 and 300. In the output layer we have 10 output units and each output unit represents a class label, we 10 classes from digit 0 to 9.

As activation function it is used the logistic (sigmoid) function: $f(x) = \frac{1}{1+e^{-x}}$.

The neural network learns through back propagation, a generalization of the **least mean squares error** algorithm in linear perceptrion. It is trying to minimize the cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 ,$$

where I is the i-th training example, θ the weights and h_θ is the hypothesis i.e. the logistic function (the net input). After gradient descent was applied to the cost function we get

$\theta_i \coloneqq \theta_i - a\sum_{i=1}^{m} x_j^{(i)}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$ to update the weights θ,

with learning rate α = 0.01. Finally, the number of the epochs-iterations that has been used is 10 and 100.

Algorithm and Parameters

Multinomial Logistic Regression algorithm is the exact one presented above, implemented as explained in the previous "Methodology and Strategy "section. Algorithm's specific parameters that had to be adjusted are the followings:

Learning Rate a: It was set to 0.0001, as after experimental trials it was proved to be neither too small to cause an overtrain and extremely slow convergence of the algorithm, nor too high and thus risk the overpass on function's minimum.

Iterations' Époques: 170 NN iterations époques were needed for the NN to be properly trained, meaning it was neither overtrained nor unprepared to accomplish the classification task in an optimal way.

Number of NN's Hidden Layers: No hidden layer needed as it was proved that the input and outup layers were enough to accomplish the optimal classification, as cross-entropy is an extremely strong measure and more than a hidden layer in the NN would cause an overfitting of the model.

For the MLP with MSE, the chosen parameters are set exactly the same, as the two algorithms share almost the same method, besides the different cost function, and therefore dataset's needs remained similar.

Finally, in order evaluate the two NN Classifiers we used the same evaluation criterion, and both NN were implemented in the same computer, so that computational needs could be compared and machine errors shall not interfere in results' comparison.

Evaluation Measure: Accuracy: the proportion of correctly classified images. It is a combination of both types of observational error (random and systematic), so high accuracy requires both high precision and high trueness.

Results and Remarks

The Softmax regression classifier performance was trained for several parameters and the accuracy and executive time are presented in the table below.

Learning Rate	Executive Time	Iterations	Accuracy
0.01	33 sec	57	90.56%
0.001	70 sec	126	91.97%
0.0001	167 sec	316	92.63 %

The first remark to be made is concerning the learning rate. As the learning rate is increased, the algorithms termination is faster, that means that it faster reaches the cost function's minimum, but the Accuracy rate is decreased, as the NN is not properly trained, trying to learn so fast. As it's stated, the resulting accuracy rate is very satisfying. The most remarkable advantage of this algorithm though, is not the sufficient accuracy but the fast training of the algorithm and the extremely low time consumption.

MLP's Results:

Hidden Layer Neurons	Executive Time	Iterations	Accuracy
100	5 min	10	94.15%
100	40 min	100	96.88%
300	11 min	10	94.97%
300	90 min	100	97.94%

Comparing the result tables:

The conclusions to be made are: Although MLP gains in Accuracy rates, the accuracy difference between the two Neural networks is disregarded due to the huge difference in time consumption. As it is obvious MLP requires a great number of neurons in hidden layer in order to achieve the classification, leading to enormous time request. The conclusion to be made therefore is that it depends to each problems needs and researchers preferences which algorithm to be chosen. If accuracy is the key to the classification problem, then MLP with MSE cost function should be preferred over Softmax algorithm. However, if it is a practical problem that needs a fast implementation in real life, the Softmax algorithm has a great advantage.

Future Work

Although we feel that this essay achieved its goal to present a complete frame line through how Information Theory can become an extremely useful tool of Machine Learning, we firmly believe that there is a lot more to be discovered and discussed and a numerous interesting algorithms still to be discovered.

Moreover, it should be mention that our first implementation was chosen to be a Clustering Problem of MNIST, using as a distance measure the Kulback- Leibler divergence. However, this implementation was not accomplished due to the fact that there was not a remarkable difference among the distributions of the clusters of NMIST, and the entropy criteria failed. Find a way to implement an entropic criterion that overcomes this problem is among our future goals.

Furthermore, it is of great interest of us to implement in natural problems all the presented algorithms and compare their results with traditional ML algorithms, in an effort to realize whether or not there are specific cases that IT criteria are more efficient and appropriate, and if so understand these particularities.

Finally, we believe that a systematic approach of the usage of IT in ML is yet to be accomplished, and presents potentials for very interesting applications and deeper understanding for both ML as well as IT.

Appendix-Matlab Code

The bellow functions are from the assignments in the site deeplearning.stanford.edu :

```
%%Cost function
function [cost, grad] = softmaxCost(theta, numClasses, inputSize, lambda, data, labels)
 % numClasses - the number of classes
% inputSize - the size N of the input vector
% lambda - learning rate/weight decay parameter
% data - the N x M input matrix, where each column data(:, i) corresponds to a single test set
% labels - an M x 1 matrix containing the labels corresponding for the input data
%
theta = reshape(theta, numClasses, inputSize);
numCases = size(data, 2);
groundTruth = full(sparse(labels, 1:numCases, 1));
cost = 0;
thetagrad = zeros(numClasses, inputSize);
y = groundTruth;
m = numCases;
% note that if we subtract off after taking the exponent, as in the text, we get NaN
td = theta * data;
td = bsxfun(@minus, td, max(td));
temp = exp(td);
denominator = sum(temp);
p = bsxfun(@rdivide, temp, denominator);
cost = (-1/m) * sum(sum(y .* log(p))) + (lambda / 2) * sum(sum(theta .^2));
thetagrad = (-1/m) * (y - p) * data' + lambda * theta;
% Unroll the gradient matrices into a vector for minFunc
grad = [thetagrad(:)];
end

%%%================================================================
%%%================================================================


%% Compute the Gradient descent
function numgrad = computeNumericalGradient(J, theta)
% numgrad = computeNumericalGradient(J, theta)
% theta: a vector of parameters
% J: a function that outputs a real-number. Calling y = J(theta) will return the
% function value at theta.

% Initialize numgrad with zeros
numgrad = zeros(size(theta));

eps = 1e-4;
n = size(numgrad);
I = eye(n, n);
for i = 1:size(numgrad)
    eps_vec = I(:,i) * eps;
    numgrad(i) = (J(theta + eps_vec) - J(theta - eps_vec)) / (2 * eps);
end
end
%%%================================================================
%%%================================================================

%% Compute the Gradient descent
```

```
function [softmaxModel] = softmaxTrain(inputSize, numClasses, lambda, inputData, labels,
options)
%softmaxTrain Train a softmax model with the given parameters on the given
% data. Returns softmaxOptTheta, a vector containing the trained parameters
% for the model.
%
% inputSize: the size of an input vector x^(i)
% numClasses: the number of classes
% lambda: weight decay parameter
% inputData: an N by M matrix containing the input data, such that
%           inputData(:, c) is the cth input
% labels: M by 1 matrix containing the class labels for the
%           corresponding inputs. labels(c) is the class label for the cth input
% options (optional): options
%   options.maxIter: number of iterations to train for

if ~exist('options', 'var')
    options = struct;
end

if ~isfield(options, 'maxIter')
    options.MaxIter = 400;
end

% initialize parameters
theta = 0.005 * randn(numClasses * inputSize, 1);

% Use minFunc to minimize the function
addpath ../common/fminlbfgs
options.Method = 'lbfgs';
% Here, we use L-BFGS to optimize our cost
% function. Generally, for minFunc to work, you
% need a function pointer with two outputs: the
 % function value and the gradient. In our problem,
 % softmaxCost.m satisfies this
options.Display = 'iter';
options.GradObj = 'on';

[softmaxOptTheta, cost] = fminlbfgs( @(p) softmaxCost(p, numClasses, inputSize, lambda,
inputData, labels), theta, options);

% Fold softmaxOptTheta into a nicer format
softmaxModel.optTheta = reshape(softmaxOptTheta, numClasses, inputSize);
softmaxModel.inputSize = inputSize;
softmaxModel.numClasses = numClasses;
end

%%%=============================================================
%%%=============================================================

%%  Prediction of the model
function [pred] = softmaxPredict(softmaxModel, data)

% data - the N x M input matrix, where each column data(:, i) corresponds to a single test set
% produce the prediction matrix
% pred, where pred(i) is argmax_c P(y(c) | x(i)).

% Unroll the parameters from theta
theta = softmaxModel.optTheta;  % this provides a numClasses x inputSize matrix
pred = zeros(1, size(data, 2));
```

```
size(pred); %   1 10000
size(data); % 784 10000
size(theta);%  10     8

p = theta*data;
[junk, idx] = max(p, [], 1);

pred = idx;

end



%%%==========================================================
%%%==========================================================

%%  Softmax Regression

%% Initialise constants and parameters
inputSize = 28 * 28; % Size of input vector (MNIST images are 28x28)
numClasses = 10;     % Number of classes (MNIST images fall into 10 classes)
lambda =  1e-4; % Weight decay parameter - learning rate

%%  Load data
images = loadMNISTImages('train-images-idx3-ubyte');
labels = loadMNISTLabels('train-labels-idx1-ubyte');
labels(labels==0) = 10; % Remap 0 to 10

inputData = images;

% For debugging purposes,  the size of the input data has been reduced in order to speed up
gradient checking.
DEBUG = false; % Set DEBUG to true when debugging.
if DEBUG
   inputSize = 8;
   inputData = randn(8, 100);
   labels = randi(10, 100, 1);
end
% Randomly initialise theta (weights)
theta = 0.005 * randn(numClasses * inputSize, 1);
%  Implement softmaxCost in softmaxCost.m.
[cost, grad] = softmaxCost(theta, numClasses, inputSize, lambda, inputData, labels);

%%  Gradient checking

if DEBUG
   numGrad = computeNumericalGradient( @(x) softmaxCost(x, numClasses, inputSize,
lambda, inputData, labels), theta);

   % Use this to visually compare the gradients side by side
   disp([numGrad grad]);

   % Compare numerically computed gradients with those computed analytically
   diff = norm(numGrad-grad)/norm(numGrad+grad);
   disp(diff);
end

%% Learning parameters
% training softmax regression code using softmaxTrain  (which uses minFunc).
```

```
options.MaxIter = 100;
softmaxModel = softmaxTrain(inputSize, numClasses, lambda, inputData, labels, options);

%%  Testing the test imagies
%    softmaxPredict.m which return predictions given a softmax model and the input data.

images = loadMNISTImages('t10k-images-idx3-ubyte');
labels = loadMNISTLabels('t10k-labels-idx1-ubyte');
labels(labels==0) = 10; % Remap 0 to 10

inputData = images;

% implement softmaxPredict in softmaxPredict.m
[pred] = softmaxPredict(softmaxModel, inputData);

% Accuracy is the proportion of correctly classified images
acc = mean(labels(:) == pred(:));
fprintf('Accuracy: %0.3f%%\n', acc * 100);
```

The following code was implemented as a part of the assignments in the course Neural Networks in Aristotle University of Thessaloniki:

```
% load training set and testing set
train_set = loadMNISTImages('train-images.idx3-ubyte')';
train_label = loadMNISTLabels('train-labels.idx1-ubyte');
test_set = loadMNISTImages('t10k-images.idx3-ubyte')';
test_label = loadMNISTLabels('t10k-labels.idx1-ubyte');

tic;   %start  timer
% parameter setting
alpha = 0.1; % learning rate
beta = 0.01; % scaling factor for sigmoid function
train_size = size(train_set);
N = train_size(1); % number of training samples
D = train_size(2); % dimension of feature vector, 784pixels
n_hidden = 300; % number of hidden layer units
K = 10; % number of output layer units

% initialize all weights between -1 and 1
W1 = 2*rand(1+D, n_hidden)-1;
W2 = 2*rand(1+n_hidden, K)-1;
epochs = 200;
Y = eye(K);   % output
test_size = size(test_set);

% training
for i=1:epochs
    disp([num2str(i), ' epoch']);  % display the number of the epoch
    permutation = randperm(N);  %randomize the order of j
    for j=1:N
        index = permutation(j);  %new index after permutation
        % propagate forward
        input_x = [1; train_set(index, :)'];
        hidden_output = [1;sigmf(W1'*input_x, [beta 0])];
        output = sigmf(W2'*hidden_output, [beta 0]);
        % back propagation
        % compute the error of output unit c
        outlayer_delta = (output-Y(:,train_label(index)+1)).*output.*(1-output);
        % compute the error of hidden unit h
        hiddenlayer_delta = (W2*outlayer_delta).*(hidden_output).*(1-hidden_output);
        hiddenlayer_delta = hiddenlayer_delta(2:end);
        % update the weights
        W1 = W1 - alpha*(input_x*hiddenlayer_delta');
        W2 = W2 - alpha*(hidden_output*outlayer_delta');
    end

    toc; %end  timer
    %train testing
    success = 0;
    for j=1:test_size(1)
    input_x = [1; test_set(j,:)'];
    hidden_output = [1; sigmf(W1'*input_x, [beta 0])];
    output = sigmf(W2'*hidden_output, [beta 0]);
    [max_unit,max_idx] = max(output);
         if(max_idx == test_label(j)+1)
             success =success + 1;
         end
    end
    perc_succ = success/test_size(1);
    perc1=100*perc_succ;
    disp([num2str(perc1), '% '])
end

logit=zeros(test_size(1),10);
% testing
num_correct = 0;
for i=1:test_size(1)
    input_x = [1; test_set(i,:)'];
    hidden_output = [1; sigmf(W1'*input_x, [beta 0])];
    output = sigmf(W2'*hidden_output, [beta 0]);
    logit(i,:) = softmax(output');
    [max_unit,max_idx] = max(output);
    if(max_idx == test_label(i)+1)
        num_correct = num_correct + 1;
    end
end
% computing accuracy
accuracy = num_correct/test_size(1);
perc=100*accuracy;
disp(['Total accuracy ', num2str(perc), '%'])
```

Bibliography[4]

Becker, S. (1991). Unsupervised Learning Procedures for Neural Networks. *Journal of Neural Systems*, *2*(1 & 2), 17–33. http://doi.org/10.1017/CBO9781107415324.004

Berger, A. L., Pietra, V. J. D., & Pietra, S. a. D. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, *22*(1), 39–71. http://doi.org/10.3115/1075812.1075844

Bridle, J. (n.d.). Training Stochastic Model Recognition Algorithms as Networks can lead to Maximum Mutual Information Estimation of Parameters, (Ml), 211–217.

Cover, T. M., & Thomas, J. A. (2005). *Elements of Information Theory*. *Elements of Information Theory*. http://doi.org/10.1002/047174882X

Csiszár, I. (1996). Maxent, Mathematics, and Information Theory. In K. M. Hanson & R. N. Silver (Eds.), *Maximum Entropy and Bayesian Methods: Santa Fe, New Mexico, U.S.A., 1995 Proceedings of the Fifteenth International Workshop on Maximum Entropy and Bayesian Methods* (pp. 35–50). Dordrecht: Springer Netherlands. http://doi.org/10.1007/978-94-011-5430-7_5

Erdogmus, D., & Principe, J. C. (n.d.). in Adaptive System Training Using Higher Order Statistics. *Computer Engineering*, 75–80.

Haykin, S. (2008). *Neural Networks and Learning Machines*. *Pearson Prentice Hall New Jersey USA 936 pLinks* (Vol. 3). http://doi.org/978-0131471399

Haykin, S. S. (2009). *Neural networks and learning machines, 3rd Edition*. http://doi.org/10987654321

Hu, B. (n.d.). Information Theory and its Relation to Machine Learning.

Hu, B.-G., He, R., & Yuan, X. (2011). Information-Theoretic Measures for Objective Evaluation of Classifications. *Acta Automatica Sinica*, *38*(7), 25. http://doi.org/10.1016/S1874-1029(11)60289-9

Jaynes, E. T. (1957). Information Theory and Statistical Mechanics. *The Physical Review*. http://doi.org/10.1103/PhysRev.106.620

Jenssen, R., Ii, K. E. H., Erdogmus, D., Principe, J. C., & Eltoft, T. (n.d.). Clustering using Renyi ' s Entropy.

Kroese, D. P., Rubinstein, R. Y., & Taimre, T. (2007). Application of the cross-entropy method to clustering and vector quantization. *Journal of Global Optimization*, *37*(1), 137–157. http://doi.org/10.1007/s10898-006-9041-0

[4] Bibliography was formed using Mendeley reference manager.

Li, H., Zhang, K., & Jiang, T. (2004). Minimum entropy clustering and applications to gene expression analysis. *Proceedings / IEEE Computational Systems Bioinformatics Conference, CSB. IEEE Computational Systems Bioinformatics Conference*, 142–151. http://doi.org/10.1109/CSB.2004.1332427

Li, T., & Ogihara, M. (2004). Entropy-Based Criterion in Categorical Clustering.

Linsker, R. (1988). Self-Organization in a Perceptual Network. *Computer*, *21*(3), 105–117. http://doi.org/10.1109/2.36

Małyszko, D., & Stepaniuk, J. (2008). Standard and Fuzzy Rough Entropy Clustering Algorithms in Image Segmentation. In C.-C. Chan, J. W. Grzymala-Busse, & W. P. Ziarko (Eds.), *Rough Sets and Current Trends in Computing: 6th International Conference, RSCTC 2008 Akron, OH, USA, October 23-25, 2008 Proceedings* (pp. 409–418). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-88425-5_42

Mitchell, T. M. (1997). *Machine Learning*. *Annual Review Of Computer Science* (Vol. 4). http://doi.org/10.1145/242224.242229

Nigam, K., Lafferty, J., & Mccallum, A. (1999). Using Maximum Entropy for Text Classification. *IJCAI-99 Workshop on Machine Learning for Information Filtering*, 61–67. http://doi.org/10.1.1.63.2111

Palubinskas, G., Descombes, X., & Kruggel, F. (1998). An Unsupervised Clustering Method using the Entropy Minimization. In *Proceedings. Fourteenth International Conference on Pattern Recognition* (Vol. 2, pp. 1816–1818).

Phillips, S. J., Anderson, R. P., & Schapire, R. E. (2006). Maximum entropy modeling of species geographic distributions. *Ecological Modelling*, *190*(3–4), 231–259. http://doi.org/10.1016/j.ecolmodel.2005.03.026

Pietra, S. D., Pietra, V. D., & Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(4), 380–393. http://doi.org/10.1109/34.588021

Principe, J. C. (2010). *Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives*. *Xtemp01*. http://doi.org/10.1007/978-1-4419-1570-2

Sahoo, P., Wilkins, C., & Yeager, J. (1997). Threshold selection using Renyi's entropy. *Pattern Recognition*, *30*(1), 71–84. http://doi.org/10.1016/S0031-3203(96)00065-9

Sergios Theodoridis, Konstantinos Koutroumbas Pattern recognition 2003. (n.d.).

Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, *27*(3), 379–423. http://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Silva, L. M., S, J. M. De, & Alexandre, L. A. (2004). Neural Network Classification using Shannon ' s Entropy.

Tabor, J., & Spurek, P. (n.d.). Cross-Entropy Clustering.

http://doi.org/10.1016/j.patcog.2014.03.006

Theodoridis, S., & Koutroumbas, K. (2009). *Pattern Recognition*. *Wiley Interdisciplinary Reviews Computational Statistics* (Vol. 5748). http://doi.org/10.1002/wics.99

Watanabe, S. (1981). Pattern recognition as a quest for minimum entropy. *Pattern Recognition*, *13*(5), 381–387. http://doi.org/10.1016/0031-3203(81)90094-7

Yeo, G., & Burge, C. B. (2004). Maximum Entropy Modeling of Short Sequence Motifs with Applications to RNA Splicing Signals. *Journal of Computational Biology*, *11*(2–3), 377–394. http://doi.org/10.1089/1066527041410418

Zellner, A. (1988). Optimal Information Processing and Bayes's Theorem. *The American Statistician*, *42*(4), 278–280. http://doi.org/10.2307/2685143

Zhu, S., Ji, X., Xu, W., & Gong, Y. (2005). Multi-labelled classification using maximum entropy method. *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '05*, 274. http://doi.org/10.1145/1076034.1076082

Zimmer, Y., Tepper, R., & Akselrod, S. (1996). A two-dimensional extension of minimum cross entropy thresholding for the segmentation of ultrasound images. *Ultrasound in Medicine and Biology*, *22*(9), 1183–1190. http://doi.org/10.1016/S0301-5629(96)00167-6

Yao YY (2003). Information-theoretic measures for knowledge discovery and data mining. In: Karmeshu (ed) Entropy Measures, Maximum Entropy Principle and Emerging Applications. Springer, Berlin. Pages 115-136

Jaynes,E.T., 1957.“Information theory and statistical mechanics,”Physical Review, vol. 106,pp.620– 630;“Information theory and statistical mechanic II,” Physical Review, vol. 108, pp. 171–190.

Diamantaras K. (2007).Artificial Neural Networks, Kleidariyhmos Publications Greece

Osborne, M. (2002). Using maximum entropy for sentence extraction. In Proceedings of the ACL'02 Workshop on Automatic Summarization, pages 1–8, Morristown, NJ, USA.