Machine Learning Basics
Part 2

Johar M. Ashfaque

I. ENTROPY

Entropy is a mathematical formula used to measure information gain. The normal distribution has the maximum entropy among all continuous distributions having a fixed mean and variance. What this implies is that the normal distribution incorporates the minimum amount of prior knowledge into a model.

II. HYPERPARAMETERS

In machine learning, we define hyperparameters as those which can not directly learn from data. These parameters are inherent to the model and are used to control the behaviour of the model. Some of the hyperparameters in a learning model are:

- The number of centroids in a clustering algorithm.
- The learning rate of an algorithm.
- The number of leaves or depth of a tree in tree-based algorithms.

III. CROSS-VALIDATION

For large data sets, the original sample of data may be partitioned into a training set on which to train the model, a validation set on which to validate our models and a test set to evaluate our trained model. However, when we do not have large samples of data, cross-validation is particularly useful. Cross-validation (CV) allows us to select a model and estimate the error in the model. When we select a model using CV, we do that by selecting one model from a range of other models, trained on a particular data set and/or by selecting the hyperparameters of a particular model.

The CV process involves partitioning the data set into k-folds consisting of k number of equal sized sub-samples of the original data set. From the k sub-samples, the first $k-1$ sub-samples are used to train the model and remaining one sub-sample is used to validate the model. The process is repeated k times and an average error is arrived at across all the k trials. The question arises as to how we choose the right value of k? TO answer this, we need to recollect that a lower value of k increases bias and a higher value of k increases variance. However, the rule of the thumb dictates using $k = 10$. CV allows us to approximate the model error as close as possible to the generalization error. CV also allows us to observe the variance in prediction. If the variance between the predicted values is high, it could tell us that the model is overfitting. After performing CV with different models and different hyperparameters, we choose the one model that has the best performance with respect to its error and its variance. We then need to rebuild the model from the training data set and evaluate on the test data set. If the model does not work well on the test data set, we need to reevaluate our list of models and the tuning parameters.

Leave one out cross-validation (LOOCV) uses a single observation from the original sample as the validation data and the remaining observations are used as the model training data. In essence, we are training the model with on observation less and validating the model on the left out observation. The k value here is the number of observations present in the data. LOOCV may be used when the size of the data is very small as otherwise it can be computationally expensive because the training process is repeated by the number of observations present in the data set.

IV. BAYESIAN STATISTICS

So far we have discussed frequentist statistics and approaches based on estimating a single value of $\boldsymbol{\theta}$, then making all predictions thereafter based on that one estimate. Another approach is to consider all possible values of $\boldsymbol{\theta}$ when making a prediction. The latter is the domain of Bayesian statistics. The frequentist perspective is that the true parameter value $\boldsymbol{\theta}$ is fixed but unknown, while the point estimate $\hat{\boldsymbol{\theta}}$ is a random variable on account of it being a function of the dataset (which is seen as random). The Bayesian perspective on statistics is quite different. The Bayesian uses probability to reflect degrees of certainty in states of knowledge. The dataset is directly observed and so is not random. On the other hand, the true parameter $\boldsymbol{\theta}$ is unknown or uncertain and thus is represented as a random variable.

Before observing the data, we represent our knowledge of $\boldsymbol{\theta}$ using the prior probability distribution, $p(\boldsymbol{\theta})$ (sometimes referred to as simply "the prior"). Generally, the machine learning practitioner selects a prior distribution that is quite broad i.e. with high entropy to reflect a high degree of uncertainty in the value of $\boldsymbol{\theta}$ before observing any data. For example, one might assume a priori that $\boldsymbol{\theta}$ lies in some nite range or volume, with a uniform distribution. Many priors instead reflect a preference for "simpler" solutions (such as smaller magnitude coefficients, or a function that is closer to being constant).

Now consider that we have a set of data samples $\{x^{(1)}, ..., x^{(m)}\}$. We can recover the eect of data on our belief about $\boldsymbol{\theta}$ by combining the data likelihood $p(x^{(1)}, ..., x^{(m)}|\boldsymbol{\theta})$ with the prior via Bayes' rule:

$$p(\boldsymbol{\theta}|x^{(1)}, ..., x^{(m)}) = \frac{p(x^{(1)}, ..., x^{(m)}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(x^{(1)}, ..., x^{(m)})}.$$

In the scenarios where Bayesian estimation is typically used, the prior begins as a relatively uniform or Gaussian distribution with high entropy, and the observation of the data usually causes the posterior to lose entropy and concentrate around a few highly likely values of the parameters.

The frequentist approach addresses the uncertainty in a given point estimate of $\boldsymbol{\theta}$ by evaluating its variance. The variance of the estimator is an assessment of how the estimate might change with alternative samplings of the observed data. The Bayesian answer to the question of how to deal with the uncertainty in the estimator is to simply integrate over it, which tends to protect well against overfitting. This integral is of course just an application of the laws of probability, making the Bayesian approach simple to justify, while the frequentist machinery for constructing an estimator is based on the rather ad hoc decision to summarize all knowledge contained in the dataset with a single point estimate. Bayesian methods typically generalize much better when limited training data is available but typically suffer from high computational cost when the number of training examples is large.

V. SUPERVISED LEARNING ALGORITHMS

Supervised learning algorithms are, roughly speaking, learning algorithms that learn to associate some input with some output, given a training set of examples of inputs $\mathbf{x}$ and outputs $\mathbf{x}$. In many cases the outputs $\mathbf{y}$ may be dicult to collect automatically and must be provided by a human "supervisor", but the term still applies even when the training set targets were collected automatically.

The k-nearest neighbors is a family of techniques that can be used for classification or regression. As a non-parametric learning algorithm, k-nearest neighbors is not restricted to a fixed number of parameters. We usually think of the k-nearest neighbors algorithm as not having any parameters but rather implementing a simple function of the training data. In fact, there is not even really a training stage or learning process. Instead, at test time, when we want to produce an output y for a new test input $\mathbf{x}$, we nd the k-nearest neighbors to $\mathbf{x}$ in the training data $\mathbf{X}$. We then return the average of the corresponding y values in the training set. This works for essentially any kind of supervised learning where we can define an average over y values. In the case of classification, we can average over one-hot code vectors $\mathbf{c}$ with $c_y = 1$ and $c_i = 0$ for all other values of i. We can then interpret the average over these one-hot codes as giving a probability distribution over classes. As a non-parametric learning algorithm, k-nearest neighbor can achieve very high capacity. One weakness of k-nearest neighbors is that it can not learn

that one feature is more discriminative than the another.

Another type of learning algorithm that also breaks the input space into regions and has separate parameters for each region is the decision tree and its many variants. Each node of the decision tree is associated with a region in the input space, and internal nodes break that region into one sub-region for each child of the node. Space is thus subdivided into non-overlapping regions, with a one-to-one correspondence between leaf nodes and input regions. Each leaf node usually maps every point in its input region to the same output. The learning algorithm can be considered non-parametric if it is allowed to learn a tree of arbitrary size, though decision trees are usually regularized with size constraints that turn them into parametric models in practice. Decision trees as they are typically used, with axis-aligned splits and constant outputs within each node, struggle to solve some problems that are easy even for logistic regression. The decision tree will thus need to approximate the decision boundary with many nodes, implementing a step function that constantly walks back and forth across the true decision function with axis-aligned steps.

As we have seen, nearest neighbour predictors and decision trees have many limitations. Nonetheless, they are useful learning algorithms when computational resources are constrained. We can also build intuition for more sophisticated learning algorithms by thinking about the similarities and differences between sophisticated algorithms and k-nearest neighbors or decision tree baselines.

A. Support Vector Machines (SVMs)

One of the most influential approaches to supervised learning is the support vector machine. One key innovation associated with support vector machines is the kernel trick. The kernel trick consists of observing that many machine learning algorithms can be written exclusively in terms of dot products between examples. In some infinite dimensional spaces, we need to use other kinds of inner products, for example, inner products based on integration rather than summation. The kernel trick is powerful for two reasons. First, it enables us to learn models that are nonlinear as a function of $\mathbf{x}$ using convex optimization techniques that are guaranteed to converge efficiently. This is possible because we consider ϕ fixed and optimize only $\boldsymbol{\alpha}$ (a vector of coefficients), that is, the optimization algorithm can view the decision function as being linear in a different space. Second, the kernel function k often admits an implementation that is significantly more computationally efficient than naively constructing two $\phi(\mathbf{x})$ vectors and explicitly taking their dot product.

Support vector machines are not the only algorithm that can be enhanced using the kernel trick. Many other linear models can be enhanced in this way. The category of algorithms that employ the kernel trick is known as kernel machines, or kernel methods. A major drawback to kernel machines is that the cost of evaluating the decision function is linear in the number of training examples, because the i-th example contributes a term $\alpha_i k(\mathbf{x}, \mathbf{x}^i)$ to the decision function. Support vector machines are able to mitigate this by learning an α vector that contains mostly zeros. Classifying a new example then requires evaluating the kernel function only for the training examples that have nonzero α_i. These training examples are known as support vectors. Kernel machines also suffer from a high computational cost of training when the dataset is large. Kernel machines with generic kernels struggle to generalize well.

VI. UNSUPERVISED LEARNING ALGORITHMS

Unsupervised algorithms are those that experience only "features" but not a supervision signal. The distinction between supervised and unsupervised algorithms is not formally and rigidly defined because there is no objective test for distinguishing whether a value is a feature or a target provided by a supervisor. Informally, unsupervised learning refers to most attempts to extract information from a distribution that do not require human labor to annotate examples. The term is usually associated with density estimation, learning to draw samples from a distribution, finding a manifold that the data lies near, or clustering the data into groups of related examples.

A classic unsupervised learning task is to find the "best" representation of the data. By "best" we can mean different things, but generally speaking we are looking for a representation that preserves as

much information about $\mathbf{x}$ as possible while obeying some penalty or constraint aimed at keeping the representation simpler or more accessible than $\mathbf{x}$ itself. There are multiple ways of defining a simpler representation. Three of the most common include lower-dimensional representations, sparse representations and independent representations. Low-dimensional representations attempt to compress as much information about $\mathbf{x}$ as possible in a smaller representation. Sparse representations embed the dataset into a representation whose entries are mostly zeros for most inputs. The use of sparse representations typically requires increasing the dimensionality of the representation, so that the representation becoming mostly zeros does not discard too much information. This results in an overall structure of the representation that tends to distribute data along the axes of the representation space. Independent representations attempt to disentangle the sources of variation underlying the data distribution such that the dimensions of the representation are statistically independent.

A. k-means Clustering

Another example of a simple representation learning algorithm is k-means clustering. The k-means clustering algorithm divides the training set into k different clusters of examples that are near each other. We can thus think of the algorithm as providing a k-dimensional one-hot code vector $\mathbf{h}$ representing an input $\mathbf{x}$. If $\mathbf{x}$ belongs to cluster i, then $h_i = 1$, and all other entries of the representation $\mathbf{h}$ are zero. The one-hot code provided by k-means clustering is an example of a sparse representation, because the majority of its entries are zero for every input. One-hot codes are an extreme example of sparse representations that lose many of the benefits of a distributed representation. The one-hot code still confers some statistical advantages (it naturally conveys the idea that all examples in the same cluster are similar to each other), and it confers the computational advantage that the entire representation may be captured by a single integer.