Machine Learning

Krishna Chandra

July 11, 2022

Contents

PART I
MACHINE LEARNING THEORY

1 INTRODUCTION

1.1 ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) is wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence.

1.2 MACHINE LEARNING

1. AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data. This capability is known as machine learning.
2. What a typical "learning machine" does, is finding a mathematical formula, which, when applied to a collection of inputs (called "training data"), produces the desired outputs.
3. Machine learning can also be defined as the process of solving a practical problem by
   a) Gathering a dataset
   b) Algorithmically building a statistical model based on that dataset.

1.3 REPRESENTATION LEARNING

1. The performance of these simple machine learning algorithms depends heavily on the representation of the data they are given. Look at figure 4.1
2. Many artificial intelligence tasks can be solved by designing the right set of features to extract for that task, then providing these features to a simple machine learning algorithm.
3. It is difficult to know what features should be extracted. Therefore, it is important to use machine learning to discover not only the mapping from representation to output but also the representation itself. This approach is known as representation learning.
4. The typical example of a representation learning algorithm is the autoencoder. An autoencoder is the combination of an encoder function that converts the input data into a different representation, and a decoder function that converts the new representation back into the original format. Different kinds of autoencoders aim to achieve different kinds of properties.

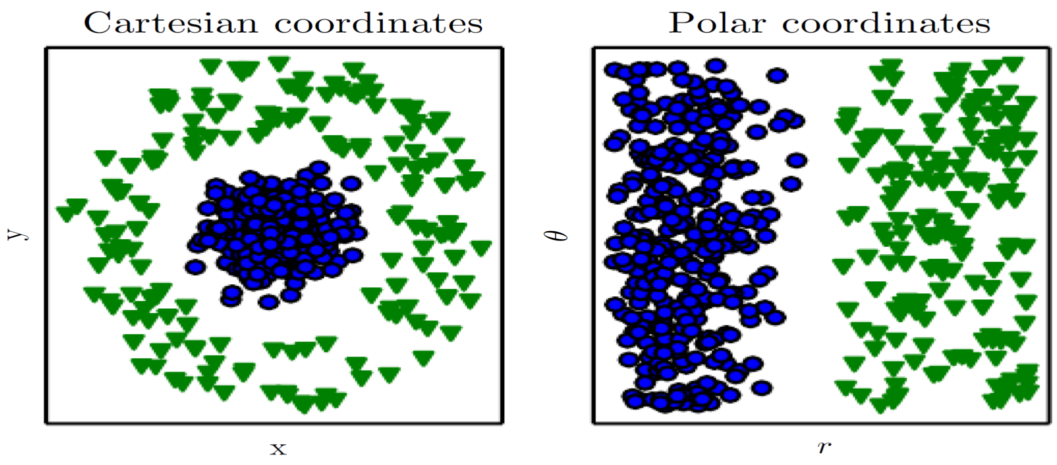


Figure 1.1: In the plot on the left, we represent some data using Cartesian coordinates, and the task is impossible. In the plot on the right, we represent the data with polar coordinates and the task becomes simple to solve with a vertical line.

1.4 Deep Learning

1. Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

2. Historical aspects of Deep Learning

   a) Deep learning known as cybernetics in the 1940s–1960s and known as connectionism in the 1980s–1990s, and deep learning beginning in 2006.

   b) The neural perspective on deep learning is motivated by two main ideas. One idea is that the brain provides a proof by example that intelligent behavior is possible, and a conceptually straightforward path to building intelligence is to reverse engineer the computational principles behind the brain and duplicate its functionality. Another perspective is that it would be deeply interesting to understand the brain and the principles that underlie human intelligence

   c) The Neocognitron introduced a powerful model architecture for processing images that was inspired by the structure of the mammalian visual system and later became the basis for the modern convolutional network.

   d) Modern deep learning draws inspiration from many fields, especially applied math fundamentals like linear algebra, probability, information theory, and numerical optimization.

   e) The field of DL is primarily concerned with how to build computer systems that are able to successfully solve tasks requiring intelligence, while the field of computational neuroscience is primarily concerned with building more accurate models of how the brain actually works.

   f) Connectionism arose in the context of cognitive science. Cognitive science is an interdisciplinary approach to understanding the mind, combining multiple different levels of analysis.

   g) The central idea in connectionism is that a large number of simple computational units can achieve intelligent behavior when networked together. This insight applies equally to neurons in biological nervous systems and to hidden units in computational models.

   h) One of these concepts is that of distributed representation. This is the idea that each input to a system should be represented by many features, and each feature should be involved in the representation of many possible inputs.

   i) During the 1990s, researchers made important advances in modeling sequences with neural networks,the long short-term memory or LSTM network had introduced to resolve the fundamental mathematical difficulties in modeling long sequences.

3. Why DL has started booming recently :

   a) Increasing dataset sizes.

   b) We have the computational resources to run much larger models today.

1.5 Reinforcement Learning

1. Reinforcement learning is a subfield of machine learning where an agent must learn to perform a task by trial and error, without any guidance from the human operator. The machine can execute actions in every state.

2. The machine can execute actions in every state. Different actions bring different rewards and could also move the machine to another state of the environment. The goal of a reinforcement learning algorithm is to learn a policy.

3. A policy is a function f (similar to the model in supervised learning) that takes the feature vector of a state as input and outputs an optimal action to execute in that state. The action is optimal if it maximizes the expected average reward.

2 Supervised Machine Learning Algorithms

The goal is, given a training set, to learn a function $h: X \to Y$ so that h(x) is a “good” predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis.

2.1 KNN Algorithm

ASSUMPTION Similar Inputs have similar outputs which imply that data points of various classes are not randomly sprinkled across the space, but instead appear in clusters of more or less homogeneous class assignments.

CLASSIFICATION RULE For a test input x, assign the most common label amongst its k most similar training inputs.

WHAT DISTANCE FUNCTION SHOULD WE USE? The k-nearest neighbor classifier fundamentally relies on a distance metric. The better that metric reflects label similarity, the better the classified will be.

1. **Minkowski distance** :
$$D(x,x') = \sum_i ((x'-x)^p)^{\frac{1}{p}}$$
   The most common choice is the
   a) $p = 1$ manhattan distance
   b) $p = 2$ Euclidean distance
   c) $p = \infty$ Max distance(max of difference of coordinates in each dimension)
2. **Cosine Similarity** :
$$CosineSimilarity(x,x') = \frac{x.x'}{|x||x'|}$$
   Generally, incase of classification of documents(bag of words), then it is better to consider cosine similarity.

NOTE : We need to choose distance metric wisely based on our problem.

HOW TO CHOOSE K

1. Generally we go for odd k.
2. In case of multi class, C1(3), C2(3), C3(1), C4(0), C1 and C2 have same number of neighbors, then we can check all its majority classes and assign that class which contains the closest point.

WHAT IF $k = n$ Then the output of all test points will be same as major class in the given data set irrespective of its input. Therefore, never ever choose $k = n$.

WHAT IF $k = 1$

1. The 1-NN could be an outlier of some another class. Then it miss-classifies the point. Therefore, its better not to choose.

2. As $n \to \infty$, the 1-NN(1 Nearest Neighbor) classifier is only a factor 2 worse than the best possible classifier.

3. Let x_{NN} be the nearest neighbor of our test point x_t. As $n \to \infty$, dist(x_{NN},x_t)$\to$0, i.e. x_{NN} $\to$x$_t$. (This means the nearest neighbor is identical to x_t.) You return the label of x_{NN}. What is the probability that this is not the label of x_t? (This is the probability of drawing two different label of x). Solving this probability, we reach to the above conclusion.

CURSE OF DIMENSIONALITY

1. As the dimension d of the input increases, the distance between two points in the space also increases.

2. So as d » 0 almost the entire space is needed to find the 10-NN.

3. This breaks down the k-NN assumptions, because the k-NN are not particularly closer (and therefore more similar) than any other data points in the training set. Why would the test point share the label with those k-nearest neighbors, if they are not actually similar to it?

4. Dont try to visualize the above point as it involves multiple dimensions greater than 3. Do the math.

5. Note : In real life, points are not uniformly distributed. Points mostly lie on the complex manifolds and may form clusters. That's why KNN works sometime even for higher dimensions.

PROS AND CONS   Pros :

1. Easy and simple

2. Non parametric

3. No assumption about data.

4. Only two choices(k values and distance metric).

Cons : Computation time : O(N.d) where N -> training samples and d -> dimensions.

Note : No need to worry about the sorting here because as we compute distance, we can sort then and there itself in linear time.

K Dimensional Tress(KD Trees)

1. Building KD trees.
   a) Split data recursively in half on exactly one feature.
   b) Rotate through features.
   c) When rotating through features, a good heuristic is to pick the feature with maximum variance.

   Max height of the tree could be $log_2(n)$.

2. Finding NN for the test point(X,y).
   a) Find region containing (x,y).
   b) Compare to all points in the region.

3. How can this partitioning speed up testing?
   - Let's think about it for the one neighbor case.
   - Identify which side the test point lies in, e.g. the right side.
   - Find the nearest neighbor ${x_{NN}}^R$ of x_t in the same side. The R denotes that our nearest neighbor is also on the right side.
   - Compute the distance between x_y and the dividing "wall". Denote this as d_w. If $d_w > d(x_t, {x_{NN}}^R)$ you are done, and we get a $2\times$ speedup.

4. Pros: Exact and Easy to build.

5. Cons:
   - Curse of Dimensionality makes KD-Trees ineffective for higher number of dimensions. May not work better if dimensions are greater than 10.
   - All splits are axis aligned (all dividing hyperplanes are parallel to axis).

6. Approximation: Limit search to m leafs only.

Ball Trees

1. Similar to KD-trees, but instead of boxes use hyper-spheres (balls). If the distance to the ball, d_b, is larger than distance to the currently closest neighbor, we can safely ignore the ball and all points within. The ball structure allows us to partition the data along an underlying manifold that our points are on, instead of repeatedly dissecting the entire feature space (as in KD-Trees).

2. Construction : Refer to fig 2.1

3. Ball-Tree Use :
   - Same as KD-Trees

Algorithm 1 Balltree in Pseudo-code

```
1:  procedure BALLTREE($S,k$)
2:      if $|S| < k$ then stop end if                         ▷ Return leaf containing $S$
3:      pick $x_0 \in S$ uniformly at random
4:      pick $x_1 = \text{argmax}_{x \in S}\ d(x_0,x)$
5:      pick $x_2 = \text{argmax}_{x \in S}\ d(x_1,x)$
6:      $\forall i = 1 \dots |S|, z_i = (x_1 - x_2)^T x_i$      $\leftarrow$ project data onto $(x_1 - x_2)$
7:      $m = \text{median}(z_1, \cdots, z_{|S|})$
8:      $S_L = \{x \in S : z_i < m\}$
9:      $S_R = \{x \in S : z_i \geq m\}$
10:     Return tree:
        – center $c = \text{mean}(S)$
        – radius $r = \max_{x \in S} d(x,c)$
        – children: Balltree($S_L,k$) and Balltree($S_R,k$)
11: end procedure
```

Figure 2.1: Construction of Ball Tress Algorithm

- Slower than KD-Trees in low dimensions (d3) but a lot faster in high dimensions. Both are affected by the curse of dimensionality, but Ball-trees tend to still work if data exhibits local structure (e.g. lies on a low-dimensional manifold).

LOCALLY SENSITIVE HASHING(LSH)

1. Divide the whole space of points into $\frac{n}{2^k}$ regions by randomly drawing k hyperplanes$(h_1, h_2, h_3, ..., h_k)$.
2. Compare x to only those $\frac{n}{2^k}$ points in that particular region.
3. Complexity : $O(Kd + d\frac{n}{2^k})$.
   a) Kd : To find out which point belongs to which region. For that we need to check with each hyperplane and find that.
   b) $d\frac{n}{2^k}$: Finding the NN by comparing d-dimensional point with $\frac{n}{2^k}$ points.
4. Limitations : Choosing the right set and number of hyperplanes are really important. Try a couple of different initializations. Based on the way we choose the hyperplanes, we may miss out the main neighbors and misclassify the point.

CONCLUSION

1. KNN is best suitable if dimensions are low and high number of data points.
2. KNN works better incase of images and faces though the data is highly dimensoinal due to its sparsity. Moreover, similar points similar labels assumption kind of stays true.
3. More the number of data points, slow is the algorithm.

```
Initialize $\vec{w} = \vec{0}$                  // Initialize $\vec{w}$. $\vec{w} = \vec{0}$ misclassifies everything.
while TRUE do                                   // Keep looping
    $m = 0$                                     // Count the number of misclassifications, $m$
    for $(x_i, y_i) \in D$ do                   // Loop over each (data, label) pair in the dataset, $D$
        if $y_i(\vec{w}^T \cdot \vec{x_i}) \leq 0$ then   // If the pair $(\vec{x_i}, y_i)$ is misclassified
            $\vec{w} \leftarrow \vec{w} + y\vec{x}$       // Update the weight vector $\vec{w}$
            $m \leftarrow m + 1$                          // Counter the number of misclassification
        end if
    end for
    if $m = 0$ then                             // If the most recent $\vec{w}$ gave 0 misclassifications
        break                                   // Break out of the while-loop
    end if
end while                                       // Otherwise, keep looping!
```

Figure 2.2: Perceptron Algorithm

2.2 Perceptron Algorithm

Basic Understanding The hyperplane separates the two classes. The hyperplane is found by finding its weights.

$$h(x_i) = sign(wx_i + b) \rightarrow eq(1)$$

Based on which side of hyperplane the point lies, its class is decided. The pseudo code of the algorithm is given in fig 2.2. Here, we include bias in the weight vector itself by adding an extra dimension of constant 1 in the input.

The update $w = w + yx$ moves the w vector(hyperplane) in direction of x. Proof : After updating the w, $(w + yx_i)^T x_i = w^T x + yx^T x$. Let this equation be eq(2). Here two cases :

1. In case of false positive : y = 0, eq(1) > 0, we need to decrease its value so that it becomes less than 0. By updating the weight vector, it reduces the eq(1) by $x^T.x$ amount.

2. Quite opposite is the case with false negative.

Assumptions

1. Binary classification (i.e. $y_i \in \{-1, +1\}$)

2. Data is linearly separable

Perceptron Convergence If the data is linearly separable, then the algo will converge definitely, otherwise the algo will loop infinetly and will never converge. Proof can be found at `https://www.youtube.com/watch?v=vAOI9kTDVoo&list=PLEAYkSg4uSQ1r-2XrJ_GBzzS6I-f8yfRU&index=16`

2.3 Generative Learning algorithms

Intro

1. Algorithms that try to learn p(y|x) directly (such as logistic regression), or algorithms that try to learn mappings directly from the space of inputs X to the labels 0, 1, (such as the perceptron algorithm) are called discriminative learning algorithms.
2. Algorithms that instead try to model p(x|y) are generative learning algorithms.
3. For instance, if y indicates whether an example is a dog (0) or an elephant (1), then p(x|y = 0) models the distribution of dogs features, and p(x|y = 1) models the distribution of elephants features.

Gaussian Discriminant Analysis .

1. In this model, we'll assume that p(x|y) is distributed according to a multivariate normal distribution.
2. This Algorithm make use of Bayes theorem to model p(y|x).

$$p(x;\mu,\Sigma) = \sqrt{\frac{1}{(2\pi)^n det(\Sigma)}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

Here, Σ = Covariance of X. Here, μ is computed for each class inputs separately. and covariance is common.

$$y \approx Bernoulli(\phi)$$

$$x|y = 0 \approx N(\mu_0, \Sigma)$$

$$x|y = 1 \approx N(\mu_1, \Sigma)$$

$$p(x;\mu_0,\Sigma) = \sqrt{\frac{1}{(2\pi)^n det(\Sigma)}} exp(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0))$$

$$p(x;\mu_1,\Sigma) = \sqrt{\frac{1}{(2\pi)^n det(\Sigma)}} exp(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1))$$

3. The log-likelihood of the data is given by

$$l(\phi,\mu_0,\mu_1,\Sigma) = log\prod_{i=1}^{m} p(x(i), y(i); \phi,\mu_0,\mu_1,\Sigma)$$

$$l(\phi,\mu_0,\mu_1,\Sigma) = log\prod_{i=1}^{m} p(x(i)|y(i); \phi,\mu_0,\mu_1,\Sigma) \times p(y(i):\phi)$$

By maximizing l with respect to the parameters, we find the maximum likelihood estimate of the parameters (see problem set 1) to be:

$$\phi = \frac{1}{m}\sum_{i=1}^{m} 1\{y(i) = 1\}$$

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y(i) = 0\} x(i)}{\sum_{i=1}^{m} 1\{y(i) = 0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y(i) = 1\} x(i)}{\sum_{i=1}^{m} 1\{y(i) = 1\}}$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x(i) - \mu_{y^(i)})(x(i) - \mu_{y^(i)})^T$$

4. **GDA vs Logistic Regression** : GDA works best if X distribution is multivariate. But LR works best even for other distributions also.

2.4 Naive Bayes Classifier

Introduction

1. In GDA, the feature vectors x were continuous, real-valued vectors.
2. Naive Bayes is different learning algorithm in which the xj 's are discrete-valued.
3. This algorithm is mostly used in case of text because words are discrete.
4. Naive Bayes works best if we have very less data.
5. It tries to model the data and finds out the parameters for the distributions of the classes and based on those parameters, it predicts the new data point.
6. This is extremely fast because there is no loop or anything. Just need to find out the parameters for the class distributions.
7. Naive Bayes is a linear classifier. You can find that proof in cornell lec 11 in first 10 mins.

Assumptions All the discrete features are independent of each other given the label.

Classifier In case of a binary classifier, $y \in \{0, 1\}$ and $X : X_{M\times N}$.

$P(X|Y = 0) = P(X_1|Y = 0) \times P(X_2|Y = 0)........... \times P(X_N|Y = 0) \rightarrow Featuresareindependent$

$$P(X|Y = 1) = P(X_1|Y = 1) \times P(X_2|Y = 1)........... \times P(X_N|Y = 1)$$

Using the Bayes theorem, we can find $P(Y|X)$.

Note Even if the naive bayes assumption violates, this algo works very well.
If the naive Bayes assumption holds, then Naive Bayes works similar to Logitstic Regression. Proof can be found in cornell notes.
PS : This is easy. Watch Krish naik video for intution and for depth understanding, watch cornell videos.

2.5 Linear Regression

Introduction

1. In Linear Regression, We try to fit the data to a linear function. Let y as a linear function of x:
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$
2. Here the output is continuous value. $y \in \Re$
3. To simplify our notation, we also introduce the convention of letting $x_0 = 1$ (this is the intercept term), so that
$$h(x) = \Sigma_{i=0}{}^m \theta_i x_i = \theta^T x$$
4. Our objective is to make h(x) close to y, at least for the training examples we have. Therefore, we try to minimize the loss function :
$$J(\theta) = \frac{1}{2}\sum_{i=0}^{m}(h_\theta(x^i) - y^i)^2$$

Note : We can go for the different loss function, that is absolute loss(MAE), power 4 etc. MAE is not differentiable at 0. Power4 or any other even power function penalizes the outliers very much and also those turns out to be non-convex function.

Minimization of Loss Function

1. We want to choose θ so as to minimize $J(\theta)$.
2. Let's consider the gradient descent algorithm, which starts with some initial , and repeatedly performs the update:
$$\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$$
where α is the learning rate.
3. The (unweighted) linear regression algorithm that we saw earlier is known as a parametric learning algorithm, because it has a fixed, finite number of parameters (the i's), which are fit to the data. Once we've fit the θi's and stored them away, we no longer need to keep the training data around to make future predictions.

Assumptions

1. Linearity: The relationship between X and the mean of Y is linear.
2. Homoscedasticity: The variance of residual is the same for any value of X.
3. Independence: Observations are independent of each other.(I.I.D)
4. Normality: For any fixed value of X, Y is normally distributed

Normal method to minimize the loss

1. X $[]_{m\times n}$ and y $[]_{m\times 1}$.
2. We have to minimize
$$J = \frac{1}{2}(X\theta - y)^T(X\theta - y)$$
3. Therefore, $\nabla_\theta J = 0$. After differentating the J, we get
$$\theta = (X^TX)^{-1}X^TY$$
(Refer to stanford notes for proof in detail.)
4. Using the property
$$\nabla_{A^T} trABA^TC = B^TA^TC^T + BA^TC$$

Probabilistic Interpretation

1. When faced with a regression problem, why might linear regression, and specifically why might the least-squares cost function J, be a reasonable choice?
2. Let us assume that the target variables and the inputs are related via the equation
$$y(i) = \theta^Tx(i) + \epsilon(i)$$
, where $\epsilon(i)$ is an error term.
3. Assumptions :
   a) $\epsilon(i)$ are distributed IID (independently and identically distributed).
   b) $\epsilon(i) \approx N(0,\sigma^2)$. This is same as $y_i \approx N(w^Tx_i,\sigma^2)$
   c) $\sigma^2 = 1$ This doesn't effect the calculation
4. Therefore,
$$p(\epsilon) = \mathcal{N}(x;0,1) = \sqrt{\frac{1}{2\pi}}exp(-\frac{1}{2}(y^i - \theta^Tx^i)^2)$$
This implies that ,
$$p(y^i|x^i;\theta) = \sqrt{\frac{1}{2\pi}}exp(-\frac{1}{2}(y^i - \theta^Tx^i)^2)$$
Try to maximize the log likelihood.

2.6 Locally weighted Linear regression

Why

1. Sometimes we cannot fit a straight line to the whole data. In those cases, if we consider only few datapoints near to the point we wanted to estimate, we may fit the line to those points in the neighborhood of that points.

HOW

1. To do so, we assign more weight to the points near to the test point and less weight to points far from it.
2. Each time we want to predict y for a test point x', we need to fit the line to points near to x'.
3. Assigning weights will be done using the formula :
$$w(i) = exp[-\frac{(x(i)-x')^2}{\tau^2}]$$
4. Fit to minimize
$$\sum_i w(i)(y(i)-\theta^T x(i))^2$$
.
5. The parameter τ controls how quickly the weight of a training example falls off with distance of its x(i) from the query point x; τ is called the bandwidth parameter,
6. This is **non-parametric** learning algorithm (i,e) to make predictions using locally weighted linear regression, we need to keep the entire training set around.

2.7 LOGISTIC REGRESSION(LR)

INTRO

1. This is used for classification.
2. This is same as Linear regression, but as we need to classify, we need discrete outputs not continuous. Therefore, we use sigmoid function at the end.
$$h_\theta x = sigmoid(\theta^T x)$$
3. Here sigmoid gives the probability of that training example belonging to class 1.

MINIMIZATION OF LOSS FUNCTION

1. We want to choose θ so as to minimize $J(\theta)$.
2. To minimize J, we find out its derivative and surprisingly, derivative will end up as same as in case of linear regression.
3. Let's consider the gradient descent algorithm, which starts with some initial θ, and repeatedly performs the update:
$$\theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial\theta_j}$$
4. If different features have different scales, then the loss is dominated by large scale features.

Probabilistic Interpretation

1. Here in case of binary classification, the outputs have the bernoulli distribution(0 and 1) and in case of multi class classification, they have multinouli distribution.
2. Binary classification :
$$P(y=1|x;) = h_\theta(x)$$
$$P(y=0|x;) = 1 - h_\theta(x)$$
$$p(y|x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{1-y}$$
Assuming that the m training examples were generated independently, we can then write down the likelihood of the parameters as
$$L(\theta) = p(\ y|X;\theta) = \Pi^m{}_{i=1}p(y(i)|x(i);\theta)$$
We try to maximize the log -likelihood function. In this process, we end up finding the logistic or sigmoid function.

2.8 Generalized Linear models

Intro

1. We've seen a regression example, and a classification example. In the regression example, we had $y|x;\theta \approx N(\mu,\sigma^2)$, and in the classification one, $y|x;\theta \approx Bernoulli(\phi)$, for some appropriate definitions of μ and as functions of x and θ. In this section, we will show that both of these methods are special cases of a broader family of models, called Generalized Linear Models (GLMs).
2. We say that a class of distributions is in the exponential family if it can be written in the form
$$p(y;\eta) = b(y)exp(\eta^T T(y) - a(\eta))$$
η - the natural parameter (also called the canonical param-eter) of the distribution;
T(y) - sufficient statistic (for the distribu- tions we consider, it will often be the case that T(y) = y);
$a(\eta)$ - log partition function. The quantity $e^{-a(\eta)}$ essentially plays the role of a normalization constant, that makes sure the distribution $p(y;\eta)$ sums/integrates over y to 1.
3. A fixed choice of T, a and b defines a family (or set) of distributions that is parameterized by η; as we vary η, we then get different distributions within this family.

Distributions in Exponential Family

1. Bernoulli
2. Gaussian
3. Multinoulli

BERNOULLI

1. The Bernoulli distribution with mean , written Bernoulli(), specifies a distribution over $y \in 0, 1$, so that
$$p(y = 1; \phi) = \phi; p(y = 0; \phi) = 1 - \phi$$
.

2. As we vary ϕ, we obtain Bernoulli distributions with different means. We now show that this class of Bernoulli distributions, ones obtained by varying , is in the exponential family; i.e., that there is a choice of T, a and b so that above equation becomes exactly the class of Bernoulli distributions.
$$p(y;\phi) = \phi^y(1-\phi)^{1-y} = exp(ylog\phi + (1-y)log(1-\phi)) = exp((log(\frac{\phi}{1-\phi}))y + log(1-\phi))$$

3. Thus, the natural parameter is given by $\eta = log(\frac{\phi}{(1-\phi)})$. Interestingly, if we invert this definition for η by solving for ϕ in terms of η, we obtain $= 1/(1+e^{-\eta})$. This is the familiar sigmoid function!

4. The formulation of the Bernoulli distribution as an exponential family distribution,
$$T(y) = y$$
$$a(\eta) = -log(1-\phi) = log(1+e^{\eta})$$
$$b(y) = 1$$

GAUSSIAN

1. When deriving linear regression, the value of σ^2 had no effect on our final choice of θ and $h_\theta(x)$.To simplify the derivation below, let's set $\sigma^2 = 1$.
$$p(y;\mu) = \frac{1}{\sqrt(2\pi)} exp(-\frac{1}{2}(y-\mu)^2) = \frac{1}{\sqrt(2\pi)} exp(-\frac{1}{2}y^2) exp(\mu y - \frac{1}{2}\mu^2)$$

2. $\eta = \mu$, $T(y) = y$, $a(\eta) = \frac{\mu^2}{2}$ $b(y) = \frac{1}{\sqrt(2\pi)} exp(\frac{y^2}{2})$.

3. If we leave σ^2 as a variable, the Gaussian distribution can also be shown to be in the exponential family, where $\eta \in \Re^2$ is now a 2-dimension vector that depends on both μ and σ.

4. For the purposes of GLMs, however, the σ^2 parameter can also be treated by considering a more general definition of the exponential family:
$$p(y;\mu,\sigma^2) = b(a,\tau)exp((\eta^T T(y) - a(\eta))/c(\tau))$$
.

5. Here, σ^2 is called the dispersion parameter, and for the Gaussian, $c(\tau) = \sigma^2$; but given our simplification above, we won't need the more general definition for the examples we will consider here.

MULTINOULLI

1. Consider a classification problem in which the response variable y can take on any one of k values, so $y \in 1, 2, ..., k$.
2. To parameterize a multinomial over k possible outcomes, one could use k parameters $\phi_1, ..., \phi_k$ specifying the probability of each of the outcomes.
3. These parameters would be redundant, or more formally, they would not be independent (since knowing any k-1 of the ϕ_i's uniquely determines the last one, as they must satisfy $\sum_{i=1}^{k} \phi_i = 1$)
4. So, we will instead parameterize the multinomial with only k-1 parameters, $\phi_1, ..., \phi_{k-1}$
5. To express the multinomial as an exponential family distribution, we will define $T(y) \in \Re^{k-1}$ as follows: T(1) = [1 0 0 .....0] T(2) = [0 1 0 0 ... 0] .... T(k) = [0 0 0 ..... 0]
6. We will write $(T(y))_i$ to denote the i-th element of the vector T(y). $(T(y))_i = 1y = i$(if condition is true, it returns 1 otherwise 0).

$$p(y;\phi) = \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} ......\phi_k^{1\{y=k\}}$$

$$p(y;\phi) = \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} ......\phi_k^{1-\sum_{i=1}^{k} 1\{y=k\}}$$

Using $e^{logN} = N$ and $(T(y))_i = 1\{y = i\}$ and simplify :

$$p(y;\phi) = exp((T(y))_1 log(\frac{\phi_1}{\phi_k}) + (T(y))_2 log(\frac{\phi_2}{\phi_k}) + ... + (T(y))_{k-1} log(\frac{\phi_{k1}}{\phi_k}) + log(\phi_k))$$

$$p(y;\eta) = b(y)exp(\eta^T T(y) - a(\eta))$$

where

$$\eta = [log(\frac{\phi_1}{\phi_k}) log(\frac{\phi_2}{\phi_k}) ....log(\frac{\phi_{k-1}}{\phi_k})]^T$$

$$a(\eta) = -log(\phi_k)$$

$$b(y) = 1$$

The link function is given (for i = 1, . . . , k) by

$$\eta_i = log((\frac{\phi_i}{\phi_k}))$$

We need to find the response function, ϕ_i. Using $\phi_k = \frac{1}{\sum_{i=1}^{k} e^{\eta_i}}$ to give response function :

$$\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}}$$

7. This function mapping from the η's to the ϕ's is called the softmax function.

2.9 SUPPORT VECTOR MACHINES(SVM)

1. Consider a positive training example (y = 1). The larger $\theta^T x$ is, the larger also is h(x) = p(y = 1|x;w, b), and thus also the higher our degree of "confidence" that the label is 1.

2. Again informally it seems that we'd have found a good fit to the training data if we can find so that $\theta^T x(i) >> 0$ whenever y(i) = 1, and $\theta^T x(i) << 0$ whenever y(i) = 0, since this would reflect a very confident (and correct) set of classifications for all the training examples.

3. we'll use $y \in \{-1, 1\}$ (instead of $\{0, 1\}$) to denote the class labels.

$$h_{w,b}(x) = g(w^T x + b)$$

   Here, $g(z) = 1$ if $z \geq 0$, and $g(z) = -1$ otherwise. w takes the role of $[\theta_1 ... \theta_n]^T$ and b is intercept term.

4. Note also that, from our definition of g above, our classifier will directly predict either 1 or -1, without first going through the intermediate step of estimating the probability of y being 1.

5. Given a training example (x(i), y(i)), we define the functional margin of (w, b) with respect to the training example

$$\gamma(i) = y(i)(w^T x + b)$$

6. if $y(i)(w^T x + b) > 0$, then our prediction on this example is correct.

7. Given our choice of g, we note that if we replace w with 2w and b with 2b, then since $g(w^T x + b) = g(2w^T x + 2b)$,this would not change $h_{w,b}(x)$ at all. I.e., g, and hence also $h_{w,b}(x)$, depends only on the sign, but not on the magnitude, of $w^T x + b$. However, replacing (w, b) with (2w, 2b) also results in multiplying our functional margin by a factor of 2.

8. Intuitively, it might therefore make sense to impose some sort of normalization condition such as that $||w||_2 = 1$ i.e., we might replace (w, b) with $(w/||w||_2, b/||w||2)$, and instead consider the functional margin of $(w/||w||_2, b/||w||_2)$.

9. Given a training set S = (x(i), y(i)); i = 1, . . . ,m, we also define the function margin of (w, b) with respect to S as the smallest of the functional margins of the individual training examples.

$$\gamma^\wedge(i) = min_{i=1,...,m}\gamma(i).$$

10. **Geometric Margins** : The geometric margin of (w, b) with respect to a training example (x(i), y(i)) to be

$$\gamma(i) = y(i)(\frac{w^T}{||w||}x(i) + \frac{b}{||w||})$$

11. Note that if $||w|| = 1$, then the functional margin equals the geometric margin — this thus gives us a way of relating these two different notions of margin.

12. Specifically, because of this invariance to the scaling of the parameters, when trying to fit w and b to training data, we can impose an arbitrary scaling constraint on w without changing anything important; for instance, we can demand that $||w|| = 1$, or$|w1| = 5$, or $|w1 + b| + |w2| = 2$, and any of these can be satisfied simply by rescaling w and b.

OBJECTIVE

$$max_{w,b}\gamma$$

s.t. $y(i)(w^T x(i) + b) \geq \gamma$ for i = 1, . . . ,m and $||w|| = 1$

1. I.e., we want to maximize γ, subject to each training example having functional margin at least γ. The $||w|| = 1$ constraint moreover ensures that the functional margin equals to the geometric margin, so we are also guaranteed that all the geometric margins are at least γ.

2. If we could solve the optimization problem above, we'd be done. But the "$||w|| = 1$" constraint is a nasty (non-convex) one, and this problem certainly isn't in any format that we can plug into standard optimization software to solve. So, let's try transforming the problem into a nicer one.

$$max_{w,b}\frac{\gamma}{||w||}$$

s.t. $y(i)(w^T x(i) + b) \geq \gamma$ for i = 1, . . . ,m

3. Here, we're going to maximize $\frac{\gamma^\wedge}{||w|}$, subject to the functional margins all being at least $\gamma^\wedge$. Since the geometric and functional margins are related by $\gamma = \frac{\gamma^\wedge}{||w|}$, this will give us the answer we want.

4. Recall our earlier discussion that we can add an arbitrary scaling constraint on w and b without changing anything. This is the key idea we'll use now.

5. We will introduce the scaling constraint that the functional margin of w, b with respect to the training set must be 1:

$$\gamma^\wedge = 1$$

6. Since multiplying w and b by some constant results in the functional margin being multiplied by that same constant, this is indeed a scaling constraint, and can be satisfied by rescaling w, b. Plugging this into our problem above, and noting that maximizing

$$\frac{\gamma^\wedge}{||w||} = \frac{1}{||w||}$$

is the same thing as minimizing $||w||^2$, we now have the following optimization problem:

$$min_{w,b}||w||^2$$

s.t. $y(i)(w^T x(i) + b) \geq 1$ for $i = 1,..., m$

7. The above is an optimization problem with a convex quadratic objective and only linear constraints. Its solution gives us the optimal margin classifier.

Note : The mathematical derivation of SVM can be found here. `https://iiitaphyd-my.sharepoint.com/:b:/g/personal/krishna_chandra_research_iiit_ac_in/EWGaLL7gMy9Ht68sJW--RmgE` `g?e=G7vngX` and the clarity will be given in cornell lec14 at 35 : 00.

WHAT IF THE DATA IS NOT LINEARLY SEPARABLE In this case, the constraint is never satisfied and we can never find the hyperplane. Therefore, a slack has been introduced. Now the objective and constraint is slightly modified so that the constraint is satisfied,

$$min_{w,b}||w||^2 + C.\sum_i \epsilon_i$$

s.t. $y(i)(w^T x(i)+b) \geq 1$ and $\epsilon_i \geq 0$ for all i. $\epsilon_i = max(0, 1-y(i)(w^T x(i)+b))$. C is a hyperparameter set by us. This penalizes the outliers or the datapoints which doesn't satisfy the constraints. For C, try from $10^{-4} to 100$ and check which one works better. For clear understanding, look at fig**??**

PARAMETRIC VS NON-PARAMETRIC ALGORITHMS An interesting edge case is kernel-SVM. Here it depends very much which kernel we are using. E.g. linear SVMs are parametric (for the same reason as the Perceptron or logistic regression). So if the kernel is linear the algorithm is clearly parametric. However, if we use an RBF kernel then we cannot represent the classifier of a hyper-plane of finite dimensions. Instead we have to store the support vectors and their corresponding dual variables i – the number of which is a function of the data set size (and complexity). Hence, the kernel-SVM with an RBF kernel is non-parametric. A strange in-between case is the polynomial kernel. It represents a hyper-plane in an extremely high but still finite-dimensional space. So technically one could represent any solution of an SVM with a polynomial kernel as a hyperplane in an extremely high dimensional space with a fixed number of parameters, and the algorithm is therefore (technically) parametric. However, in practice this is not practical. Instead, it is almost always more economical to store the support vectors and their corresponding dual variables (just like with the RBF kernel). It therefore is technically parametric but for all means and purposes behaves like a non-parametric algorithm.

IMPORTANT Q & A

1. Why shouldn't we incorporate bias as a constant features when working with SVMs ?

   If we include bias in the datapoint, then when maximizing the margin, we are trying to minimize the $||w||^2 + b^2$ which changes the objective function and this is not what we want. In the formula of distance from point to plane, we dont include bias in the denominator. therefore, bias will not play any role in the objective function, but it plays in the satisfying the constraint.

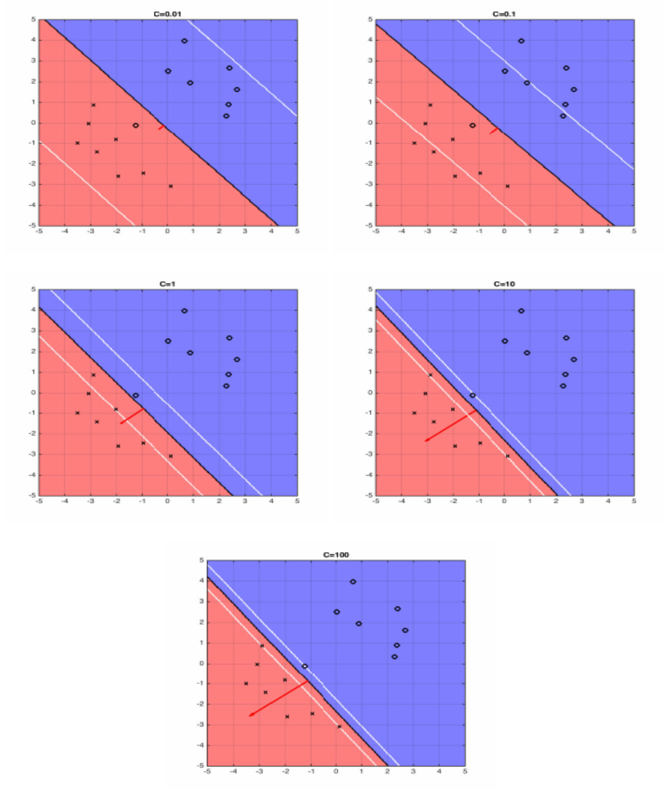


Figure 2.3: Variation of Hyperplane with parameter C

2.10 Empirical Risk Minimization

Reference https://www.cs.cornell.edu/courses/cs4780/2018sp/lectures/lecturenote10.html
Remember the Unconstrained SVM Formulation

$$\min_{\mathbf{w}} \underbrace{C\sum_{i=1}^{n} max[1-y_i\underbrace{(w^\top \mathbf{x}_i+b)}_{h(\mathbf{x}_i)},0]}_{Hinge-Loss} + \underbrace{\|w\|_z^2}_{l_2-Regularizer}$$

The hinge loss is the SVM's loss/error function of choice, whereas the l_2-regularizer reflects the complexity of the solution, and penalizes complex solutions. Unfortunately, it is not always possible or practical to minimize the true error, since it is often not continuous and/or differentiable. However, for most Machine Learning algorithms, it is possible to minimize a "Surrogate" Loss Function, which can generally be characterized as follows:

$$\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \underbrace{l_{(s)}(h_{\mathbf{w}}(\mathbf{x}_i),y_i)}_{Loss} + \underbrace{\lambda r(w)}_{Regularizer}$$

...where the Loss Function is a continuous function which penalizes training error, and the Regularizer is a continuous function which penalizes classifier complexity. Here we define λ as $\frac{1}{C}$.
The science behind finding an ideal loss function and regularizer is known as Empirical Risk Minimization or Structured Risk Minimization.

Commonly Used Binary Classification Loss Functions

1. As hinge-loss decreases, so does training error.
2. As $z \to -\infty$, the log-loss and the hinge loss become increasingly parallel.
3. The exponential loss and the hinge loss are both upper bounds of the zero-one loss.
4. Zero-one loss is zero when the prediction is correct, and one when incorrect.

Refer table 2.1

Commonly Used Regression Loss Functions

Regularizers

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} \ell(\mathbf{w}^\top \vec{x}_i+b,y_i)+\lambda r(\mathbf{w}) \Leftrightarrow \min_{\mathbf{w},b} \sum_{i=1}^{n} \ell(\mathbf{w}^\top \vec{x}_i+b,y_i) \text{ subject to: } r(w)\le B \tag{2.1}$$

For each $\lambda \ge 0$, there exists $B \ge 0$ such that the two formulations in (4.1) are equivalent, and vice versa. In previous sections, l_2-regularizer has been introduced as the component in SVM that reflects the complexity of solutions. Besides the l_2-regularizer, other types of useful regularizers and their properties are listed in 2.3

Metrics		
Loss $\ell(h_{\mathbf{w}}(\mathbf{x}_i, y_i))$	Usage	Comments
1.Hinge-Loss $max\left[1-h_{\mathbf{w}}(\mathbf{x}_i)y_i, 0\right]^p$	Standard SVM($p=1$) (Differentiable) Squared Hingeless SVM ($p=2$)	When used for Standard SVM, the loss function denotes margin length between linear separator and its closest point in either class. Only differentiable everywhere at $p=2$.
2.Log-Loss $log(1+e^{-h_{\mathbf{w}}(\mathbf{x}_i)y_i})$	Logistic Regression	One of the most popular loss functions in Machine Learning, since its outputs are very well-tuned.
3.Exponential Loss $e^{-h_{\mathbf{w}}(\mathbf{x}_i)y_i}$	AdaBoost	This function is very aggressive. The loss of a mis-prediction increases exponentially with the value of $-h_{\mathbf{w}}(\mathbf{x}_i)y_i$.
4.Zero-One Loss $\delta(\text{sign}(h_{\mathbf{w}}(\mathbf{x}_i)) \neq y_i)$	Actual Classification Loss	Non-continuous and thus impractical to optimize.

Table 2.1: Loss Functions With Classification $y \in \{-1, +1\}$

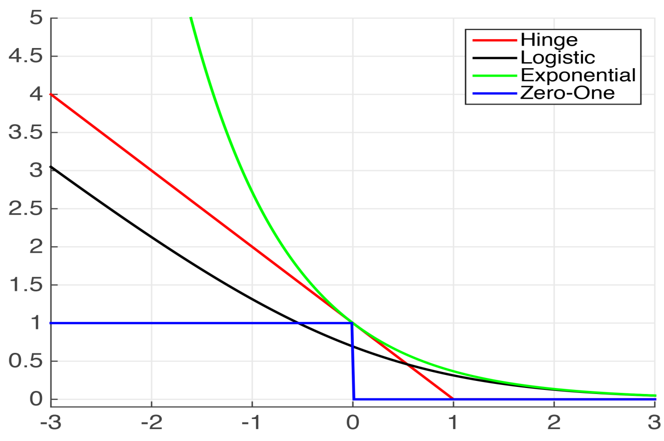


Figure 2.4: Plots of Common Classification Loss Functions - x-axis: $h(\mathbf{x}_i)y_i$, or "correctness" of prediction; y-axis: loss value

Loss $\ell(h_{\mathbf{w}}(\mathbf{x}_i, y_i))$	Comments
1.Squared Loss $(h(\mathbf{x}_i) - y_i)^2$	• Most popular regression loss function • Estimates Mean Label • ADVANTAGE: Differentiable everywhere • DISADVANTAGE: Somewhat sensitive to outliers/noise • Also known as Ordinary Least Squares (OLS)
2.Absolute Loss $\|h(\mathbf{x}_i) - y_i\|$	• Also a very popular loss function • Estimates Median Label • ADVANTAGE: Less sensitive to noise • DISADVANTAGE: Not differentiable at 0
3.Huber Loss $\frac{1}{2}\left(h(\mathbf{x}_i) - y_i\right)^2$ if $\|h(\mathbf{x}_i) - y_i\| < \delta$, otherwise $\delta(\|h(\mathbf{x}_i) - y_i\| - \frac{\delta}{2})$	1. Also known as Smooth Absolute Loss 2. ADVANTAGE: "Best of Both Worlds" of Squared and Absolute Loss 3. Once-differentiable 4. Takes on behavior of Squared-Loss when loss is small, and Absolute Loss when loss is large. .
4.Log-Cosh Loss $log(cosh(h(\mathbf{x}_i) - y_i))$, $cosh(x) = \frac{e^x + e^{-x}}{2}$	ADVANTAGE: Similar to Huber Loss, but twice differentiable everywhere

Table 2.2: Loss Functions With Regression, i.e. $y \in \mathbb{R}$

Regularizer $r(\mathbf{w})$	Properties
1.l_2-Regularization $r(\mathbf{w}) = \mathbf{w}^\top \mathbf{w} = (\|\mathbf{w}\|_2)^2$	• ADVANTAGE: Strictly Convex • ADVANTAGE: Differentiable • DISADVANTAGE: Uses weights on all features, i.e. relies on all features to some degree (ideally we would like to avoid this) - these are known as Dense Solutions.
2.l_1-Regularization $r(\mathbf{w}) = \|\mathbf{w}\|_1$	• Convex (but not strictly) • DISADVANTAGE: Not differentiable at 0 (the point which minimization is intended to bring us to • Effect: Sparse (i.e. not Dense) Solutions
3.Elastic Net $\alpha\|\mathbf{w}\|_1 + (1-\alpha)(\|\mathbf{w}\|_2)^2$ $\alpha \in [0,1)$	• ADVANTAGE: Strictly convex (i.e. unique solution) • DISADVANTAGE: Non-differentiable .
4.lp-Norm often $0 < p \leq 1$ $\|\mathbf{w}\|_p = (\sum_{i=1}^{d} v_i^p)^{1/p}$	• DISADVANTAGE: Non-convex • ADVANTAGE: Very sparse solutions • Initialization dependent • DISADVANTAGE: Not differentiable

Table 2.3: Loss Functions With Regression, i.e. $y \in \mathbb{R}$

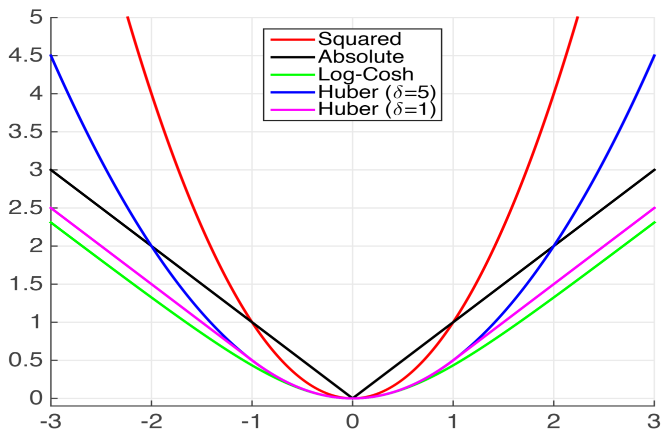


Figure 2.5: Plots of Common Regression Loss Functions - x-axis: $h(\mathbf{x}_i) y_i$, or "error" of prediction; y-axis: loss value

2.11 Bias and Variance Tradeoff

VARIANCE : Captures how much your classifier changes if you train on a different training set. How "over-specialized" is your classifier to a particular training set (overfitting)? If we have the best possible model for our training data, how far off are we from the average classifier?

BIAS : What is the inherent error that you obtain from your classifier even with infinite training data? This is due to your classifier being "biased" to a particular kind of solution (e.g. linear classifier). In other words, bias is inherent to your model.

NOISE : How big is the data-intrinsic noise? This error measures ambiguity due to your data distribution and feature representation. You can never beat this, it is an aspect of the data.

DECOMPOSITION OF TEST ERROR The error between the classifier predicted values and the given labels is represented as the sum of Bias, Variance and Intrinsic error present in the given data itself.

$$Testerror = Variance + Noise + Bias$$

So, to reduce the test error, we need to reduce both variance and bias. But it is really hard to reduce both at the same time because if one increases, then other increase. Therefore, we need to find that sweet spot between them.

HIGH VARIANCE Symptoms:

1. Training error is much lower than test error.

2. Training error is lower than ϵ.

3. Test error is above ϵ.

Remedies:

1. Add more training data.
2. Reduce model complexity – complex models are prone to high variance.
3. Bagging.

HIGH BIAS Symptoms: Training error is higher than ϵ.
Remedies:

1. Use more complex model (e.g. kernelize, use non-linear models).
2. Add features.
3. Boosting.

The link to the proof of Bias and Variance Tradeoff equation : `https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html`

2.12 DECISION TREES

MOTIVATION In case of KD trees, If you knew that a test point falls into a cluster of 1 million points with all positive label, you would know that its neighbors will be positive even before you compute the distances to each one of these million distances. It is therefore sufficient to simply know that the test point is an area where all neighbors are positive, its exact identity is irrelevant.

INTRODUCTION Decision trees are exploiting exactly that. Here, we do not store the training data, instead we use the training data to build a tree structure that recursively divides the space into regions with similar labels. The root node of the tree represents the entire data set. This set is then split roughly in half along one dimension by a simple threshold t. All points that have a feature value t fall into the right child node, all the others into the left child node. The threshold t and the dimension are chosen so that the resulting child nodes are purer in terms of class membership. Ideally all positive points fall into one child node and all negative points in the other. If this is the case, the tree is done. If not, the leaf nodes are again split until eventually all leaves are pure (i.e. all its data points contain the same label) or cannot be split any further (in the rare case with two identical points of different labels).

PARAMETRIC OR NON-PARAMETRIC Decision Trees are also an interesting case. If they are trained to full depth they are non-parametric, as the depth of a decision tree scales as a function of the training data (in practice O(log2(n))). If we however limit the tree depth by a maximum value they become parametric (as an upper bound of the model size is now known prior to observing the training data). We can also split on the same feature multiple times.

Advantages

1. Once the tree is constructed, the training data does not need to be stored. Instead, we can simply store how many points of each label ended up in each leaf - typically these are pure so we just have to store the label of all points;
2. Decision trees are very fast during test time, as test inputs simply need to traverse down the tree to a leaf - the prediction is the majority label of the leaf
3. Decision trees require no metric because the splits are based on feature thresholds and not distances.
4. Best for Interpretability.

Limitations

1. Splitting the tree until each and every point in the training set is correct because leads to overfitting.
2. **Decision boundaries are always parallel to the axes.**
3. To find out the best split, impurity function for all the possible splits in all the possible features have to be tried and consider the split with lowest impurity. I guess, it is very slow in case of continuous features.

Impurity Functions Data: $S = \{(x_1, y_1), \dots, (x_n, y_n)\}, y_i \in \{1, \dots, c\}$, where c is the number of classes.

1. Gini impurity Let $S_k \subset S$ where $S_k = \{(x, y) \in S : y = k\}$ (all inputs with labels k) $S = S_1 \cup S_2 \cup .....S_c$ Define:

   $$p_k = \frac{|S_k|}{|S|}$$

   – fraction of inputs in S with label k Gini Impurity :

   $$G(S) = \sum_{k=1}^{c} p_k(1 - p_k)$$

   Gini impurity of a tree:

   $$GT(S) = \frac{|S_L|}{|S|} GT(S_L) + \frac{|S_R|}{|S|} GT(S_R)$$

   where:

   $$S = S_L \cup S_R$$

   $$S_L \cap S_R = \phi$$

   $\frac{|S_L|}{|S|}$ – fraction of inputs in left substree
   $\frac{|S_R|}{|S|}$ – fraction of inputs in right substree

2. Entropy : Let p1,...,pk be defined as before. We know what we don't want (Uniform Distribution): $p_1 = p_2 = ... = p_c = \frac{1}{c}$ This is the worst case since each leaf is equally likely. Prediction is random guessing. Define the impurity as how close we are to uniform.

   Entropy :

$$H(S) = \sum_{k=1}^{C} p_k log(p_k)$$

$$H(S) = p_L H(S_L) + p_R H(S_R)$$

   $p_L = \frac{|S_L|}{|S|}, p_R = \frac{|S_R|}{|S|}$

ID3 ALGORITHM ID3 algorithm stop under two cases. The first case is that all the data points in a subset of have the same label. If this happens, we should stop splitting the subset and create a leaf with label y. The other case is there are no more attributes could be used to split the subset. Then we create a leaf and label it with the most common y or mean y in case of regression.

HOW TO SPLIT Try all features and all possible splits. Pick the split that minimizes impurity (e.g. s>t) where f←feature and t←threshold Shape of data : $(N \times d)$. Split between every two points in all the dimensions. Therefore, N-1 splits, d dimensions gives us $d \times (N-1)$ splits.

DRAWBACK Decision trees has a bias and variance problem and finding a sweet spot between them is hard. That is why in real life, they don't work very well. To solve the variance problem, we use bagging and can be used in any algorithm.

INCASE OF REGRESSION Incase of regression, we compute squared loss instead of gini and entropy. After each split we will compute variance on the left and right trees and split when the weighted variance is low. Stopping criteria would be either the limit set for the depth or split until every leaf contains one datapoint.

Average squared difference from average label

$$L(S) = \frac{1}{|S|} \sum_{(x,y)\in S} (y - \overline{y})^2$$

where $\overline{y_S} = \frac{1}{|S|} \sum_{(x,y)\in S} y$ That is why decision trees are also called as CART(Classification and Regression trees).

2.13 BAGGING

WHY To reduce Variance. This can used with all the algorithm if there exists the variance problem.

Introduction

$$Variance = E[h_D(x) - \overline{h(x)}]^2$$

where $\overline{h(x)}$ is the expected classifier that is average of all the classifiers trained on different datasets drawn from same distribution and $h_D(x)$ is the classifier trained on a single dataset.

1. To reduce the variance, we need to reduce difference between and $h_D(x)$ and $\overline{h(x)}$.
2. The weak law of large numbers says (roughly) for i.i.d. random variables xi with mean $\overline{x}$, we have,
   $\frac{1}{m}\sum_{i=1}^{m} x_i \to \overline{x}$ as $m \to \infty$.
3. Apply this to classifiers: Assume we have m training sets D1,D2,...,Dm drawn from P. Train a classifier on each one and average result:
   $\widehat{h} = \frac{1}{m}\sum_{i=1}^{m} h_{D_i} \to \overline{h}$ as $m \to \infty$
4. We refer to such an average of multiple classifiers as an ensemble of classifiers.
5. **Good news**: If $\widehat{h} \to \overline{h}$ the variance component of the error must also vanish.
6. **Problem:** We don't have m data sets D1,....,Dm, we only have D.

Algorithm

1. Sample m data sets D1,. . . ,Dm from D **with replacement**.
2. For each Dj train a classifier hj(x)
3. The final classifier is
   $$h(x) = \frac{1}{m}\sum_{j=1}^{m} h_j(x)$$
   In practice larger m results in a better ensemble, however at some point you will obtain diminishing returns. Note that setting m unnecessarily high will only slow down your classifier but will not increase the error of your classifier.

Advantages

1. Easy to implement
2. Reduces variance.
3. As the prediction is an average of many classifiers, we obtain a mean score and variance. Latter can be interpreted as the uncertainty of the prediction.
4. No need to split the train set into train and val further. The idea is that each training point was not picked and all the data sets Dk. If we average the classifiers hk of all such data sets, we obtain a classifier (with a slightly smaller m) that was not trained on (xi,yi) ever and it is therefore equivalent to a test sample. If we compute the error of all these classifiers**(which doesn't contain that particular point)**, we obtain an estimate of the true test error. The beauty is that we can do this without reducing the training set. We just run bagging as it is intended and obtain this so called out-of-bag error for free.

Disadvantages

1. Leads to Overfitting.

2.14 Random Forest

A Random Forest is essentially nothing else but bagged decision trees, with a slightly modified splitting criteria.

Algorithm

1. The algorithm works as follows: Sample m data sets D1,...,Dm from D **with replacement**.

2. For each D_j train a full decision tree $h_j()$ ($depth_{max} = \infty$) with one small modification: before each split randomly subsample $k \leq d$ features (**without replacement**) and only consider these for your split. (This further increases the variance of the trees.)

3. The final classifier is
$$h(x) = \frac{1}{m}\sum_{j=1}^{m} h_j(x)$$

Advantages

1. The RF only has two hyper-parameters, m and k. It is extremely insensitive to both of these. A good choice for k is $k = \sqrt{d}$ (where d denotes the number of features). You can set m as large as you can afford.

2. Decision trees do not require a lot of preprocessing. For example, the features can be of different scale, magnitude, or slope. This can be highly advantageous in scenarios with heterogeneous data, for example the medical settings where features could be things like blood pressure, age, gender, ..., each of which is recorded in completely different units.

Why would we sample features without replacement ? At each node, features are sampled without replacement. Sampling with replacement would **effectively** reduce the number of features sampled at each split, because the best split among some feature is the same for that feature sampled a second time.

Across the entire tree, you might choose the same feature more than once. This is because information about previous splits in no way informs how subsequent splits are chosen. Splitting multiple times on the same feature can be necessary to model relationships which are not step functions with a single step. And, why would you split on the same feature twice?

2.15 Boosting

Intuition Just as humans learn from their mistakes and try not to repeat them further in life, the Boosting algorithm tries to build a strong learner (predictive model) from the mistakes of several weaker models (Weak learner : Classifier better than random guess i.e. may be coin tossing). You start by creating a model from the training data. Then, you create a second model from the previous one by trying to reduce the errors from the previous model. Models are added sequentially, each correcting its predecessor, until the training data is predicted perfectly or the maximum number of models have been added.

Boosting basically tries to reduce the bias error which arises when models are not able to identify relevant trends in the data. This happens by evaluating the difference between the predicted value and the actual value.

Types of Boosting Algorithms

1. AdaBoost (Adaptive Boosting)
2. Gradient Tree Boosting
3. XGBoost

Maths behind Boosting Create ensemble classifier $H_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$. This ensemble classifier is built in an iterative fashion. In iteration t, we add the classifier $\alpha_t h_t(x)$ to the ensemble. At test time we evaluate all classifier and return the weighted sum.

The process of constructing such an ensemble in a stage-wise fashion is very similar to gradient descent. However, instead of updating the model parameters in each iteration, we add functions to our ensemble. Let l denote a (convex and differentiable) loss function. With a little abuse of notation we write

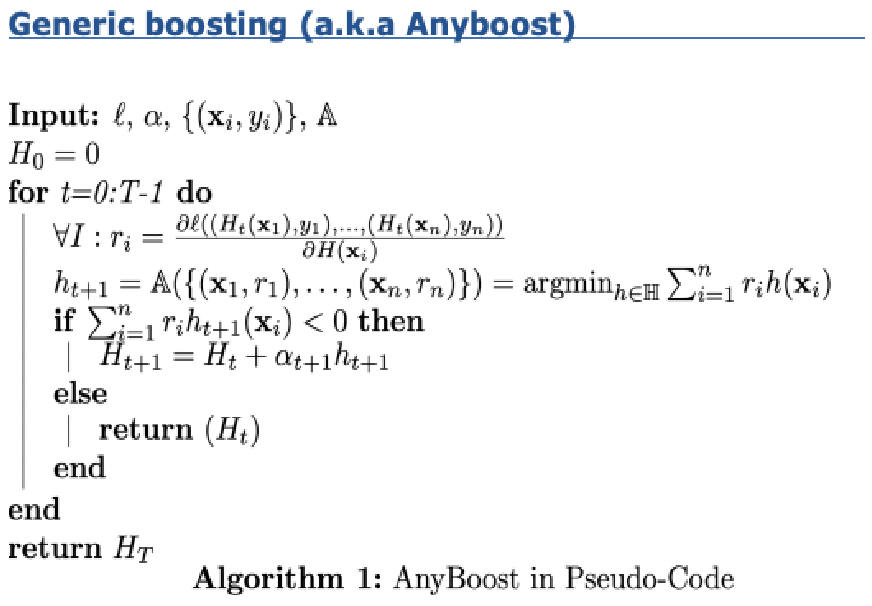
$$l(H) = \frac{1}{n}\sum_{t=1}^{n} \ln l(H(x_i), y_i)$$

Assume we have already finished t iterations and already have an ensemble classifier $h_t(x)$). Now in iteration t+1 we want to add one more weak learner h_{t+1} to the ensemble. To this end we search for the weak learner that minimizes the loss the most, $h_{t+1} = argmin_{h\in H} l(H_t + \alpha h_t)$ Once h_{t+1} has been found, we add it to our ensemble, i.e.

$$H_{t+1} := H_t + \alpha h$$

How can we find such h Answer: Use gradient descent in function space. In function space, inner product can be defined as $< h, g >= \sum_{t=1}^{n} h(x_i) g(x_i)$. Check the pseduo code 2.6

My comments Generally, in case of gradient descent, we move in the opposite direction of gradient $\partial L/\partial H(\theta)$. Similarly, here we are training a extra model classifier whose predictions point the direction opposite to the gradient $(\partial l(H)/\partial H_t)$, so that, the loss decreases and Overall predictions get closer the original (ground truth) values.



Generic boosting (a.k.a Anyboost)

```
Input: $\ell$, $\alpha$, $\{(\mathbf{x}_i, y_i)\}$, $\mathbb{A}$
$H_0 = 0$
for $t$=0:T-1 do
    $\forall I : r_i = \frac{\partial \ell((H_t(\mathbf{x}_1), y_1), \ldots, (H_t(\mathbf{x}_n), y_n))}{\partial H(\mathbf{x}_i)}$
    $h_{t+1} = \mathbb{A}(\{(\mathbf{x}_1, r_1), \ldots, (\mathbf{x}_n, r_n)\}) = \text{argmin}_{h \in \mathbb{H}} \sum_{i=1}^{n} r_i h(\mathbf{x}_i)$
    if $\sum_{i=1}^{n} r_i h_{t+1}(\mathbf{x}_i) < 0$ then
        $H_{t+1} = H_t + \alpha_{t+1} h_{t+1}$
    else
        return $(H_t)$
    end
end
return $H_T$
```

Algorithm 1: AnyBoost in Pseudo-Code

Figure 2.6: Pseudo code of Anyboosting algorithm

2.16 Gradient Boosted Regression Tree(GBRT)

In order to use regression trees for gradient boosting, we must be able to find a tree h() that maximizes $h = argmin_{h \in H} \sum_{i=1}^{n} r_i.h(x_i)$ where $r_i = \partial l / \partial H(x_i)$.

If the loss function l is the squared loss, i.e.

$$l(H) = \frac{1}{2} \sum_{i=1}^{n} (H(x_i) y_i)^2$$

, then it is easy to show that

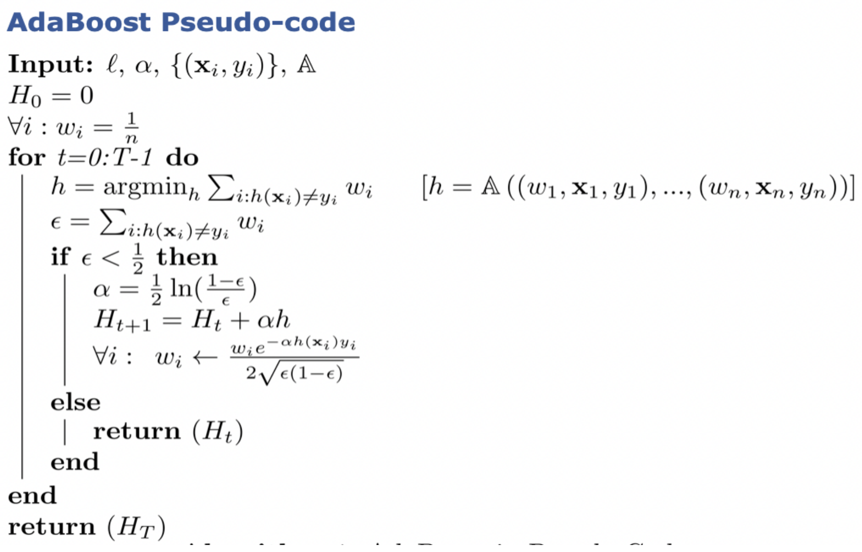
$$t_i = -\partial l / \partial H(x_i) = y_i - H(x_i)$$

which is simply the residual, i.e. r is the vector pointing from y to H. However, it is important that you can use any other differentiable and convex loss function l and the solution for your next weak learner h() will always be the regression tree minimizing the squared loss.

My comments In case of regression task and mean squared error, $\frac{\nabla l(H)}{\nabla H_t} = H_t(x_i) - y_i$. We are training a regressor which minimizes the product $(H_t(x_i) - y_i).h$. Hence, we are finding the direction of $H_t(x_i) - y_i$ and moving opposite to it (closer to y_i) which is what we want.

AdaBoost

Introduction More weight is assigned to the incorrectly classified samples so that they're classified correctly in the next decision stump. Weight is also assigned to each classifier based on the accuracy of the classifier, which means high accuracy = high weight!



AdaBoost Pseudo-code

```
Input: $\ell$, $\alpha$, $\{(\mathbf{x}_i, y_i)\}$, $\mathbb{A}$
$H_0 = 0$
$\forall i : w_i = \frac{1}{n}$
for $t$=0:T-1 do
    $h = \text{argmin}_h \sum_{i:h(\mathbf{x}_i)\neq y_i} w_i$    $[h = \mathbb{A}((w_1, \mathbf{x}_1, y_1), ..., (w_n, \mathbf{x}_n, y_n))]$
    $\epsilon = \sum_{i:h(\mathbf{x}_i)\neq y_i} w_i$
    if $\epsilon < \frac{1}{2}$ then
        $\alpha = \frac{1}{2}\ln(\frac{1-\epsilon}{\epsilon})$
        $H_{t+1} = H_t + \alpha h$
        $\forall i : \quad w_i \leftarrow \frac{w_i e^{-\alpha h(\mathbf{x}_i) y_i}}{2\sqrt{\epsilon(1-\epsilon)}}$
    else
        return $(H_t)$
    end
end
return $(H_T)$
```

Figure 2.7: Ada Boost Algorithm pseudo code

1. Classification : $y_i \in \{+1, 1\}$
2. Weak learners: $h \in H$ are binary, $h(x_i) \in \{1, +1\}$, $\forall x$.
3. Step-size : We perform line-search to obtain best step-size α.
4. Loss function: Exponential loss $l(H) = \sum_{i=1}^{n} e^{-y_i H(x_i)}$.

STEPS 2.7

1. First we compute the gradient $r_i = \partial l / \partial H(x_i) = -y_i e^{-y_i . H(x_i)}$.

   For notational convenience (and for reason that will become clear in a little bit), let us define $w_i = \frac{1}{Z} e^{-y_i . H(x_i)}$, where $Z = \sum_{i=1}^{n} e^{-y_i . H(x_i)}$ is a normalizing factor so that $\sum_{i=1}^{n} w_i = 1$.

   Note that the normalizing constant Z is identical to the loss function. Each weight w_i therefore has a very nice interpretation. It is the relative contribution of the training point (x_i, y_i) towards the overall loss. Let us denote this weighted classification error as $\epsilon = \sum_{i:h(x_i) y_i = 1} w_i$. So for AdaBoost, we only need a classifier that can take training data and a distribution over the training set (i.e. normalized weights w_i for all training samples) and which returns a classifier $h \in H$ that reduces the weighted classification error of these training samples.

2. When we are given l, H, h, we would like to solve the following optimization problem:

$$\alpha = argmin_{\alpha} l(H + \alpha h) = argmin_{\alpha} \sum_{i=1}^{n} e^{-y_i [H(x_i) + \alpha h(x_i)]} \tag{2.2}$$

Taking the derivative of the above equation and equating to zero gives us the value of optimal α.

3. Optimal α is found to be
$$\alpha = \frac{1}{2}\ln\frac{1-\epsilon}{\epsilon}$$
.

4. After you take a step, i.e. $H_{t+1} = H_t + \alpha h$, you need to re-compute all the weights and then re-normalize. The update rule would be
$$w_i := w_i.\frac{e^{-\alpha h(x_i).y_i}}{2\epsilon(1-\epsilon)}$$

3 Terminology

Parametric vs Non-parametric algorithms

1. A parametric algorithm is one that has a constant set of parameters, which is **independent** of the number of training samples. You can think of it as the amount of much space you need to store the trained classifier. An examples for a parametric algorithm is the Perceptron algorithm, or logistic regression. Their parameters consist of w,b, which define the separating hyperplane. The dimension of w depends on the dimension of the training data, but not on how many training samples you use for training.

2. In contrast, the number of parameters of a non-parametric algorithm scales as a function of the training samples. An example of a non-parametric algorithm is the k-Nearest Neighbors classifier. Here, during "training" we store the entire training data – so the parameters that we learn are identical to the training set and the number of parameters (the storage we require) grows linearly with the training set size.

4 Unsupervised Machine Learning Algorithms

4.1 Introduction to Clustering

Clustering algorithms are used to group data points based on certain similarities. There's no criterion for good clustering. Clustering determines the grouping with unlabelled data. It mainly depends on the specific user and the scenario. Clusters found by one clustering algorithm will definitely be different from clusters found by a different algorithm. Typical cluster models include:

1. Connectivity models – like hierarchical clustering, which builds models based on distance connectivity.

2. Centroid models – like K-Means clustering, which represents each cluster with a single mean vector.

3. Distribution models – here, clusters are modeled using statistical distributions.
4. Density models – like DBSCAN and OPTICS, which define clustering as a connected dense region in data space.
5. Group models – these models don't provide refined results. They only offer grouping information.
6. Graph-based models – a subset of nodes in the graph such that an edge connects every two nodes in the subset can be considered as a prototypical form of cluster.
7. Neural models – self-organizing maps are one of the most commonly known Unsupervised Neural networks (NN), and they're characterized as similar to one or more models above

4.2 Agglomerative hierarchical clustering (AHC)

Introduction

1. The Agglomerative Hierarchical Cluster Algorithm is a form of bottom-up clustering, where it starts with an individual element and then groups them into single clusters.
2. Hierarchical clustering is often used in the form of descriptive modeling rather than predictive. It doesn't work well on large datasets, it provides the best results in some cases only.

Algorithm

1. Each data point is treated as a single cluster. We have K clusters in the beginning. At the start, the number of data points will also be K.
2. Now we need to form a big cluster by joining 2 closest data points in this step. This will lead to total K-1 clusters.
3. Two closest clusters need to be joined now to form more clusters. This will result in K-2 clusters in total.
4. Repeat the above three steps until K becomes 0 to form one big cluster. No more data points are left to join.
5. After forming one big cluster at last, we can use dendrograms to split the clusters into multiple clusters depending on the use case.

Advantages

1. AHC is easy to implement, it can also provide object ordering, which can be informative for the display.

2. We don't have to pre-specify the number of clusters. It's easy to decide the number of clusters by cutting the dendrogram at the specific level.
3. In the AHC approach smaller clusters will be created, which may uncover similarities in data.

DISADVANTAGES

1. The objects which are grouped wrongly in any steps in the beginning can't be undone.
2. Hierarchical clustering algorithms don't provide unique partitioning of the dataset, but they give a hierarchy from which clusters can be chosen.
3. They don't handle outliers well. Whenever outliers are found, they will end up as a new cluster, or sometimes result in merging with other clusters.

4.3 K-MEANS CLUSTERING ALGORITHM

Source : https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm

INTRO

1. Group similar data points together and discover underlying patterns.
2. The K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

A cluster refers to a collection of data points aggregated together because of certain similarities.

HOW IT WORKS

1. To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids
2. It halts creating and optimizing clusters when either:
   a) The centroids have stabilized — there is no significant change in their values because the clustering has been successful.
   b) The defined number of iterations has been achieved.

ALGORITHM Let $X = x1, x2, x3, \ldots\ldots.., xn$ be the set of data points and $V = v1, v2, \ldots\ldots., vc$ be the set of centers.

1. Randomly select 'c' cluster centers.
2. Calculate the distance between each data point and cluster centers.
3. Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.
4. Recalculate the new cluster center using:

   $$\mu_i = \sum_{j=1}^{c_i} x_j$$

   where, 'c_i' represents the number of data points in i_{th} cluster.
5. Recalculate the distance between each data point and new obtained cluster centers.
6. If no data point was reassigned then stop, otherwise repeat from step 3).

OBJECTIVE

1. To minimize

   $$J = \sum_{i=1}^{k} \sum_{j=1}^{c_i} ||x_j - \mu_i||^2$$

2. The number of optimal clusters are decided based on the above objective function (Elbow method) or any other metrics.
3. Elbow Method : When Loss vs K graph is plotted, Usually it looks like an elbow where sudden huge decrease in the loss happens. That point is considered to be elbow and K at the point is considered to be optimal.
4. As K increases, loss always decreases. Therefore, we consider elbow point.
5. Difficult to find this point most of the times. If failed to find the optimal point by measuring other metrics at each value of K.

LIMITATIONS

1. K-Means clustering is good at capturing the structure of the data if the clusters have a spherical-like shape. It always tries to construct a nice spherical shape around the centroid. This means that the minute the clusters have different geometric shapes, K-Means does a poor job clustering the data.
2. This algorithm highly depends on the initialization.

Points to remember

1. Since K-Means use a distance-based measure to find the similarity between data points, it's good to standardize the data to have a standard deviation of one and a mean of zero.
2. The elbow method used to select the number of clusters doesn't work well as the error function decreases for all Ks.
3. If there's overlap between clusters, K-Means doesn't have an intrinsic measure for uncertainty for the examples belonging to the overlapping region to determine which cluster to assign each data point.
4. K-Means clusters data even if it can't be clustered, such as data that comes from uniform distributions.

4.4 Mini-Batch K-Means

An efficient version of K-Means. To reduce the complexity of K-Means, Mini batch K-Means is proposed. Pseudo code of the algorithm can be found in fig **??**.
Convergence of MiniBatch Kmeans is not clear.

4.5 Gaussian mixture model

Introduction

1. GMM can be used to find clusters in the same way as K-Means. The probability that a point belongs to the distribution's center decreases as the distance from the distribution center increases. The bands show a decrease in probability in the below image.
2. Since GMM contains a probabilistic model under the hood, we can also find the probabilistic cluster assignment.
3. When you don't know the type of distribution in data, you should use a different algorithm.

Parameters A GM is a function composed of several Gaussians, each identified by $K \in 1, \ldots, K$, where K is the number of clusters. Each Gaussian K in the mixture is comprised of following parameters:

1. A mean μ that defines its center.
2. A covariance $\sum$ that defines its width.
3. A mixing probability that defines how big or small or big the Gaussian function will be.

Algorithm The mean and variance for each gaussian is determined using a technique called Expectation-Maximization (EM).

Algorithm 1: Mini-batch *k*-means Clustering

```
Input:    data set $D = \{x_1, x_2, \dots, x_n\}$; cluster number $k$; mini-batch size $b$; iterations $t$
Output:   clusters $C = \{C_1, C_2, \dots, C_k\}$
Process:
 1: Initialize $k$ cluster centers $\mu = \{\mu_{C_1}, \mu_{C_2}, \dots, \mu_{C_k}\}$ with $k$ samples randomly picked from $D$
 2: $C_i \leftarrow \emptyset \ (1 \le i \le k)$  # initialize clusters
 3: $N_{C_i} \leftarrow 0 \ (1 \le i \le k)$ # initialize sample number for each cluster
 4: for $j = 1, 2, \dots, t$ do
 5:    $M \leftarrow \{x_m | 1 \le m \le b\}$  # M is the batch data set, and $x_m$ is the sample randomly picked from D
 6:    for $m = 1, 2, \dots, b$ do  # step 6 ~ step 8 are to catch cluster center for each sample in the batch set
 7:       $\mu_{C_i}(x_m) \leftarrow \frac{1}{|C_i|}\sum_{x_m \in C_i} x_m \ (x_m \in M)$  # $\mu(x_m)$ is the cached cluster center nearest to data sample $x_m$
 8:    end for
 9:    for $m = 1, 2, \dots, b$ do  # step 9 ~ step 14 are to update the cluster centers with each batch set
10:       $\mu_{C_i} \leftarrow \mu_{C_i}(x_m)$  # get the cached center for $x_m$
11:       $N_{C_i} \leftarrow N_{C_i} + 1$  # update sample number for each cluster center
12:       $\eta \leftarrow 1/N_{C_i}$  # calculate learning rate for each cluster center
13:       $\mu_{C_i} \leftarrow (1 - \eta)\mu_{C_i} + \eta x_m$  # take gradient step to update cluster center
14:    end for
15: end for
```

Figure 4.1:

Expectation-Maximization in Gaussian Mixture Models

1. We typically use EM when the data has missing values, or in other words, when the data is incomplete.

2. Expectation : For each point x_i, calculate the probability that it belongs to cluster/distribution c1, c2, … ck. This is done using the below formula:

$$r_{ic} = \frac{P(x_i \in c_i)}{\sum P(x_i \in c_1)P(x_i \in c_2)....}$$

   This value will be high when the point is assigned to the right cluster and lower otherwise.

3. The mean and the covariance matrix are updated based on the values assigned to the distribution, in proportion with the probability values for the data point. Hence, a data point that has a higher probability of being a part of that distribution will contribute a larger portion:

$$\mu_c = \frac{1}{N_c} \sum_i r_{ic} x_i$$

$$\sum_c = \frac{1}{N_c} \sum_i r_{ic} (x_i - \mu_c)^T (x_i - \mu)$$

Advantages

1. One of the advantages of GMM over K-Means is that K-Means doesn't account for variance (here, variance refers to the width of the bell-shaped curve) and GMM returns the probability that data points belong to each of K clusters.

2. In case of overlapped clusters, all the above clustering algorithms fail to identify it as one cluster.

3. GMM uses a probabilistic approach and provides probability for each data point that belongs to the clusters.

Disadvantages

1. Mixture models are computationally expensive if the number of distributions is large or the dataset contains less observed data points.

2. It needs large datasets and it's hard to estimate the number of clusters.

4.6 DBSCAN clustering

Video : `https://www.youtube.com/watch?v=C3r7tGRe2eI&t=714s`

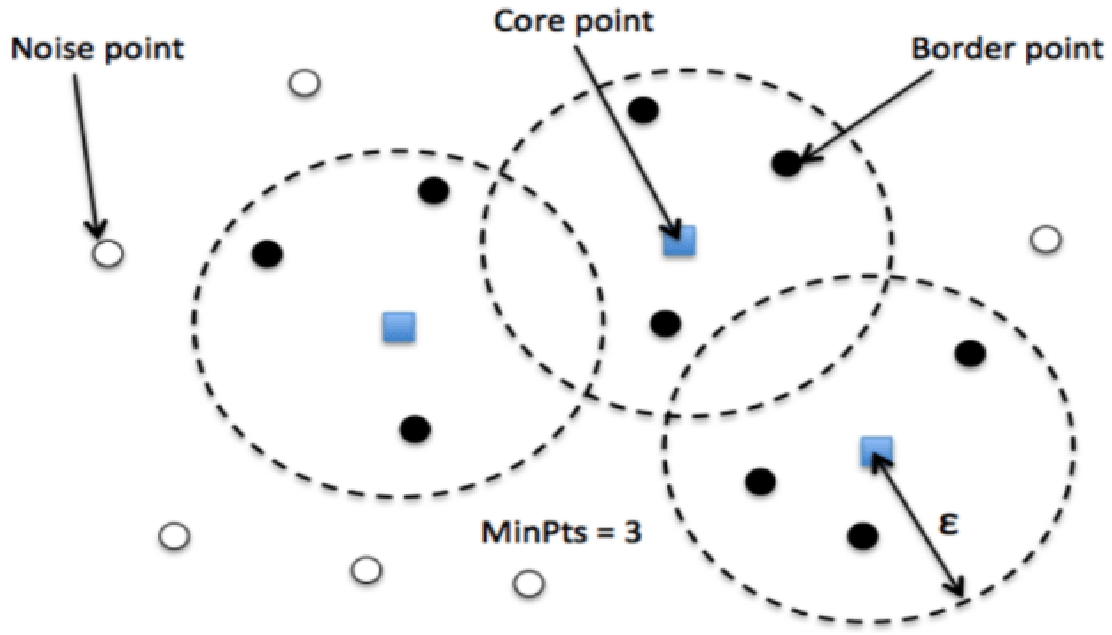


Figure 4.2: Core is a point that has some (m) points within a particular (n) distance from itself.
The border is a point that has at least one core point at distance n.
Noise is a point that is neither border nor core.

INTRODUCTION

1. This type of clustering technique connects data points that satisfy particular density criteria (minimum number of objects within a radius).
2. There are three types of points: core, border, and noise. See Fig 4.2.

PARAMETERS

1. minPts: for a region to be considered dense, the minimum number of points required is ‘minPts‘.
2. eps: to locate data points in the neighborhood of any points, eps() is used as a distance measure.

ALGORITHM Better to watch the above video than reading the text.

LIMITATIONS It expects some kind of density drop to detect cluster borders. DBSCAN connects areas of high example density. The algorithm is better than K-Means when it comes to oddly shaped data.

ADVANTAGES

1. It doesn't require a predefined number of clusters.
2. It also identifies noise and outliers. Furthermore, arbitrarily sized and shaped clusters are found pretty well by the algorithm.

4.7 METRICS

SILHOUETTE SCORE

1. Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually. This measure has a range of [-1, 1].
2. A value of +1 indicates sample is far way from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.
3. The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.
4. Formula
$$silhouettescore(s) = \frac{b-a}{max(a,b)}$$
where a: The mean distance between a sample and all other points in the same class. b: The mean distance between a sample and all other points in the next nearest cluster.
5. Limitations : The Silhouette Coefficient is generally higher for convex clusters than other concepts of clusters, such as density based clusters like those obtained through DBSCAN.

DAVIES BOULDIN INDEX (DBI) The index is defined as the average similarity between each cluster C_i for i = 1,2,.. ,k and its most similar one C_j. In the context of this index, similarity is defined as a measure R_{ij} that trades off:

1. si, the average distance between each point of cluster i and the centroid of that cluster – also know as cluster diameter.
2. d_{ij}, the distance between cluster centroids i and j.

A simple choice to construct R_{ij} so that it is nonnegative and symmetric is:

$$R_{ij} = \frac{s_i + s_j}{d_{ij}}$$

Then the Davies-Bouldin index is defined as:

$$DB = \sum_{i=1}^{k} max_{i \neq j} R_i$$

5 Anomaly detection Techniques

5.1 Isolation Forest(BASIC INTRO)

1. Isolation forest is a machine learning algorithm for anomaly detection.
2. It's an unsupervised learning algorithm that identifies anomaly by isolating outliers in the data.
3. Isolation Forest is based on the Decision Tree algorithm. It isolates the outliers by randomly selecting a feature from the given set of features and then randomly selecting a split value between the max and min values of that feature. This random partitioning of features will produce shorter paths in trees for the anomalous data points, thus distinguishing them from the rest of the data.
4. Isolation Forest isolates anomalies in the data points instead of profiling normal data points. As anomalies data points mostly have a lot shorter tree paths than the normal data points, trees in the isolation forest does not need to have a large depth so a smaller maxdepth can be used resulting in low memory requirement.
5. Using Isolation Forest, we can detect anomalies faster and also require less memory compared to other algorithms.
6. We can use sklearn.ensemble.IsolationForest

Part II
Practical

6 Feature Engineering

Introduction

Why Feature engineering fulfils mainly two goals:

1. It prepares the input dataset in the form which is required for a specific model or machine learning algorithm.
2. Feature engineering helps in improving the performance of machine learning models magically.