Machine Intelligence (UE18CS303)
Unit 2

Aronya Baksy

September 2020

1 K-Nearest Neighbours Classification

1.1 Instance Based Learning

- The K-nearest neighbours algorithm is an **instance-based** learning algorithm
- In an instance-based algorithm the training phase consists only of storing the data
- When a query with some previously unseen data point is made to the algorithm, the most similar instances are used to classify the unseen query instance.

1.2 K-Nearest Neighbours Algorithm

- The KNN algorithm considers each training instance to be represented as a single point in an n-dimensional Euclidean space $\mathbb{R}^n$
- An arbitrary instance x can be represented using the feature vector $\langle a_1(x), a_2(x), a_3(x), ..., a_n(x)\rangle$ where $a_r(x)$ denotes the value of the r^{th} attribute of point x.
- The distance measure used to compute the distance between two examples x_i and x_j is given by the Euclidean Distance Formula

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2} \tag{1}$$

- The target function to be learnt is denoted as $f(x)$ and it takes a discrete set of values V (which are the classes of the classification algorithm).
- Given an unseen query instance x_q, the algorithm assigns the **most common class** from among the k **closest training examples** to the query instance.

Algorithm 1 K-Nearest Neighbours

procedure KNN_Train($\langle x, f(x)\rangle$)
  Add the example $\langle x, f(x)\rangle$ to the list of examples
end procedure

procedure KNN_Classify(x_q: Query Instance)
  $x_1, x_2, x_3, ..., x_k \leftarrow$ The k closest train examples to x_q
  $f(x_q) \leftarrow \underset{v \in V}{\text{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$
  return $f(x_q)$
end procedure

- Here, $\delta(a, b) = 1$ if $a == b$ else $\delta(a, b) = 0$

1.3 Distance Weighted K-nearest Neighbours

- The contribution of the k-nearest neighbours to the target function value for the query can be weighted by distance
- The contribution of farther points (points with greater distance from x_q) is less than the closer points (points with smaller distance from x_q).
- The formula to count the frequency of the classes of the k neighbours is modified as

$$f(x_q) = \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \frac{1}{d(x_q, x_i)^2} \delta(v, f(x_i)) \tag{2}$$

1.4 Additional Remarks on KNN

- KNN Classification is notable in that it suffers from **the curse of dimensionality**, which is a term used to refer to the presence of a large number of irrelelvant attributes that distort the actual distance measure between 2 examples.
- For example, if all examples consist of 20 attributes but only 2 attributes are needed for classification, the remaining 18 attributes dominate the distance calculation (as KNN uses all the attributes while calculating distance) and can cause misleading results.
- A simple approach against this is to weight each attribute differently, such that the significant attributes get more weight in the distance calculation and the less significant ones get less weight.
- Another approach against the curse of dimensionality is to eliminate redundant attributes.
- The other significant concern is the indexing and storage of large number of training examples that need to be processed every time a new query is made. Efficient data structures like *kd-trees* are used for indexing.
- In a kd-tree, instances are stored at the leaves of a tree, with nearby instances stored at the same or nearby nodes. The internal nodes of the tree sort the new query x_q, to the relevant leaf by testing selected attributes of x_q.

2 Artificial Neural Networks (ANN)

- ANNs are models that are used to approximate discrete-valued, real-valued or vector-valued functions.
- ANNs provide a general, practical and robust approach to learn such functions, and they can be modified to be used in a wide variety of contexts (eg: Convolutional NNs for image data, Recurrent NNs for sequential data like text/audio/video)

2.1 Biological Motivation behind ANN

- ANN architecture is loosely based on the structure of the human brain, which is a complicated network of many interconnected neurons.
- ANNs consist of a densely connected set of simple units, where each unit takes in a single real-valued input, and each gives out a single real-valued output that can serve as further input to one or more units.
- Some complexities of biological neural nets are ignored in ANNs.
- For example, while an ANN unit outputs a single real value, biological neurons output a complex time series of electrical spikes.

2.2 Appropriate Problems for ANNs

Below are the characteristics for problems that are appropriate to be solved by neural networks.

- Instances represented as many attribute-value pairs
- The target function may be real valued, discrete valued or a vector of real/discrete values
- Train examples may contain errors
- Long train times are acceptable
- Fast evaluation of learned target value
- Human readability of resulting weights is not important

2.3 The Perceptron

- The perceptron is the basic building block of an ANN.
- It takes in inputs $x_1, x_2, ..., x_n$, calculates a weighted sum of these inputs. The input $x_0 = 1$ is considered a dummy input to make the representation easier.
- If the weighted sum is larger than some threshold, the perceptron outputs a +1, else it outputs a -1.
- Let

$$z = w_0x_0 + w_1x_1 + w_2x_2 + ... + w_nx_n = \sum_{i=1}^{n} w_ix_i$$

be the weighted sum of the inputs $x_1, x_2, ..., x_n$. Then the output o is given as

$$o = \begin{cases} +1 \text{ if } a > 0 \\ -1 \text{ if } a \leq 0 \end{cases} \tag{3}$$

- The constant w_0 is called the **bias** of the perceptron, and the negative of the bias indicates the amount that the weighted sum of the inputs must surpass.
- The bias term is added to make sure that the decision hyperplane represented by the weighted sum of inputs does not always pass through the origin, which increases the capability of the model to represent more real-word scenarios.

2.4 Representation of functions with Perceptrons

- A single perceptron can be used to represent any linearly separable function that takes binary values.
- For example, Boolean functions except XOR and XNOR are all linearly separable and can be represented using a single perceptron

Function	w_0	w_1	w_2
AND	-0.8	0.5	0.5
NAND	-0.8	0.5	0.5
OR	-0.3	0.5	0.5
NOR	-0.3	0.5	0.5
NOT (single IP)	1	-2	N/A

- The above table details the weights for some linearly separable Boolean functions of two inputs.
- To represent XOR and XNOR functions, two neurons connected in sequence are required.

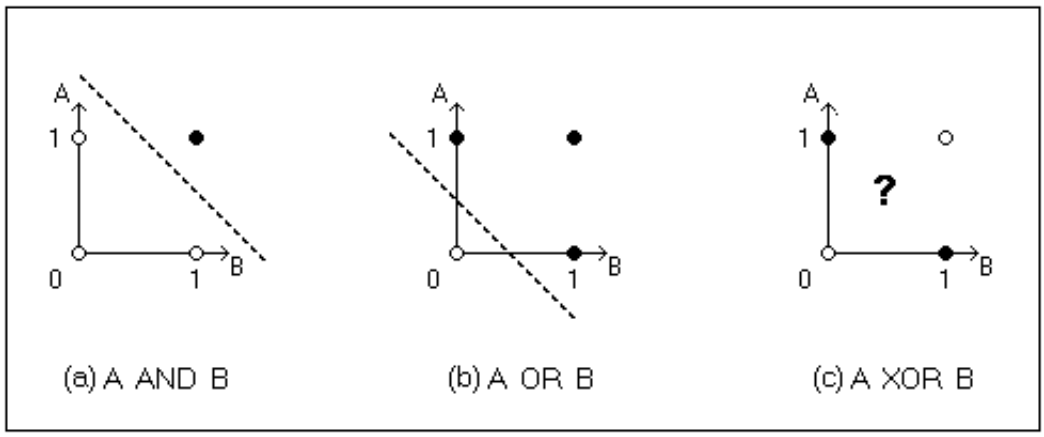

Figure 1: Linearly Separable and non-Linearly Separable Boolean Functions

2.5 Perceptron Training Rule

- The most common training algorithm for training of perceptrons is the **gradient descent** rule, where the weights are adjusted in an iterative manner to prevent misclassification
- This method requires the definition of an **error function** (also known as a **cost function**) that denotes the error between the learned label and the actual label classified. One commonly used cost function is the mean squared error.

$$E(\overrightarrow{w}) = \frac{1}{2}\sum_{i=1}^{n}(t_i - o_i)^2 \tag{4}$$

Where t_i is the *target label* for the i^{th} example in the training data, and o_i is the *output label* outputted by the perceptron unit for the i^{th} example in the training data.

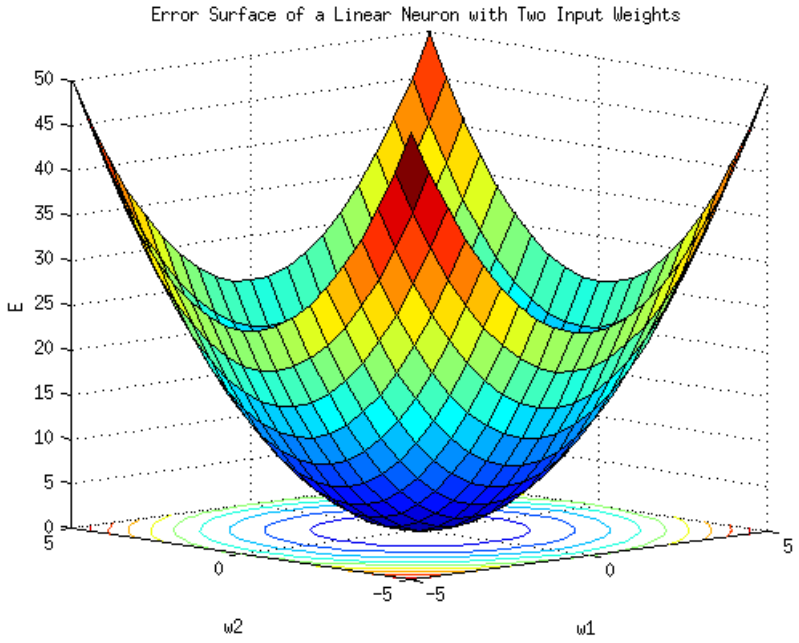

Figure 2: Error Surface for two weight perceptron

- The gradient descent rule starts with an arbitrary vector of weights $(w_1, w_2, w_3, ..., w_n)$, and iteratively modifying this vector in small steps.

- The modification is done such that the vector moves towards the **global minimum** of the cost function. Hence the modification must be done in the direction of *steepest descent* of the error surface.
- The direction of the steepest descent is given by the **negative of the gradient** of the error function.
- The gradient of the error function is given as

$$\nabla E = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_n}\right]$$

- Therefore, the training rule for gradient descent is given as

$$w_i = w_i - \alpha \nabla E(w_i) \tag{5}$$

- This implies that each component w_i of the weight vector $\overrightarrow{w}$ is modified in proportion to the partial derivative with respect to that component $\frac{\partial E}{\partial w_i}$
- The constant α is known as the *learning rate*. It determines the speed at which the descent along the error surface takes place.
- It is important to find an optimal value for the hyperparameter α because too large a value may lead to overshooting the global minimum, but too small a value may lead to increased training time.
- The gradient with respect to the kth weight w_k can be calculated as

$$\begin{aligned}
\frac{\partial E}{\partial w_k} &= \frac{\partial}{\partial w_k}\frac{1}{2}\sum_{i=1}^{n}(t_i - o_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{n}\frac{\partial}{\partial w_k}(t_i - o_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{n}2(t_i - o_i)\frac{\partial}{\partial w_k}(t_i - o_i) \\
&= \frac{1}{2}\sum_{i=1}^{n}2(t_i - o_i)\frac{\partial}{\partial w_k}(t_i - \overrightarrow{w}.\overrightarrow{x}) \\
&= \sum_{i=1}^{n}(t_i - o_i)(-x_{ik})
\end{aligned}$$

- x_{ik} represents the value of the k^{th} attribute of the i^{th} training example.

2.6 Stochastic Gradient Descent (SGD)

- The gradient descent algorithm, as presented above, involves computing the cost function for **all the training examples at once** then and doing the weight update .
- This is not only computationally expensive, but also does not guarantee that a global minimum will be reached in case the cost function has several local minima.
- In *stochastic gradient descent*, the weights are updated **for each training example at a time**.
- The key differences between standard Gradient Descent and Stochastic Gradient Descent are:
  - In standard gradient descent, the error is summed over all examples before updating weights, whereas in stochastic gradient descent weights are updated upon examining each training example.

- Stochastic Gradient Descent is capable of avoiding local minima, while Standard gradient descent is more susceptible to falling into a local minimum.
- Summing over multiple examples in standard gradient descent requires more computation per weight update step than Stochastic Gradient Descent.
- Because it uses the true gradient, standard gradient descent is often used with a larger step size per weight update (also known as the learning rate α) than stochastic gradient descent.

3 Multi-Layer ANNs

- Multilayer networks in conjunction with non-linear **activation functions** are capable of learning a large variety of non-linear decision boundaries.
- This make such networks more suited to a wider variety of tasks than single perceptron units.
- The **backpropagation** algorithm is used to train multi-layer ANNs.

3.1 Activation Functions

- Neurons that use he linear threshold function described in the above section is only capable of learning linear decision boundaries
- In order to learn more complex decision boundaries, a differentiable yet non-linear threshold is required. This non-linear threshold function is known as the activation function.
- Let the activation function be described as $\sigma(x)$. Then the computation of the neuron is described as

$$z = \sum_{i=0}^{n} w_i x_i \text{ where } x_0 = 1$$

and the output a of the neuron is written as

$$a = \sigma(z)$$

- Some popular activation functions are the **sigmoid**, the **tanh**, the **ReLU** and the **leaky ReLU** functions.

3.1.1 Sigmoid Function

- The sigmoid function is given by

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- It's derivative, which is required for back propagation, is given by

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- Advantages: Everywhere differentiable, easy to compute derivative, does not suffer from exploding gradients
- Disadvantages: Suffers from vanishing gradients (as $\sigma(z) \to 1$ as $z \to \infty$, hence $\sigma(z)(1-\sigma(z)) \to 0$).

3.1.2 tanh function

- The Hyperbolic tan function (tanh in short) is given by

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- Its derivative is given as

$$\sigma'(z) = 1 - \sigma(z)^2$$

- Advantages and disadvantages are similar to sigmoid function

3.1.3 Rectified Linear Unit (ReLU)

- The ReLU function is given by

$$\sigma(z) = \begin{cases} z \text{ if } z \geq 0 \\ 0 \text{ if } z < 0 \end{cases}$$

- Advantages: Does not suffer from vanishing gradients, easy to compute, better real-world performance than sigmoids
- Disadvantages: Suffers from exploding gradients as the value of activation is not constrained in the positive direction, suffers from Dying ReLU problem caused by too many negative activations (negative activations lead to 0 output, hence neuron does not learn anymore)

3.1.4 Leaky Rectified Linear Unit (Leaky ReLU)

- The Leaky ReLU is given by

$$\sigma(z) = \begin{cases} z \text{ if } z \geq 0 \\ az \text{ if } z < 0, \text{ where } 0 < a < 1 \end{cases}$$

3.2 Backpropagation Algorithm

- Backpropagation is the most commonly used algorithm in multi-layer neural networks to minimize the cost function.
- The derivation presented here is for Stochastic gradient descent, which considers the calculation for only one training example at a time.
- Let E_d be the error for one training example summed over all the units in the output layer of the multi-layer network.

$$E_d = \sum_{k \in output\_units} (t_k - o_k)^2$$

- The update rule can then be defined as:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \tag{6}$$

- Nomenclature:
  1. x_{ji}: The i^{th} input to the j^{th} unit.
  2. w_{ji}: The weight associated with the i^{th} input to the j^{th} unit.
  3. net_j: $\sum_i w_{ji} x_{ji}$, ie the weighted sum of inputs to the unit j
  4. o_j, t_j: Output and target of the j^{th} unit.
- For every training instance, the forward propagation step is done, and then the errors are propagated back (using the chain rule of differentiation) and then the weights are all modified. This is the stochastic version of back propagation.

3.2.1 Local Minima Avoidance in multilayer ANNs

- Using **stochastic** instead of batch gradient descent. As with SGD, the minimization of error happens over a different error function for each iteration, therefore on average, the likelihood that all the surfaces will descend into a local minima (and not a global one) is low.
- Adding **momentum** to the gradient descent update rule. With the momentum term, the new update rule is given by

  $$\Delta w_{ji}(n) = \eta\ \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1)$$

  here, $\Delta w_{ji}(n-1)$ is the update that occured to the weight w_{ji} in the $n-1^{th}$ iteration.
- Training multiple networks, each with **different initial weight values**. Either the best output can be selected (based on the validation data performance) or a weighted average of all the network's outputs can be used as the final prediction.

3.2.2 Preventing Overfitting in ANNs

- Make the model simpler (fewer layers, fewer units per layer)
- Regularization: A technique that places a penalty on the sum of the weights (L1) or the sum of squares of the weights (L2) by forcing that quantity to be minimized along with the loss function. The general formula for a loss function with L2 regularization is given as

  $$\textbf{min}\ L(y, \hat{y}) + \lambda \sum w_{ji}^2$$

- Early Stopping is a technique wherein the model training is stopped (ie. the number of iterations is reduced) so as to combat overfitting on the train set.
- Data Augmentation: Adding transformations of various types to the current data so as to obtain more data to train the model.

4 Support Vector Machines

- SVM is a supervised learning algorithm that can be used for both classification and regression tasks.
- When classifying, the aim of an SVM is to construct the **maximum margin classifier**, ie. the classifier that maximises the distance between the positive and negative points.
- Mathematically speaking, the SVM learns the equation of a linear decision hyperplane

  $$y = \vec{w}x + b$$

  Where w, b are parameters of the plane.
- For convention, the values of y for positive and negative examples are chosen as +1 and -1 respectively.

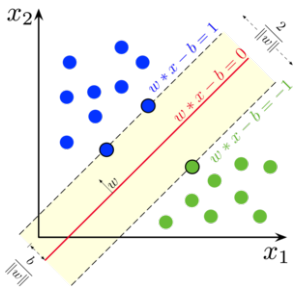

Figure 3: Support Vector Machine

- For the positive and negative examples respectively, we have

$$\overrightarrow{w}.\overrightarrow{x} + b \geq 1$$
$$\overrightarrow{w}.\overrightarrow{x} + b \leq -1$$

- Using the above definition of y, these can be combined into one equation

$$y.(\overrightarrow{w}.\overrightarrow{x} + b) - 1 \geq 0 \tag{7}$$

- For the points that lie *on tha margins*, the $\geq$ becomes an equality sign.
- Let x_+ and x_- be a positive and negative example (these lie on the margins be definition). The width of the margin is

$$d_m = \frac{\overrightarrow{w}}{||\overrightarrow{w}||}(\overrightarrow{x_+} - \overrightarrow{x_-})$$

substituting from equation (7), we have

$$d_m = \frac{1}{||\overrightarrow{w}||}(1 - b - (-1 - b))$$
$$d_m = \frac{2}{||\overrightarrow{w}||}$$

- Minimizing the quantity $\frac{2}{||\overrightarrow{w}||}$ can be shown to be equivalent to maximizing the quantity $\frac{1}{2}||\overrightarrow{w}||^2$ subject to the constraint in equation (7).

4.1 Constrained Optimization using Lagrangians

- First, the Lagrangian $\mathbb{L}$ is constructed as

$$\mathbb{L} = \frac{1}{2}||\overrightarrow{w}||^2 - \sum_{i=1}^{m} \alpha_i [y.(\overrightarrow{w}.\overrightarrow{x} + b) - 1] \tag{8}$$

- Where α_i is the **Lagrange Multiplier** for the ith training example. The summation is over the training examples.
- We compute the partial derivatives of the Lagrangian with respect to the parameters that are to be learnt, and set all the partial derivatives to 0.

$$\frac{\partial \mathbb{L}}{\partial w} = \overrightarrow{w} - \sum_{i=1}^{m} \alpha_i \overrightarrow{x_i} y_i = 0$$
$$\overrightarrow{w} = \sum_{i=1}^{m} \alpha_i \overrightarrow{x_i} y_i$$

and

$$\frac{\partial \mathbb{L}}{\partial b} = -\sum_{i=1}^{m} \alpha_i y_i = 0$$
$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

- By substituting these values back in the Lagrangian, we obtain the **Lagrangian Dual** $\mathbb{L}_d$

$$\mathbb{L}_d = \frac{1}{2} \sum_{i=1}^{m} \alpha_i \overrightarrow{x_i} y_i \sum_{j=1}^{m} \alpha_j \overrightarrow{x_j} y_j - \left[\sum_{i=1}^{m} \alpha_i \overrightarrow{x_i} y_i \sum_{j=1}^{m} \alpha_j \overrightarrow{x_j} y_j - \sum_{i=1}^{m} \alpha_i \right]$$

which gives:

$$\mathbb{L}_d = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{m,m} \alpha_i y_i \alpha_j y_j (\overrightarrow{x_i}.\overrightarrow{x_j}) \tag{9}$$

- The problem now reduces to

$$\begin{aligned} &\textbf{max } \mathbb{L}_d \\ \text{st. } &\sum_{i=1}^{m} \alpha_i y_i = 0 \\ &C > \alpha_i > 0 \ \forall \ i \end{aligned}$$

- The constraint C on the value of α is called the **slack** condition, which ensures that α does not go to infinity in case of a misclassification.
- The solution to the Lagrangian dual is obtained by solving $\frac{\partial \mathbb{L}_d}{\partial \alpha} = 0$ for the value of α. This value of α is subsituted above to get the value of w and b for the max margin classifier.
- The final solution needs to satisfy the **Karush-Kuhn-Tucker** conditions (as the constraint involves an inequality). These are:
  1. **Stationary Condition**:

$$\begin{aligned} \frac{\partial \mathbb{L}}{\partial w} &= 0 \\ \frac{\partial \mathbb{L}}{\partial b} &= 0 \end{aligned}$$

  2. **Primal Feasibility**:

$$y.(\overrightarrow{w}.\overrightarrow{x} + b) - 1 \geq 0 \ \forall \ i$$

  3. **Dual Feasibility**:

$$\alpha_i \geq 0 \ \forall \ i$$

  4. **Complementary Slackness**:

$$\alpha_i[y.(\overrightarrow{w}.\overrightarrow{x} + b) - 1] = 0$$

- From the slackness condition, it can be derived that for all non-support vectors, the value of α_i is 0. This means that the majority of α_i are 0, leading to less memory needed to store them.