Artificial Intelligence

Summer of Science 2022

Anurag Chaudhari

Mentor: Mohammad Taufeeque

IIT BOMBAY

Contents

What have I learnt?

Machine Learning (ML) has always been a field of interest for me. Even if I had zero clue about what concepts it includes or in fact, what forms the basis of ML, I always wonder how these automated systems learn new responses. Does someone have to continuously program the response into the system for each newly generated prompt? There has to be a better way, I used to think.

One day, I was introduced to it through a friend. I immediately decided I had to follow my career in this field. Me, not being from a Computer Science background, the first couple of weeks of Summer of Science were tough as a lot of new concepts and definitions were thrown at me at once. However, I tried my best to learn bit by bit, concept by concept and I can say now that I do have at least some idea of what goes on in ML.

The first half, till the midterm report, was mostly focused on learning concepts and new algorithms in ML. I learnt a lot of basic concepts like regression, classification, multi-variable classification, and more. I tried to gain clarity on these concepts through as many examples as possible.

After the midterm, I learnt the Neural Network model. I gained a lot of interest in it and its applications. Hence, I decided to try and make a model with the application of Neural Network, myself. I went through a lot of related resources and videos. I found an easy and beginner-friendly application of NN to make a Digit Recognition Model. Hence, I decided to make one of my own. I will be presenting this model in the presentation video.

Learning concepts is one thing. But to actually apply those and successfully make a working model provides another level of satisfaction.

What is Machine Learning?

Machine learning is a set of tools that allow us to “teach” computers how to perform tasks by providing examples of how they should be done. For example, we wish to write a program to distinguish between valid email messages and unwanted spam. We could try to write a set of simple rules for flagging the messages that contain certain features, such as the word “spam”. However, focusing on writing rules to accurately distinguish which text is valid can be quite difficult to do well, resulting in either many missed spam messages, or, many lost emails. Worse, the spammers will actively adjust the way they send spam to trick these strategies (e.g., writing “sp@m”). Writing effective rules, then keeping them up-to-date, quickly becomes an impossible task. Fortunately, machine learning provides a solution. Modern spam filters are “learned” from examples. That is, we provide the learning algorithm with example emails that we have manually labeled as “valid” or “spam”, and the algorithms learn to distinguish between them automatically.

Applications of Machine Learning:

The application of machine learning methods to large databases is called data mining. In data mining, a large volume of data is processed to construct a simple model with valuable use.

Here are some of the applications of machine learning:

i. Image Recognition
ii. Speech Recognition
iii. Traffic prediction
iv. Product recommendations
v. Self-driving cars
vi. Virtual Personal Assistant
vii. Online Fraud Detection
viii. Stock Market trading
ix. Automatic Language Translation

Definition of learning:

A computer program is said to learn from experience E concerning some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.

Examples

i) A robot driving learning problem
   - Task T: Driving on highways using vision sensors
   - Performance measure P: Average distance traveled before an error
   - Training experience E: A sequence of images and steering commands recorded while observing a human driver

ii) A chess learning problem
   - Task T: Playing chess
   - Performance measure P: Percent of games won against opponents
   - Training experience E: Playing practice games against itself

iii) Handwriting recognition learning problem
   - Task T: Recognising and classifying handwritten words within images
   - Performance P: Percent of words correctly classified
   - Training experience E: A dataset of handwritten words with given classifications

A computer program that learns from experience is called a machine learning program.

Types of Machine Learning:

1. **Supervised Learning,** in which the training data is labeled with the correct answers or outputs, e.g., “spam” or “valid”. The two most common types of supervised learning are:
   i. Regression, where the outputs are real-valued.
      For example, given a picture of a person, we have to predict their age based on the given picture.
   ii. Classification, where the outputs are discrete labels.
      Given a patient with a tumor, we have to predict whether the tumor is malignant or benign.

2. **Unsupervised learning,** correct responses are not provided, but instead, the algorithm tries to identify similarities between the inputs. So, inputs that have something in common are

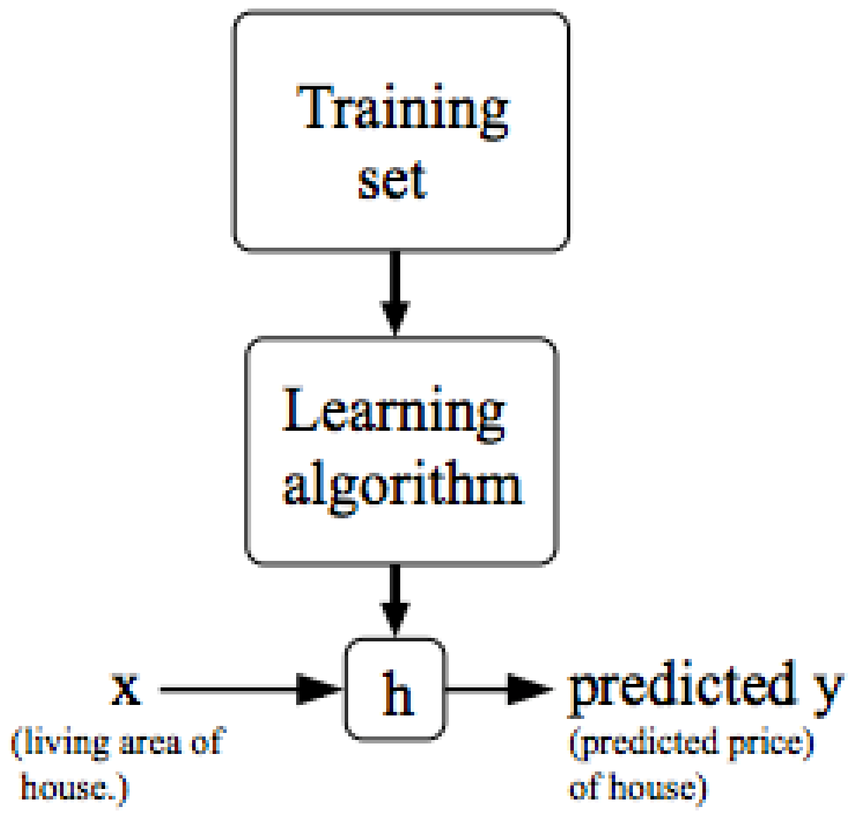
categorized together. Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses.
In unsupervised learning algorithms, classification or categorization is not included in the observations. There are no output values and so there is no estimation of functions. Since the examples given to the learner are unlabeled, the accuracy of the structure that is output by the algorithm cannot be evaluated.

i. **Clustering**: Take a collection of 1,000,000 different genes, and find a way to automatically group these genes into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.
ii. **Non-clustering:** The "Cocktail Party Algorithm", allows you to find structure in a chaotic environment. (i.e. identifying individual voices and music from a mesh of sounds at a cocktail party).

Model Representation:

Our goal in the supervised learning problem is, given a training set, to learn a function

h: X → Y so that h(x) is a "good" predictor for the corresponding value of y. This function h is called a hypothesis. The process is therefore represented pictorially like this:



For example, given data on Housing prices vs Size, we are asked to predict the selling price of our friend's house. We know the size of the house. The target variable that we're trying to predict here is continuous, hence we call the learning problem a regression problem.

But when y can take on only a small number of discrete values, such as if given the living area, we wanted to predict if a dwelling is a house or an apartment, we call it a classification problem.

To establish notation for future use, we'll use $x^{(i)}$ to denote the "input" variables (living area in this case), also called input features. And $y^{(i)}$ will denote the "output" or target variable that we are trying to predict (housing price). A pair $(x^{(i)}, y^{(i)})$ is called a training example, and the dataset that we'll be using to learn, a list of m training examples $(x^{(i)}, y^{(i)})$; i = 1,...,m is called a training set. The superscript "(i)" in the notation is an index into the training set and has nothing to do with exponentiation. We will also use X to denote the space of input values, and Y to denote the space of output values. In this example, X = Y = $\mathbb{R}$.

Linear Regression:

Cost Function:

The accuracy of our hypothesis function can be measured by using a cost function. This takes an average difference of all the results of the hypothesis with inputs from x's and the actual output y's. This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the $\frac{1}{2}$ term.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

It is $\frac{1}{2}\bar{x}$ where $\bar{x}$ is the mean of the squares of $h_\theta(x_i) - y_i$, or the difference between the predicted value and the actual value.

In linear regression with one variable, $h_\theta(x) = \theta_0 + \theta_1 x$.

Our training data set is scattered on the x-y plane and we are trying to make a straight line defined by $h_\theta(x)$ that passes through these scattered data points. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. Ideally, the line should pass through all the points of our training data set. In this case, the value of $J(\theta_0, \theta_1)$ will be 0. Thus as a goal, we should try to minimize the cost function.

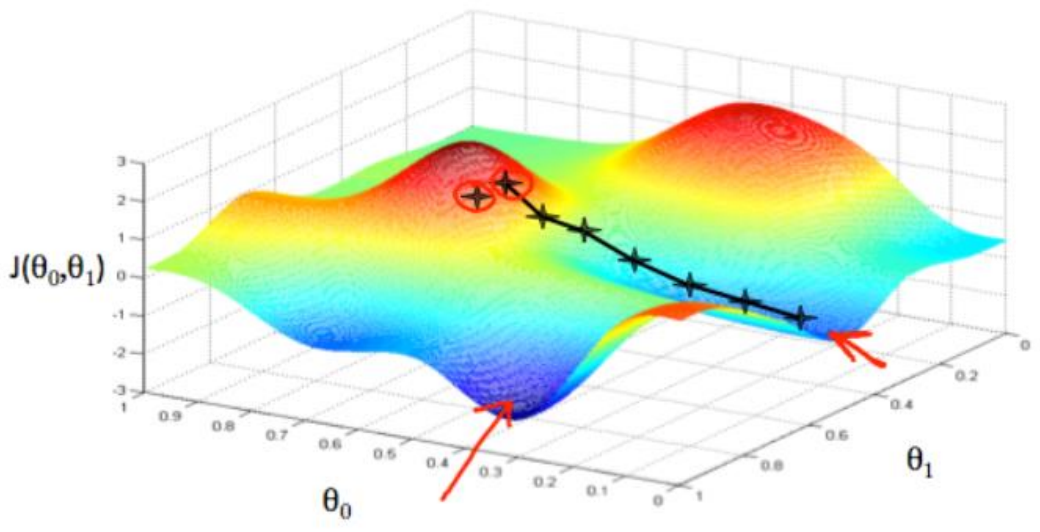
Cost Function - Intuition I: If θ_0 is considered 0, $h_\theta(x) = \theta_1 x$. This represents straight lines passing through the origin. Our goal is to minimize J, which due to the presence of the term h^2, will be a quadratic equation. And the minimum value of $J(\theta_1)$ will be at the value of θ_1 at the global minimum.

Cost Function - Intuition II: If θ_0 is not taken as zero, we will get a shape in 3D. This can be represented graphically for better understanding using the contour plots. A contour plot is a graph that contains many contour lines. A contour line of a two-variable function has a constant value at all points of the same line. To find the point of global minima of this shape, several approaches are followed.

Gradient Descent:

- Gradient descent is used to minimize the $J(\theta_0, \theta_1)$ by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively updating the values to reach the minimum cost function.

We graph the cost function as a function of the parameter estimates. We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters. We put θ_0 on the x-axis and θ_1 on the y-axis, with the cost function on the vertical z-axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters. We will succeed when the cost function would lie at the bottom of the pit of absolute minima. Below is a graphical representation of gradient descent for some data.



We do so, by considering some value of θ_0, θ_1 and then decreasing it in the steps of α times the partial derivative of $J(\theta_0, \theta_1)$. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The parameter α is called the learning rate and it determines the size of each step.

The distance between each 'star' in the graph above represents a step determined by our parameter α. A smaller α would result in a smaller step and a larger α results in a larger step. The direction in which the step is taken is determined by the partial derivative of $J(\theta_0, \theta_1)$. Depending on where one starts on the graph, one could end up at different points. The image above shows us two different starting points that end up in two different places. The left star would lead us to the minima marked at the left of the graph.

To generalize it for multiple parameters,

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

j = represents the feature index number.

At each iteration j, one should simultaneously update the parameters θ_1, θ_2,..., θ_n. Updating a specific parameter before calculating another one on the j^{th} iteration would yield a wrong implementation.

Also, choosing an apt value of α is crucial as, if the value is too large, it would not converge. While, if the α is small, it would take a very long time to get to the minima.

Multivariate Linear Regression:

Linear regression with multiple variables is also known as "multivariate linear regression". The multivariable form of the hypothesis function accommodating multiple features is as follows:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

Using the definition of matrix multiplication, our multivariable hypothesis function can be concisely represented as:

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \ldots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

Gradient Descent for Multiple Variables:

The gradient descent equation itself is generally the same form; we just have to repeat it for our ‘n’ features:

$$\begin{aligned} &\text{repeat until convergence: } \{ \\ &\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0...n} \\ &\} \end{aligned}$$

Normal Equation:

Gradient descent gives one way of minimizing J. Let’s discuss a second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm. In the “Normal Equation” method, we will minimize J by explicitly taking its derivatives wrt the θ_j*’s*, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:

$$\theta = (X^T X)^{-1} X^T y$$

The following is a comparison of gradient descent and the normal equation:

Gradient Descent	**Normal Equation**
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate the inverse of X^TX
Works well when n is large	Slow if n is very large

When implementing the normal equation in octave we want to use the 'pinv' function rather than 'inv'. The 'pinv' function will give you a value of θ even if X^TX is not invertible. The exact definition of the 'pinv' function has not been discussed here, as it is not necessary to understand the concepts of Machine Learning.

If X^TX is noninvertible, the common causes might be having:

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization" (to be explained in a later lesson).

Solutions to the above problems include deleting a feature that is linearly dependent on another or deleting one or more features when there are too many features.

Classification:

In Regression algorithms, we have predicted the output for continuous values, but to predict the categorical values, we need Classification algorithms. The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations based on training data. In Classification, a program learns from the given dataset or observations and then classifies new observations into several classes or groups. Classes can be called targets/labels or categories. Unlike regression, the output variable of Classification is a category, not a value. We will focus on the binary classification problem in which y can take on only two values, here we take 0 and 1. Most of what we say here will also generalize to the multiple-class case. Suppose, if we are trying to build a spam classifier for email, then $x^{(i)}$ maybe some features of a piece of email, and y maybe 1 if it is a piece of spam mail, and 0 otherwise. Hence,

$y \in \{0,1\}$. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols “-” and “+.” Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the label for the training example.

Logistic Regression:

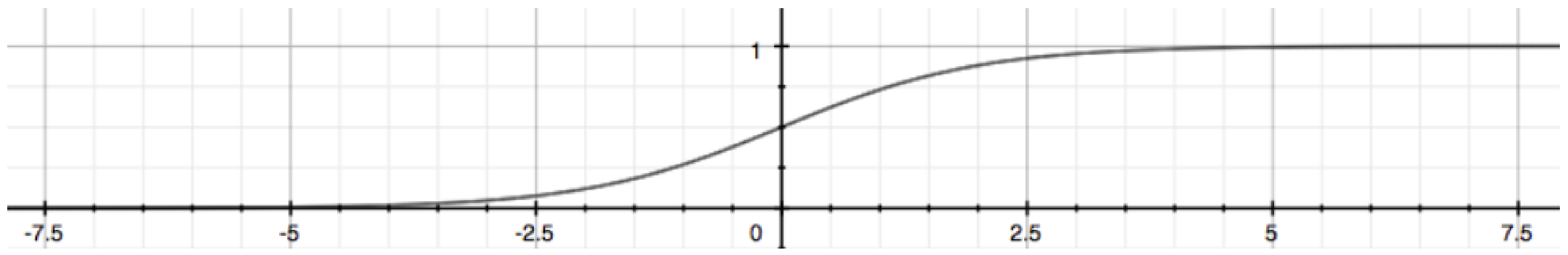
Logistic Regression is a significant machine learning algorithm because it can provide probabilities and classify new data using continuous and discrete datasets. We could approach the classification problem by ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn’t make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$. To fix this, let’s change the form of our hypotheses $h_\theta(x)$ to satisfy $0 \leq h_\theta(x) \leq 1$. This is accomplished by plugging $\theta^T x$ into the Logistic Function.

Our new form uses the Sigmoid Function, also called the Logistic Function:

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$
$$g(z) = \frac{1}{1+e^{-z}}$$

the sigmoid function looks like this:



The function g(z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification. $h_{\theta(x)}$ will give us the probability that our output is 1. For example, $h_{\theta(x)} = 0.7$ gives us a probability of 70% that our output is 1. If the probability that it is 1 is 70%, then the probability that it is 0 is 30%.

$$h_\theta(x) = P(y=1|x;\theta) = 1 - P(y=0|x;\theta)$$
$$P(y=0|x;\theta) + P(y=1|x;\theta) = 1$$

Decision Boundary:

The decision boundary is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

$h_\theta(x) \geq 0.5 \rightarrow y = 1$

$h_\theta(x) < 0.5 \rightarrow y = 0$

Also,

$z = 0, e^0 = 1 \Rightarrow g(z) = \frac{1}{2}$

$z \rightarrow \infty, e^{-\infty} \rightarrow 0 \Rightarrow g(z) = 1$

$z \rightarrow -\infty, e^{\infty} \rightarrow \infty \Rightarrow g(z) = 0$

When input in g is greater than or equal to zero, its output is greater than or equal to 0.5. So if the input is $\theta^T X$, then that means:

$h_\theta(x) = g(\theta^T x) \geq 0.5$

$when\ \theta^T x \geq 0$

This implies:

$\theta^T x \geq 0 \Rightarrow y = 1$

$\theta^T x < 0 \Rightarrow y = 0$

Example:

$$\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$$
$$y = 1\ if\ 5 + (-1)x_1 + 0x_2 \geq 0$$
$$5 - x_1 \geq 0$$
$$-x_1 \geq -5$$
$$x_1 \leq 5$$

In this case, the decision boundary is a straight vertical line placed on the graph where $x_1 = 5$, and the portion to the left of that denotes y = 1, while that to the right denotes y = 0. The input to the sigmoid function g(z) doesn't need to be linear and could be a function that describes a circle or any shape that fits our data.

Cost Function:

The same cost function that we use for linear regression cannot be used because the Logistic Function will cause the output to be wavy, causing many local optima. It will not be a convex function. Our cost function for logistic regression looks like this:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x(i)\right), y(i)\right)$$

$Cost(h_\theta(x), y) = -\log(h_\theta(x) \qquad if\ y = 1$

$Cost(h_\theta(x), y) = -\log(1 - h_\theta(x) \qquad if\ y = 0$

So,

$Cost(h_\theta(x), y) = 0 \qquad if\ h\theta(x) = y$

$Cost(h_\theta(x), y) \rightarrow \infty \qquad if\ y = 0\ and\ h_\theta(x) \rightarrow 1$

$Cost(h_\theta(x), y) \rightarrow \infty \qquad if\ y = 1\ and\ h_\theta(x) \rightarrow 0$

Writing the cost function in this way guarantees that $J(\theta)$ is convex for logistic regression.

Multiclass Classification: One-vs-all:

This approach is for the classification of data when we have more than two categories. Instead of $y = \{0,1\}$ we will expand our definition so that $y = \{0,1...n\}$. Since $y = \{0,1...n\}$, we divide our problem into n+1 binary classification problems. n+1 because the index starts at 0. In each one, we predict the probability that y is a member of one of our classes.

$y \in \{0,1 \dots n\}$

$h_\theta^{(0)}(x) = P(y = 0|x; \theta)$

$h_\theta^{(1)}(x) = P(y = 1|x; \theta)$

.

.

.

$h_\theta^{(n)}(x) = P(y = n|x; \theta)$

$prediction = \max_i(h_\theta^{(i)}(x))$

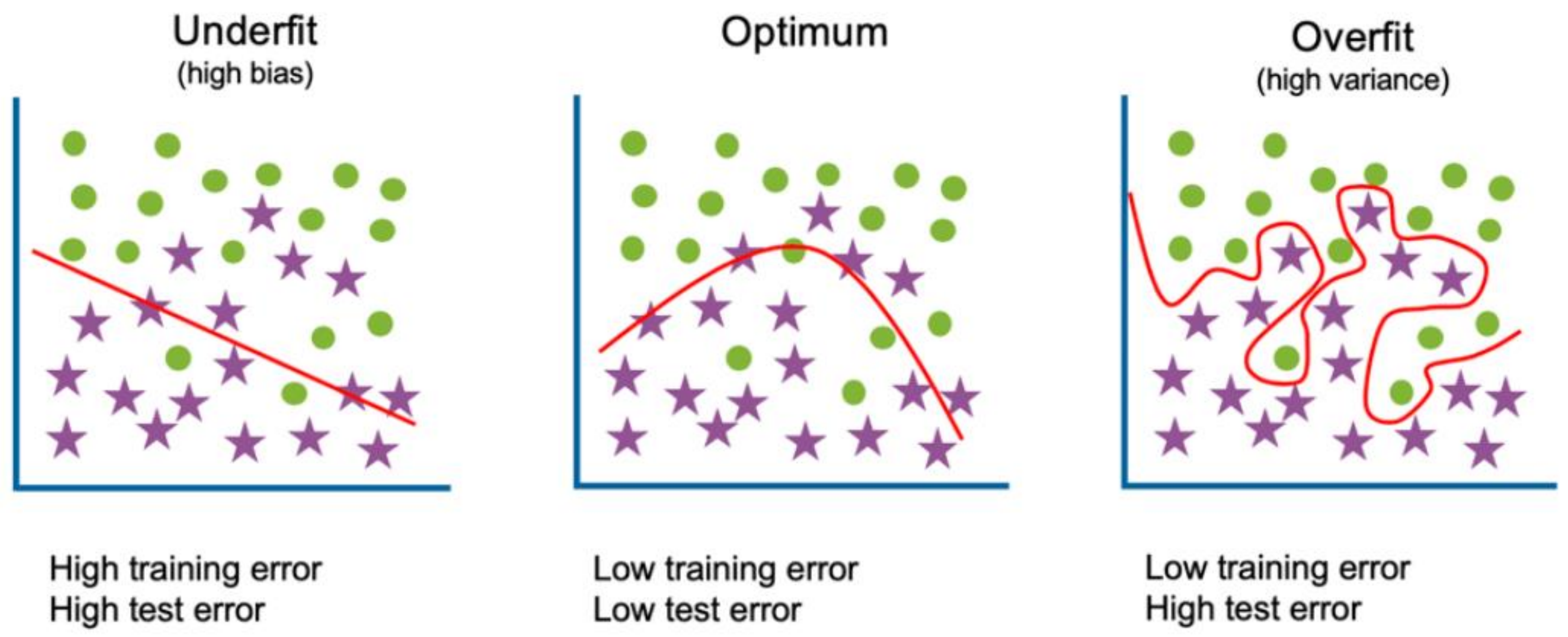
We choose one class and then lump all the others into a single second class. We do this repeatedly, applying binary logistic regression to each case. Then use the hypothesis that returned the highest value as our prediction.

Overfitting in Machine Learning:

In the real world, the dataset present will never be clean and perfect. Each dataset contains impurities, noise, outliers, missing data, or imbalanced data. Due to these impurities, different problems occur that affect the accuracy and the performance of the model.

Before understanding overfitting, we need to know some basic terms, which are:

- Noise: Noise is irrelevant data present in the dataset. It affects the performance of the model.
- Bias: Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.
- Variance: If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.
- Generalization: It shows how well a model is trained to predict unseen data.



The left figure above shows the result of fitting $y = \theta_0 + \theta_1 x$ to a dataset. We can see that the straight line is not a very good fit for the data. This is underfitting.

If we had added an extra feature x^2, and fit $y = \theta_0 + \theta_1 x + \theta_2 x^2$, then we obtain a slightly better fit to the data as seen in the figure in the center.

It might seem that the more features we add, the better fit it would be to the data. The rightmost figure is the result of fitting a higher order polynomial, let us say it is a 5th order polynomial $y = \sum_{j=0}^{5} \theta_j \, x^j$. Even though the curve passes through the data perfectly, we would not expect this to be a very good predictor due to multiple local minima. This is an example of overfitting.

There are two main options to address the issue of overfitting:

1) Reduce the number of features by manually selecting which features to keep.
2) Regularization: Keep all the features, but reduce the magnitude of parameters θ_j.

   Regularization works well when we have a lot of slightly useful features.

Regularization:

Cost Function:

If we have overfitting from our hypothesis function, we can reduce the weight that some of the terms in our function carry by increasing their cost.

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i\right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

The λ is the regularization parameter. It determines how much the costs of our theta parameters are inflated. Using the above cost function with the extra summation, we can smooth the output of our hypothesis function to reduce overfitting. If lambda is chosen to be too large, it may smooth out the function too much and cause underfitting.

Regularization to both linear regression and logistic regression. We will approach linear regression first.

Gradient Descent:

θ_0 is separated from the rest of the parameters before we modify gradient descent because we do not want to penalize θ_0.

$Repeat\ \{$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}\right) + \frac{\lambda}{m} \theta_j\right] \qquad j \in \{1, 2 \ldots n\}$$

$\}$

With some manipulation our update rule can also be represented as:

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

The first term in the above equation, $1 - \alpha \frac{\lambda}{m}$ will always be less than 1. It reduces the value of θ_j by some amount on every update. The second term is the same as it was before.

Normal Equation:

Let's approach regularization using the alternate method of the non-iterative normal equation. The equation is similar to our original, except that we add another term inside the parentheses:
$\theta = (X^T X + \lambda \cdot L)^{-1} X^T y$

$$\text{where } L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

L is a matrix with 0 at the top left and 1's down the diagonal, with 0's everywhere else. It should have dimension $(n + 1) \times (n + 1)$. This can be supposed as the identity matrix, though we are not including x_0, multiplied with a single real number λ. We had seen that if $m < n$, then $X^T X$ is non-invertible. However, when we add the term $\lambda \cdot L$, then $X^T X + \lambda \cdot L$ becomes invertible.

Regularized Logistic Regression:

We can regularize logistic regression in a similar way that we regularize linear regression. As a result, we can avoid overfitting.

Our cost function for logistic regression was:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$

It can be regularized by adding a term to the end:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2\mathrm{m}}\sum_{j=1}^{n} \theta_j^2$$

Neural Network:

Model Representation:

Neurons are basically computational units that take inputs (dendrites) as electrical inputs (called "spikes") that are channeled to outputs (axons). In our neural network model, our dendrites are like the input features $x_1, x_2 \ldots . x_n$, and the output is the result of our hypothesis function. In this model our x_0, the input node is sometimes called the "bias unit". It is always equal to 1. In neural networks, we use the same logistic function as in classification, $\frac{1}{1 + e^{-\theta^T x}}$, yet we sometimes call it a sigmoid (logistic) activation function. "theta" parameters are sometimes called "weights".

$$\left[x_0 x_1 x_2 x_3\right] \rightarrow \left[a_1^{(2)} a_2^{(2)} a_3^{(2)}\right] \rightarrow h_\theta(x)$$

$a_i^{(j)} =$ "activation" of unit i in layer j
$\Theta^{(j)} =$ matrix of weights controlling function mapping from layer j to layer $j+1$

Our input nodes (layer 1), also known as the "input layer", go into another node (layer 2), which finally outputs the hypothesis function, known as the "output layer". We can have intermediate layers of nodes between the input and output layers called the "hidden layers". These intermediate or "hidden" layer nodes are labeled as $a_0^2, \ldots, a_n^2$ and are called "activation units".

The values for each of the "activation" nodes are obtained as follows:

$$\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \\
h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})
\end{aligned}$$

Each layer gets its matrix of weights, $\Theta^{(j)}$. The dimensions of these matrices of weights are determined as follows; if the network has s_j units in layer j and s_{j+1} units in layer j+1, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (\mathrm{s_j} + 1)$. The +1 comes from the addition in $\Theta^{(j)}$ of the "bias" nodes, x_0 and $\Theta_0^{(j)}$. In other words, the output nodes will not include the bias nodes while the inputs will.

Now, we will do a vectorized implementation of the above functions. We are going to define a new variable $z_k^{(j)}$ that encompasses the parameters inside our g function. If we replaced all the parameters with the variable z:

$$a_1^{(2)} = g(z_1^{(2)})$$
$$a_2^{(2)} = g(z_2^{(2)})$$
$$a_3^{(2)} = g(z_3^{(2)})$$

In other words, for layer j=2 and node k, the variable z will be:

$$z_k^{(2)} = \Theta_{k,0}^{(1)} x_0 + \Theta_{k,1}^{(1)} x_1 + \cdots + \Theta_{k,n}^{(1)} x_n$$

The vector representation of x and z^j is:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \cdots \\ z_n^{(j)} \end{bmatrix}$$

Setting $x = a^{(1)}$, we can rewrite the equation as:

$$z^{(j)} = \Theta^{(j-1)} a^{(j-1)}$$

We are multiplying our matrix $\Theta^{(j-1)}$ with dimensions $s_j \times (n+1)$ (where s_j is the number of our activation nodes) by our vector $a^{(j-1)}$ with height (n+1). This gives us our vector $z^{(j)}$ with height s_j. Now we can get a vector of our activation nodes for layer j as follows:

$$a^{(j)} = g(z^{(j)}).$$

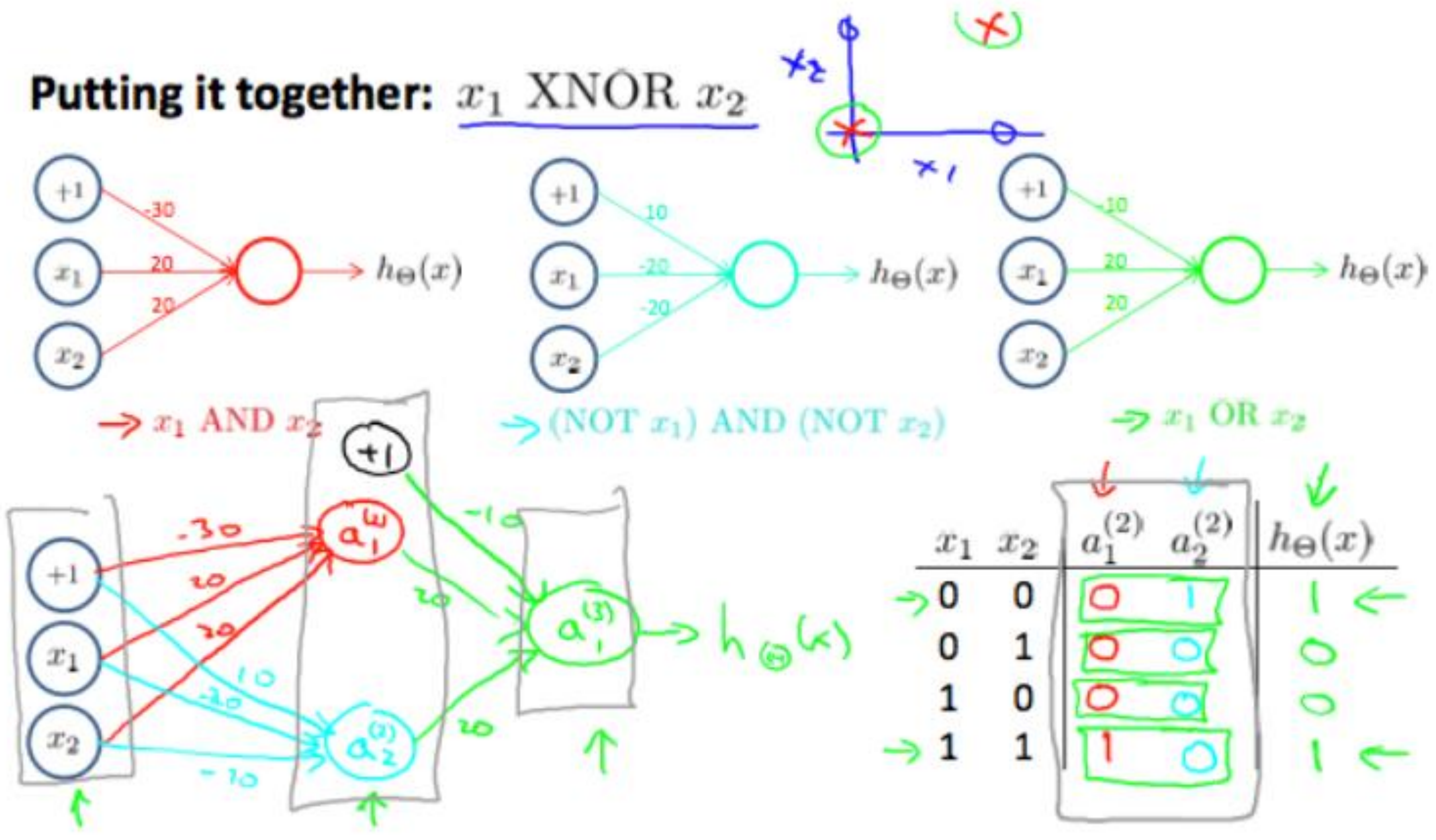
Where our function g can be applied element-wise to our vector $z^{(j)}$. We can then add a bias unit (equal to 1) to layer j after we have computed $a^{(j)}$. This will be the element $a_0^{(j)}$ and will be equal to 1. To compute our final hypothesis, let us first compute another z vector:

$$z^{(j+1)} = \Theta^{(j)} a^{(j)}$$

We get this final z vector by multiplying the next theta matrix after $\Theta^{(j-1)}$ with the values of all the activation nodes we just got. This last theta matrix $\Theta^{(j)}$ will have only one row which is multiplied by one column $a^{(j)}$ so that our result is a single number. We then get our final result with:

$$h_\Theta(x) = a^{(j+1)} = g(z^{(j+1)})$$

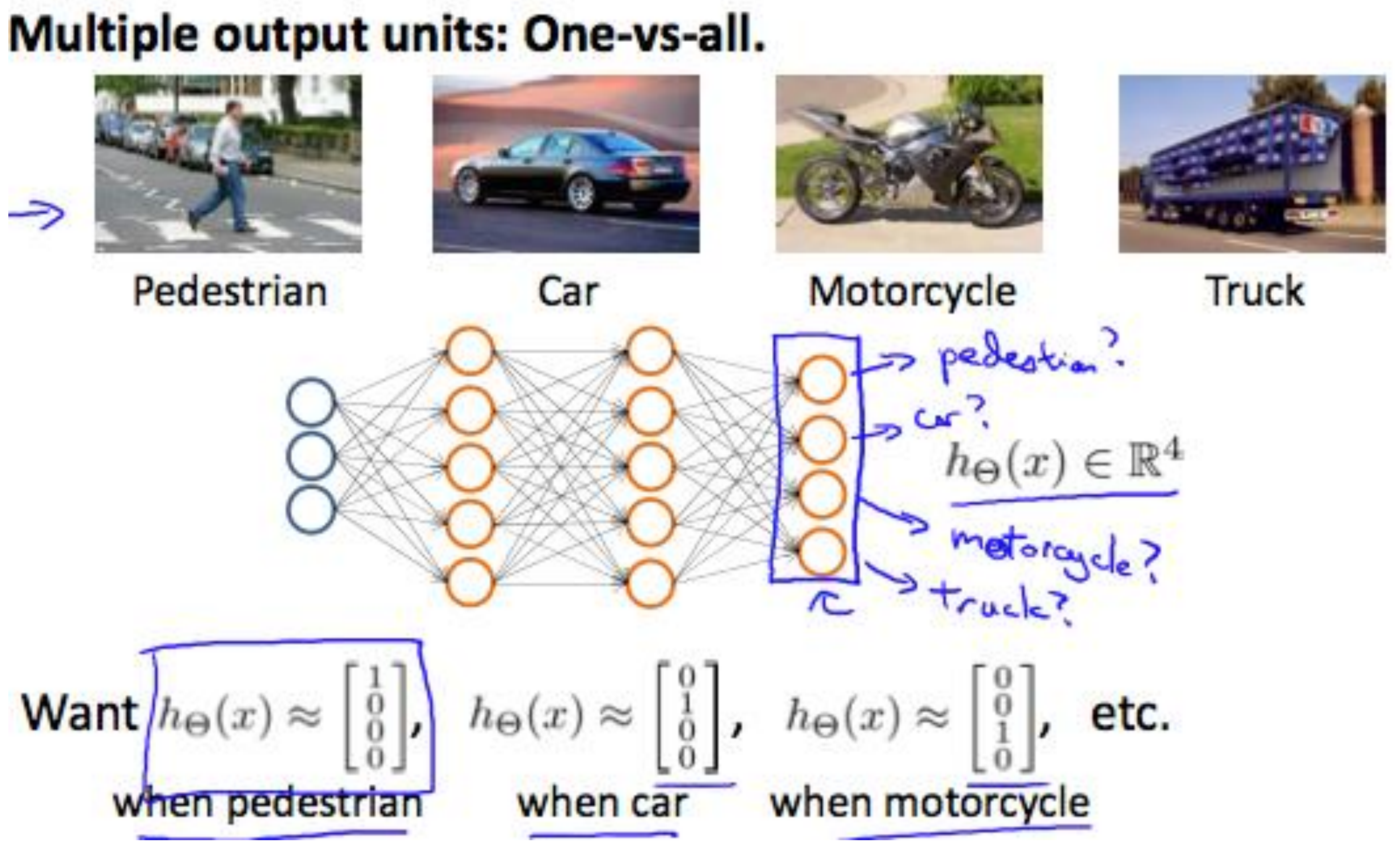
Example: XNOR operator


Putting it together: x_1 XNOR x_2
x_1 AND x_2
(NOT x_1) AND (NOT x_2)
x_1 OR x_2
$h_\Theta(x)$

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Multiclass Classification:

To classify data into multiple classes, we let the hypothesis function return a vector of values. We will use the following example to see how this classification is done. This algorithm takes as input an image and classifies it accordingly:


Multiple output units: One-vs-all.
Pedestrian
Car
Motorcycle
Truck
pedestrian?
car?
$h_\Theta(x) \in \mathbb{R}^4$
motorcycle?
truck?
Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.
when pedestrian
when car
when motorcycle

We can define our set of resulting classes as y:

$$y^{(i)} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix},$$

Each $y^{(i)}$ represents a different image corresponding to either a car, pedestrian, truck, or motorcycle. The inner layers, each provide some new information which leads to our final hypothesis function. The setup looks like this:

$$\begin{bmatrix}x_0\\x_1\\x_2\\\cdots\\x_n\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{(2)}\\a_1^{(2)}\\a_2^{(2)}\\\cdots\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{(3)}\\a_1^{(3)}\\a_2^{(3)}\\\cdots\end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix}h_\Theta(x)_1\\h_\Theta(x)_2\\h_\Theta(x)_3\\h_\Theta(x)_4\end{bmatrix}$$

Our resulting hypothesis for one set of inputs may look like this:

$h_\Theta(x) = [0010]$

In which case our resulting class is the third one down, or $h_\Theta(x)_3$, which represents the motorcycle.

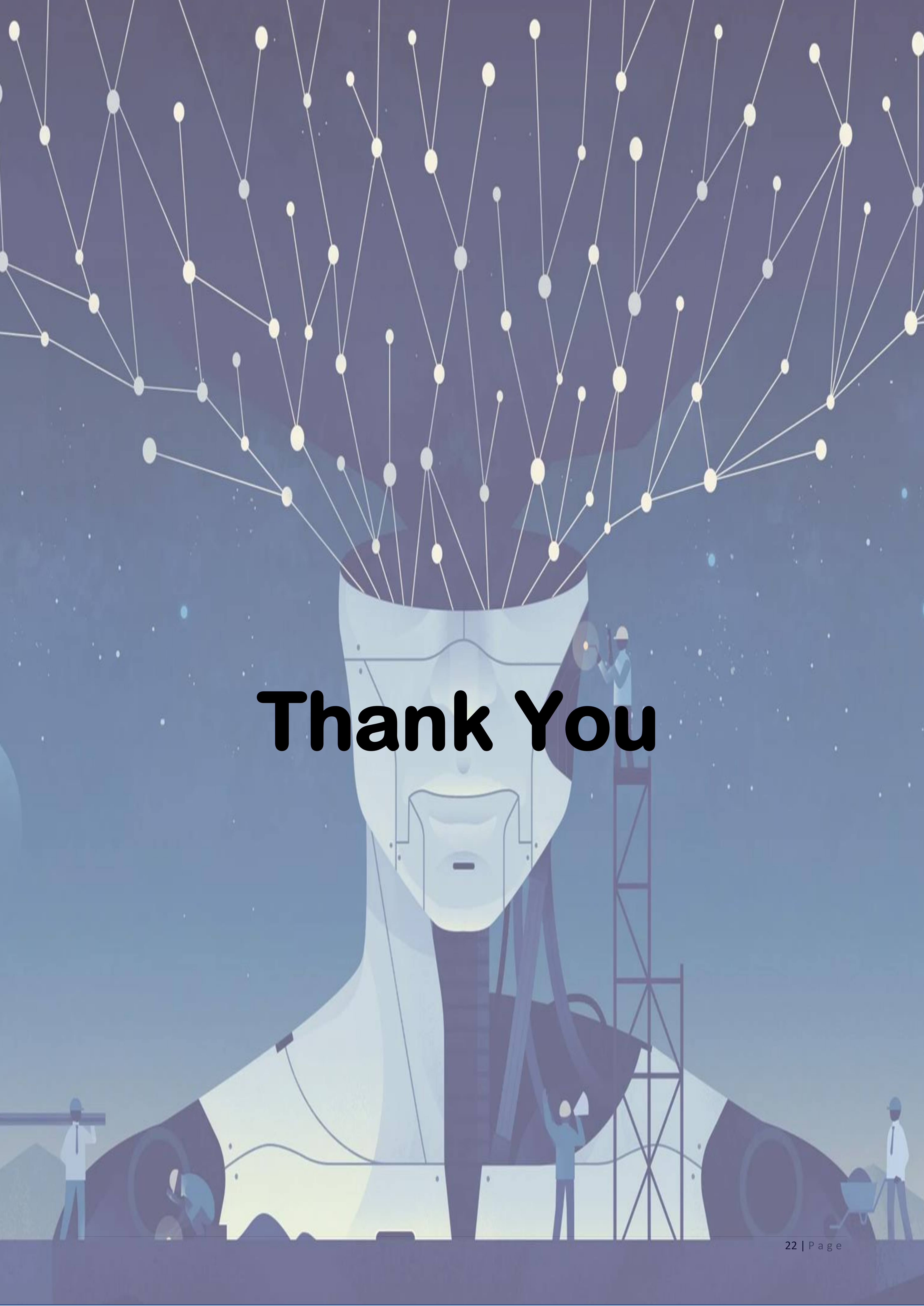
References

1. Machine Learning by Stanford University, Instructor: Andrew Ng, Coursera

2. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, Book by Geron Aurelien.

3. Machine Learning Full Course - Learn Machine Learning 10 Hours, Youtube video by Edureka

4. Geeks for Geeks Website: www.geeksforgeeks.org/



Thank You