2023

Machine Learning - Term End Assignment

O.P.JINDAL GLOBAL UNIVERSITY
SHUAIB SULEMAN
APPLICATIONS OF MACHINE LEARNING

Table of Contents

Question 1

a) Explain the process by which a machine learns about a business problem.

The process by which a machine learns about a business problem is as follows:

Learner: The machine itself is the first component of machine learning, which serves as the learner. This machine must have the necessary storage, space, and processing capacity to perform the intended task.

The second component is data, which replicates events that occurred in the past when decisions were made within the context of the problem. The data is separated into training data, which assists the machine in learning, and test data, which is used to evaluate the machine's learning after completing training. The data may be organised, unorganised, or semi-organized.

The third component is the algorithm, which is a mathematical formula that receives and analyses inputs to predict outputs within an acceptable range. We construct the rule set from the problem's available instructions.

The parameters are the fourth component of the machine learning process. These are conditions that influence how the algorithm will function. Imagine parameters as gauges on a machine that can be altered to influence the algorithm's operations and achieve the desired output.

Hyperparameters are the final essential element of this machine-learning conundrum. A human must determine these variables and dimensions for the machine. They determine the extent of the learning process.

Training: During training, the machine calculates the inputs and adjusts the model parameters as needed to generate the outputs corresponding to those inputs. As the machine processes the training data, it generates a model consisting of the algorithm and parameters that compute outputs based on inputs with the highest degree of precision.

Feedback and Adjustment: The machine uses any discrepancies between the generated and expected outputs as feedback and attempts to adjust its algorithmic parameters so that the actual output values are as close as possible to the expected values. This is the crux of the learning process. It generates the final model with defined parameters that can be evaluated on new data to make predictions or solve a classification problem.

Prediction: Now, when inputs with unknown outputs are fed into a machine, the machine can calculate or predict the outputs and assist us in making decisions for various business problems.

Iterative Process The process of machine learning is iterative. As the machine receives more data, its model is continuously refined. This is how the machine "learns" - by constantly adjusting its model to suit the data better.

This is comparable to how humans learn. We begin with a set of instructions or rules, gain experience by applying these rules, and then modify our strategy based on the results of our feedback. This iterative process of learning, modifying, and applying is what enables humans and machines to learn and develop over time.

b) Explain the different components of the ML process and their role in the learning process.

The six essential components of machine learning (ML) and their respective functions in the learning process:

Business objective: This component is essential for establishing the foundation of the ML process. Comprehending the business objectives, defining the specific problem to be addressed, and determining the appropriate machine learning task (e.g., classification, regression, clustering) are all required. A well-defined problem ensures that the ML model concentrates on the correct problem and generates valuable insights.

Data Collection: The quality of the data used for training and validating the machine learning model substantially affects its performance. During this phase, relevant data is collected from various sources, such as databases, APIs, and web crawling. High-quality, diverse, and representative data enables the model to learn more efficiently and generalise to new, unseen data more precisely.

Data Pre-treatment or Pre-processing: Raw data needs to be more coherent and consistent. Data pre-processing entails cleansing and transforming the data to make it compatible with the chosen ML algorithm. This section handles absent values, eliminates outliers, and normalises or scales features. Appropriate pre-processing guarantees that the machine learning model learns from precise, consistent, and meaningful data.

Feature Selection: This component generates new features or variables from existing data to improve the model's predictive capabilities. Techniques such as encoding categorical variables, creating interaction terms, and aggregating data may be utilised in feature engineering. A model's ability to capture complex relationships and increase its overall performance can be enhanced by well-designed features.

Selection and Training of Models: A suitable ML algorithm is chosen based on the problem definition and data characteristics. The selected algorithm is then applied to a subset of the pre-processed data to train a model. During training, the model discovers data patterns and relationships, enabling it to make predictions and decisions. This element is essential for developing a model that effectively addresses the business issue.

Evaluation and Deployment of Models: After training, the efficacy and generalizability of the model are evaluated using a distinct subset of data. This component identifies potential overfitting and underfitting issues and verifies that the model is deployable. Once the model satisfies predefined performance criteria, it can be deployed to a production environment to facilitate data-driven decision-making. Routine monitoring and updates are required to maintain the model's efficacy as new data becomes available and the business problem evolves.

After deployment, frequently checking the model's performance and accuracy is critical. Data distribution or drift changes may need model retraining or fine-tuning. Updating the data and fixing any problems that may develop are also part of maintenance.

Question 2

a) What are arrays and arrays programming? What are NumPy arrays, and how to create vectors and matrices? Discuss them with examples. Solve the following matrix using NumPy.

$$2x + 3y + 4z = 20$$

$$x + 2y + 3z = 14$$

$$4x + y + z = 09$$

Arrays are data structures that contain elements of a single data type in a contiguous memory block. Arrays are data structures that hold a collection of items, with each element accessible by an index. Arrays may have various dimensions, with 1-dimensional arrays known as vectors, 2-dimensional arrays known as matrices, and higher-dimensional arrays known as tensors. They are used to implement various data structures and algorithms and are a fundamental concept in many programming languages.

Array programming is a programming paradigm in which operations are performed on entire arrays instead of individual elements, which can result in more concise and efficient code. Each element in an array has a unique index, which is an integer number in array programming. Indexes are ordered or sequenced, often beginning at zero. Array programming languages may execute operations on complete arrays effectively by employing indexes, taking advantage of memory location contiguity and the consistent data type of array components.

NumPy (Numerical Python) is an open-source package offering a powerful and versatile data structure: the n-dimensional array. NumPy is a major Python library that supports enormous multidimensional arrays and matrices. NumPy's primary functionality focuses on the ndarray multidimensional array object, which enables fast array operations such as mathematical, logical, shape manipulation, sorting, selecting, I/O, and discrete Fourier transforms. Because these arrays may be used to represent vectors, matrices, and tensors, NumPy is a must-have tool for dealing with numerical data and executing complicated mathematical operations in Python.

In addition to ndarrays, NumPy supports derived objects such as masked arrays and matrices and a collection of algorithms for performing quick array operations. Because of its ability to swiftly execute mathematical and logical operations on arrays, the library is frequently used in scientific computing, data analysis, and machine learning.

In conclusion, NumPy is an essential Python library that offers a high-performance multidimensional array object and a full collection of tools for interacting with these arrays. It is the cornerstone of Python's data science toolset. Therefore, it is a must-have for anybody dealing with numerical data and scientific computing in Python.

Here are examples of vector and matrix creation:

create vectors and matrices using NumPy

```
import numpy as np

# Creating a vector (1D array)
vector = np.array([1, 2, 3])
print("Vector:\n", vector)

# Creating a matrix (2D array)
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("Matrix:\n", matrix)
```

```
Vector:
 [1 2 3]
Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

To solve the given system of linear equations using NumPy, we need to represent the coefficients of the variables in a matrix (A) and the constants in another matrix (B). Then, we can use the numpy.linalg.solve function to find the values of the variables (x, y, z).

Here is the solution in Python:

```
[7] import numpy as np

    A = np.array([[2, 3, 4],
                  [1, 2, 3],
                  [4, 1, 1]])

    B = np.array([20, 14, 9])

    variables = np.linalg.solve(A, B)
    x, y, z = variables

    print(f"x = {x}, y = {y}, z = {z}")
```

```
x = 0.9999999999999998, y = 2.0000000000000044, z = 2.999999999999997
```

x = 1, y = 2 and z = 3

a) What are pandas' series and pandas dataframes? Explain them with examples. Create a 5 x 5 matrix (A) of random numbers using NumPy, then convert it into pandas data frames. Access the elements of the first rows.

Pandas is a well-known Python library for manipulating and analysing data. Pandas provide two essential data structures: Series and DataFrames.

A Pandas Series is a one-dimensional array storing any data type (integers, strings, Python objects, etc.). The axis identifiers are referred to collectively as the index. It is comparable to a column in a spreadsheet, a database field, or a mathematical vector. You can create a Series from a Python list.

Example:

```python
import pandas as pd

data = [3, 5, 7, 9, 11]
index = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index)

print(series)
```

```
a     3
b     5
c     7
d     9
e    11
dtype: int64
```

Pandas DataFrame: A DataFrame is a two-dimensional labelled data structure with columns that may be of varying data types. It can be compared to a spreadsheet, SQL table, or dictionary of Series objects. It is typically the most frequently employed pandas object. Example:

```python
import pandas as pd

data = {
    "A": [1, 2, 3],
    "B": [4, 5, 6],
    "C": [7, 8, 9],
}

df = pd.DataFrame(data)

print(df)
```

```
   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9
```

The code below generates a 5 x 5 matrix (A) of random numbers using NumPy and transforms it into a pandas Data Frame. To access the elements of the first row, we can use the iloc function in pandas and access the elements of the first row:

```python
import numpy as np
import pandas as pd

# Create a 5x5 matrix of random numbers
A = np.random.rand(5, 5)

# Convert the NumPy array to a pandas DataFrame
df = pd.DataFrame(A)

# Print the DataFrame
print(df)

# Access the elements of the first row
first_row = df.iloc[0]

print("\nElements of the first row:")
print(first_row)
```

```
          0         1         2         3         4
0  0.469587  0.391308  0.828228  0.374084  0.860221
1  0.188266  0.845390  0.320804  0.706181  0.296998
2  0.300442  0.298254  0.515054  0.656592  0.713732
3  0.695835  0.329203  0.865285  0.374608  0.119151
4  0.867509  0.792477  0.357063  0.674170  0.878684

Elements of the first row:
0    0.469587
1    0.391308
2    0.828228
3    0.374084
4    0.860221
Name: 0, dtype: float64
```

Question 3

Explain the role of the following in the context of artificial neural networks:

a) Activation Function

In an artificial neural network, the activation function plays a crucial role in determining the output of a neuron (or "node"). As an argument, it receives the weighted sum of the inputs and bias and generates an output that serves as input to the next network layer. The activation function is designed to induce nonlinearity into the neuron's output. This is crucial because most data in the actual world is nonlinear, and we want neurons to learn from such data.

Without a non-linear activation function, our neural network would operate identically to a single-layer perceptron regardless of the number of layers, as the sum of these layers would result in a linear function.

Here are several frequent activation functions:

- Sigmoid Function: The Sigmoid function is a type of activation function that has been used extensively in neural networks. The output of the sigmoid function is employed to transform the neuron's direct output into a value between 0.0 and 1.0. It was frequently implemented in the output layer of binary classification networks.
- ReLU (Rectified Linear Unit) Function: Currently, the ReLU (Rectified Linear Unit) function is the most popular activation function in the field of deep learning. It returns x if x is positive and 0 if it is negative. Both the function and its derivative have monotonic behaviour. In practise, however, it functions well and trains quickly.
- The tanh (hyperbolic tangent) function is comparable to the sigmoid function but superior. The tanh function has a range of (-1 to 1). Tanh shares the same characteristics as the sigmoid, namely that it is nonlinear. The tanh function is, therefore, a scaled sigmoid function.
- Softmax Function: The softmax function is typically implemented in the output layer of a neural network for multi-class classification problems. It transforms a K-dimensional vector of arbitrary real values into a K-dimensional vector of real values in the interval [0, 1] that sum to 1.

Each activation function has its benefits and drawbacks, and choosing which to use depends on the model's specific requirements and the problem we are attempting to solve.

Activation functions serve two primary functions:

- Real-world data are frequently non-linear, meaning they cannot be accurately depicted using a straight line. Activation functions introduce nonlinearity into the network, allowing it to model intricate patterns and capture complex relationships between input features. Without nonlinearity, neural networks would only be able to acquire linear relationships, rendering them ineffective at solving complex problems.
- Activation functions serve to normalise the output of a neuron, ensuring that it falls within a predetermined range. This normalisation contributes to the stability of the learning process

by preventing the propagation of extreme values through the network, which could result in unstable gradients during training.

In conclusion, activation functions play a crucial role in artificial neural networks by introducing nonlinearity and normalising neuronal output. They allow the network to learn complex patterns and accurately predict various problems, including image classification and natural language processing. The choice of activation function depends on the specific problem, network architecture, and desired results.

b) Cost Function

In an artificial neural network, the cost function (also known as the loss function or error function) measures the distance between the network's predictions and the actual values or ground truth. In other terms, it measures the error of the network's prediction. The objective of neural network training is to minimise this cost function. The cost function is used in backpropagation to alter the weights and biases, which is a crucial step in the neural network's learning procedure. The network's weights and biases are adjusted during training to minimise the cost function's value. Typically, this is achieved using optimisation algorithms such as Gradient Descent.

Here are some common examples of cost functions:

- Mean Squared Error (MSE): Regression problems frequently employ this statistic. It computes the square of the difference between the predicted and actual values before averaging it across the number of data points. It penalises severely for significant errors.
- Cross-Entropy Loss: This is utilised frequently in classification issues. Log loss is calculated in binary classification based on the predicted probability of the actual class. It computes the sum of the log of the predicted probabilities for each class (categorical cross-entropy) for multi-class classification.
- Hinge Loss: This is utilised for "maximum margin" classification, specifically for support vector machines. The hinge loss of the prediction y is defined as max(0, 1 ty) for an intended output of t = 1 and a classifier score of y.
- Log-Cosh Loss: This less common loss function works well for regression problems and is less sensitive to outliers than the Mean Squared Error.

Each cost function has its own benefits and drawbacks, and the choice of which to employ depends on the model's specific requirements and the type of problem you're attempting to solve.

The cost function fulfils multiple vital functions:

- The cost function measures the efficacy of the neural network by calculating the difference between its predictions and the actual targets. A lesser value for the cost function indicates superior performance, while a higher value indicates inferior performance. By contrasting the cost function values during training, it is possible to evaluate the learning process and determine when the model has attained its optimal state.
- Neural networks guide the learning process by modifying their weights and biases to minimise the cost function. The gradient of the cost function concerning the model’s parameters provides information on how to adjust these parameters to reduce the error. The learning algorithm uses this gradient information, such as gradient descent or a variant thereof, to iteratively update the model's parameters until the cost function converges to a minimum value.

- Regularization: Cost functions may also include regularisation terms to prevent overfitting, which occurs when the model becomes excessively complex and captures noise in the training data. Regularisation techniques, such as L1 and L2 regularisation, augment the cost function with a penalty term proportional to the magnitude of the model's weights. This penalty discourages excessively large weights, resulting in a simpler and more generalisable model for unobserved data.

The cost function is an essential component of artificial neural networks because it quantifies the model's performance and governs the learning process during training. By minimising the cost function, neural networks can enhance their precision and generalizability. Different cost functions are adequate for various categories of problems. The selection of the appropriate cost function is contingent on the nature of the problem, the network architecture, and the intended performance characteristics. Regularisation strategies can also be incorporated into the cost function to prevent overfitting and improve the model's ability to generalise to unobserved data.

By grasping the role of the cost function in artificial neural networks and selecting the appropriate cost function for a particular problem, it is possible to construct more accurate, robust, and efficient models that perform well on training data and previously unseen data.

c) Gradient Descent

Gradient Descent is an optimisation algorithm that minimises a neural network's cost function. It serves a crucial function in the network's learning process. The objective of gradient descent is to adjust the network's weights and biases in the direction that minimises the cost function the most. This direction is determined by the negative of the cost function's gradient with respect to the weights and biases.

A detailed explanation of how gradient descent operates within a neural network follows:

1. Initialise Weights and Biases: Random values are used to initialise the network's weights and biases.
2. Calculate Cost: Using the current weights and biases, the network makes predictions, and the cost function is calculated based on these predictions and the actual values.
3. Calculate Gradient: The gradient of the cost function is computed with respect to each weight and bias. The gradient is a vector pointing in the direction of the function's steepest ascent, and its magnitude indicates the rate of ascent.
4. The weights and biases are then updated by subtracting the gradient times a learning rate from the gradient. The learning rate is a hyperparameter that controls the update step size. A slower learning rate could delay the process of learning, while a faster learning rate could cause the process to diverge.
5. Iterate: Repeat steps 2 through 4 until the cost function converges to its minimal value.

Numerous gradient descent variants exist, such as batch gradient descent, stochastic gradient descent, and mini-batch gradient descent. These variants utilise different quantities of data to compute the gradient of the cost function.

Batch Gradient Descent: It computes the gradient of the cost function at each stage using the entire training dataset. It is computationally inefficient and impractical for large datasets that cannot fit in memory.

Stochastic Gradient Descent (SGD): It computes the gradient at each stage using only a single training example. SGD is quicker and can be used for online learning, but its updates have greater variance, causing the cost function to vacillate significantly.

Mini-Batch Gradient Descent: A compromise between SGD and batch gradient descent. The gradient is computed using a mini batch of n training examples at each phase. It reduces update variance and efficiently uses highly optimised matrix operations in the backend to execute computations.

Gradient descent aims to discover the optimal weights and biases that minimise the cost function, thereby improving the neural network's predictions.

Gradient descent fulfils multiple crucial functions in artificial neural networks:

- Neural networks learn by modifying their weights and biases to minimise the cost function, quantifying the difference between the predicted output and the actual objective. Gradient descent offers a methodical approach for updating these parameters to minimise error, optimising the model's performance.
- Convergence to a minimum: The gradient descent algorithm moves the model's parameters iteratively in the direction of the steepest decline in the cost function. This iterative process continues until convergence, which occurs when the cost function reaches a minimum value or when its change falls below a specified threshold. This process allows the algorithm to converge to a local or global minimum of the cost function, resulting in a trained neural network that can make accurate predictions.
- Gradient descent is a general optimisation algorithm applicable to various cost functions, including mean squared error, cross-entropy loss, and hinge loss. It is appropriate for various categories of problems and network architectures due to its adaptability. The learning rate determines the parameter updates' step size and impacts the algorithm's convergence.

Gradient descent is a fundamental optimisation algorithm in artificial neural networks, responsible for minimising the cost function and enhancing the model’s performance. Its adaptability to different cost functions, network architectures, and many variants and enhancements make it a versatile and potent instrument for training neural networks to make accurate and robust predictions.

Question 4

a) What is linear regression analysis? Discuss it with an example using the sklearn library. Predict the price of a house in the area [100]?

The data is given as follows: Area [20, 40, 80, 120, 150, 200] corresponding prices listed in thousands INR [15, 28, 55, 95, 136, 185]

The statistical technique of linear regression is utilised for predictive analysis. Modelling the relationship between a dependent variable and one or more independent variables using a linear approach. Simply put, it attempts to fit the data points to a line (in 2D space) or a hyperplane (in higher dimensions) that best suits them. In machine learning, it is a popular algorithm for predicting continuous output values, like the price of a house based on its area.

To illustrate linear regression with an example using the sklearn library, we will predict the price of a property in the area [100] using the provided data. The data includes the dwelling areas and their associated prices in thousands of Indian Rupees (INR).

We will begin by importing the necessary libraries and preparing the data:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Data preparation
area = np.array([20, 40, 80, 120, 150, 200]).reshape(-1, 1)
prices = np.array([15, 28, 55, 95, 136, 185])
```

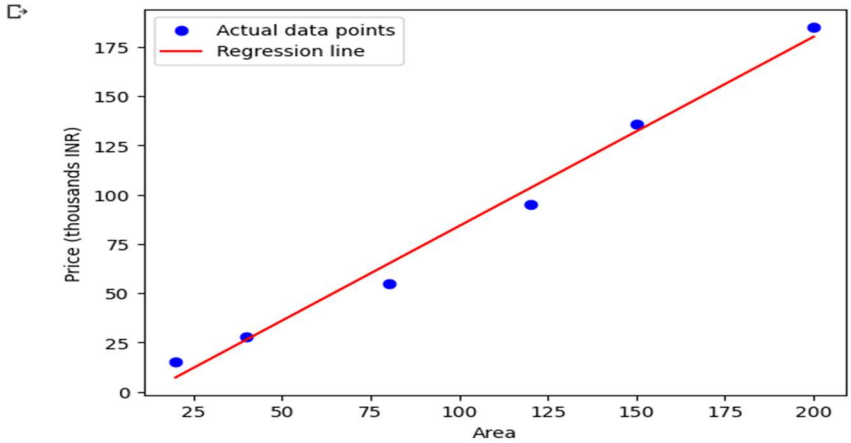
Once we have imported the essential libraries (sklearn's numpy and LinearRegression). The input data (areas) are reshaped to have a 2D form as needed by the linear regression model. We next build a linear regression object, fit it to our data, and use the predict function to forecast the price of a 100-square-foot home.:

```
# Create the linear regression model
model = LinearRegression()

# Fit the model to the data
model.fit(area, prices)

# Make a prediction for area [100]
predicted_price = model.predict([[100]])
print(f"Predicted price for a house in area [100]: {predicted_price[0]:.2f} thousands INR")
```

```
Predicted price for a house in area [100]: 84.07 thousands INR
```

To visualise the linear relationship between area and prices, the data points can be plotted along with the regression line:

```
plt.scatter(area, prices, color='blue', label='Actual data points')
plt.plot(area, model.predict(area), color='red', label='Regression line')
plt.xlabel('Area')
plt.ylabel('Price (thousands INR)')
plt.legend()
plt.show()
```



The linear regression model predicts that the price of a home in the area [100] will be approximately 84,07 thousand INR. The scatter plot would display the data points and regression line, illustrating the linear relationship between area and price.

We must note that linear regression is sensitive to outliers and presupposes a linear relationship between the variables. Before using the model to make predictions, it is essential to validate its assumptions and evaluate its performance using appropriate evaluation metrics, such as mean squared error, R-squared, or adjusted R-squared.

b) What is binary classification with logistic regression? What is a logistic function and its properties? Find the relation between logistic regression and linear regressions and explain its physical significance.

Binary classification with logistic regression uses logistic regression to predict one of two possible class labels for a given input (e.g., 0 or 1, positive or negative). Logistic regression is a statistical technique characterising the association between a binary dependent variable and one or more independent variables. It extends the concept of linear regression by modelling the probability of an observation belonging to a specific class using a logistic function. It is typically employed when the dependent variable is binary, that is, it can only take two values, such as "yes" or "no." Logistic regression differs from linear regression in that linear regression predicts continuous outcomes.

The logistic function (also referred to as the sigmoid function) is the central element of logistic regression. The following is its formula:

$\sigma(x) = 1 / (1 + e^{(-x)})$

x is the input value, and e is the natural logarithm base (approximately 2.71828). The logistic function possesses several crucial characteristics:

- Range: The logistic function produces values between 0 and 1, making it appropriate for modelling probabilities.
- The function is monotonically increasing, meaning that an increase in x will result in an increase in f(x).
- S-shape: The logistic function has an S-shaped curve, which enables it to encapsulate the nonlinear relationship between the independent variables and the probability of the dependent variable belonging to a particular class.
- Differentiability: The logistic function is differentiable, crucial for gradient-based optimisation algorithms such as gradient descent, which are used to train logistic regression models. we can use calculus (specifically gradient-based methods) to find the function's minimum and maximum values.

Logistic and linear regression are related in that they are both statistical models used for prediction. However, they are used for different types of prediction tasks. The relationship between logistic and linear regression resides in their fundamental assumptions and output transformations. Linear regression assumes that the dependent variable is continuous and models the relationship between the independent and dependent variables as a linear combination of the features. In contrast, logistic regression represents the likelihood of an observation belonging to a particular class by applying a logistic function to a linear combination of the features.

The relationship between logistic regression and linear regression lies in their model structure. In linear regression, the output is modelled as a linear combination of the input variables:

$y = \beta0 + \beta1 * x1 + \beta2 * x2 + ... + \betan * xn$

In logistic regression, the same linear combination of input variables is used, but it is transformed by the logistic function to model the probability of the binary outcome:

$p(Y=1) = 1 / (1 + e^{-(\beta 0 + \beta 1 * x1 + \beta 2 * x2 + ... + \beta n * xn)})$

Linear regression is used for regression tasks where the goal is to predict a continuous output variable. Logistic regression, on the other hand, is used for binary classification tasks, where the goal is to predict which of two classes an input belongs to.

The logistic regression model uses the logistic function to squeeze the output of a linear equation between 0 and 1. The underlying linear equation is like that used in linear regression, but the output is passed through the logistic function to ensure that it is a valid probability.

Physically, the distinction between logistic and linear regression can be explained as follows:

- Logistic regression is designed for problems of binary classification in which the objective variable has only two possible outcomes. In contrast, linear regression is utilised for regression problems where the objective variable is continuous.
- Logistic regression produces the probability that an observation belongs to a particular class, which can then be the threshold to acquire the predicted class label. The output of linear regression is the predicted value for the objective variable.
- Decision boundary: Logistic regression models produce a linear decision boundary delineating the two classes in the feature space. This results from the logistic function converting a linear combination of characteristics into a probability.

The physical significance of logistic regression lies in its ability to provide probabilities and classify new samples using continuous and discrete measurements. The coefficients of the logistic regression algorithm can be interpreted as the change in the log odds of the output variable for a one-unit change in the input variable.

In conclusion, binary classification with logistic regression predicts one of two possible class labels for a given input. It extends the concept of linear regression by employing a logistic function to represent the probability of an observation belonging to a particular class. The range, S-shape, and differentiability of the logistic function make it suitable for modelling probabilities and learning non-linear relationships between features and class labels. The primary differences between logistic and linear regression are the nature of the dependent variable (categorical for logistic regression and continuous for linear regression), the transformation applied to the output (logistic function in logistic regression), and the decision boundary (linear in linear regression and non-linear in logistic regression).

Question 5

a) Explain the concept of Transfer Learning.

Transfer learning is a machine learning technique that utilises a previously trained model as a starting point for a distinct but related problem. For instance, if you have a model that has been pre-trained for image classification tasks, you can use the learned features (weights and biases) as the initial settings for a new model being trained for a related task. This can significantly reduce training time and enhance performance, particularly when the new task has limited labelled data.

Transfer learning is utilised extensively in deep learning for tasks such as image classification, natural language processing, and speech recognition. It is especially valuable when there are few labelled data for the new assignment. If a model trained on a specific task has learned some general features of the data, these features can be used by a new model to solve a related task, according to the concept of transfer learning.

Transfer learning is based on the premise that the features learned by a neural network while solving one problem can be applied to solve other problems with a similar underlying structure or characteristics. By leveraging these learned features, a model can perform better on the new task and converge quicker than if trained from zero.

The basic assumption underlying transfer learning is that if a model has been trained on an extensive and varied enough dataset, it should have previously learnt relevant features or patterns that may be utilised in other related tasks. When opposed to training a model from the start, this may save time and money, and it can also help overcome issues associated with insufficient or unbalanced data in the new job.

This technique can be applied to various machine learning models, including deep learning models such as artificial neural networks and reinforcement models. The objective is to utilise the knowledge of a previously trained model while conducting a different task.

Transfer learning is commonly used in deep learning, particularly in the field of computer vision and natural language processing, where pre-trained models like ImageNet for image classification or BERT for text analysis have demonstrated significant improvements in performance on various tasks.

Transfer learning can be implemented in various methods based on the similarity between the source and target tasks and the neural network architectures involved. Typical strategies include:

- Feature extractor: We use a pre-trained neural network as a fixed feature extractor. The network's last layer(s) is removed, and the remaining layers are used to convert the input data into a more condensed and meaningful representation. This new representation can be used as input to a simplified classifier (e.g., logistic regression, support vector machine) or another neural network trained specifically for the intended task.
- Fine-tuning: In this technique, a pre-trained neural network is used as a starting point, and some or all its layers are modified to accommodate the target task. Typically, this entails initialising the network with the pre-trained weights and then training the network on the new task with a slower learning rate to prevent substantial changes in the learned features. Depending on the similarity between source and target tasks and the intended level of adaptation, fine-tuning can be performed on the entire network or a subset of layers.
- In some instances, the source and target tasks may have distinct input distributions, which can hinder the transferability of the acquired features. Domain adaptation techniques, such as adversarial training or maximum mean discrepancy, can align the distributions of the source and target domains, facilitating more effective transfer learning.

In computer vision, for instance, deep convolutional neural networks (CNNs) trained on large-scale datasets such as ImageNet have been shown to acquire generic features that can be applied to various tasks. Using the pre-trained weights of these networks as a starting point, researchers can attain state-of-the-art performance on various tasks, including object detection, segmentation, and classification, with substantially less data and training time.

Transfer learning is a powerful technique in artificial neural networks that enables the knowledge acquired from one task to be utilised to enhance the performance of a related task. This method is especially useful in situations with limited labelled data or computational resources. Transfer learning can be implemented in numerous ways, including feature extraction, domain adaptation, and fine-tuning, depending on the similarity between the source and target tasks and the intended level of adaptation.

Transfer learning offers the following benefits: In other forms of learning, you must construct a model with no prior knowledge. Transfer learning provides a more advantageous starting point; one can perform specific duties without training. Transfer learning provides a greater learning rate during training because the problem has already been trained for a similar task. With a superior starting point and a faster learning rate, transfer learning enables a machine learning model to converge on a higher performance level, resulting in more accurate output. Due to the use of a pre-trained model, the training can accomplish the intended performance more quickly than traditional learning methods.

Transfer learning has the benefit of decreasing the training time for a neural network model and can result in lower generalisation errors. However, transfer learning may be different from conventional learning models. The impact of transfer learning can only be determined once the target model is developed.

b) List a few advantages and disadvantages of using the hugging face pipeline as introduced in Module 6, Lesson 1, Video/Topic 6 over the custom-trained Neural Network model for sentiment classification.

The Hugging Face pipeline is an easy-to-use, high-level API for applying transformers models. It abstracts away the complex intricacies of importing models, handling tokenisation, and executing the model, enabling developers to use state-of-the-art models for tasks such as sentiment analysis, text generation, and translation with only a few lines of code.

Utilising the Hugging Face pipeline has these benefits:

- The pipeline provides a straightforward, high-level API. This makes it accessible to developers who might not have a solid grasp of the underlying models.
- Hugging Face provides many pre-trained models that can be utilised immediately for various tasks. Compared to training a model from inception, this can save significant time and computational resources.
- Hugging Face keeps abreast of the most recent developments in NLP and adds new state-of-the-art models to its library regularly.
- Hugging Face has a large and active community which provides support. This means that if you encounter problems or have concerns, you will likely find assistance promptly.

Negative aspects of the Hugging Face pipeline:

- While the pipeline is simple to use, it provides a different level of flexibility than custom training a model. If you need to modify the model architecture or training procedure, you may find the pipeline to be restrictive.

- Overhead: The pipeline automatically manages many details, which can result in some overhead. A custom solution may be preferable if you work on a task where efficacy is crucial.
- Size of Models Hugging Face's pre-trained models can be quite large. This can be problematic if you are operating with limited resources.

Finally, whether we use the Hugging Face pipeline or a custom-trained neural network model for sentiment categorisation is determined by your individual goals, skills, and available resources. The Hugging Face pipeline might be a good fit if you need a fast, easy-to-implement solution with cutting-edge performance. On the other hand, a custom-trained neural network model may be a preferable solution if you want greater control and customisation or are worried about dependencies and computing resources.

c) In the Hands-On exercise in Module 5, Lesson 2, Videos/Topics 2 and 3, we train a sentiment analysis model (ANN Classification using TensorFlow.ipynb) using Neural Network. Extend the code in ANN Classification using TensorFlow.ipynb to implement the sentiment analysis using the hugging face transformer pipeline for sentiment analysis on the train.csv file.

Refer to the file wk5_ANN-Classification-using-TensorFlow EXTENDED.ipynb

d) Comment on the accuracy achieved using the hugging face transformer pipeline in comparison to the Countvectorizer and Neural Network based TensorFlow model.

The TensorFlow model achieved an accuracy of approximately 99.35% on the training data, indicating a high level of performance. The precision, recall, and f1-score were also high for both classes (0 and 1), suggesting that the model effectively classified positive and negative sentiments in the training data.

A comparison of the two models in respect of accuracy on the test data reveals:

Model	Accuracy	Precision	Recall	F1-score
TensorFlow (CountVectorizer + NN)	99.12%	0.99	0.99	0.99
Hugging Face Transformer PipeLine	90.77%	0.90	0.91	0.91

The TensorFlow model achieved an accuracy of approximately 99.12% on the test data, which is very close to the performance on the training data. This indicates that the model could generalise well and perform similarly on unseen data. The precision, recall, and f1-score were also high for both classes in the test data, similar to the training data.

Precision is a measurement of the proportion of accurate positive predictions. Recall is the proportion of genuine positive instances that were correctly identified. The F1-score is the harmonic mean of precision and recall, resulting in a singular metric that is proportional to both values.

Hugging Face Transformer pipeline: This model's accuracy on the test set was approximately 90.77 percent. Both classes' precision, recall, and F1-score are marginally inferior to those of the TensorFlow model. Nonetheless, they are still quite high, indicating that this model performs well, albeit not quite as well as the TensorFlow model. The CountVectorizer and Neural Network-based TensorFlow model appears to outperform the Hugging Face Transformer pipeline on the supplied test set regarding accuracy, precision, recall, and F1-score, even though both models perform well.

Question 6

a) Explain the reasons why CNN models are a preferred approach for building image classification models versus the ANN models.

Convolutional Neural Networks (CNNs) are a subtype of artificial neural networks (ANNs) designed specifically to process grid-like data, such as images. Convolutional Neural Networks (CNNs) are favoured over Artificial Neural Networks (ANNs) for creating image classification models due to several important factors that make CNNs more suited for processing image data. CNNs have become the preferred method for developing image classification models due to their distinctive architectural characteristics and the following advantages. Here are several advantages of CNNs over traditional ANNs for image classification tasks:

Local Connectivity: Pixels that are close together in an image are more likely to be semantically related (part of the same object) than far apart pixels. By connecting each neuron to only a local region of the input volume, CNNs reflect this property. In contrast, ANN neurons are entirely interconnected, which does not correspond to the local nature of images.

Translation Invariance: If an object in an image adjusts marginally, it does not alter the object's presence in the image. By utilising shared weights and biases across spatial locations, CNNs inherently implement this property. This implies that if a CNN learns a feature in one portion of an image, it can recognise it in any other portion. ANNs do not possess this quality.

Dimensionality Reduction: Images can have a high dimensionality (a 100x100 pixel image with RGB colour channels already has 30,000 dimensions). CNNs use pooling layers to gradually reduce the spatial extent of the data, thereby reducing the number of network parameters and computation. This makes training the network more manageable.

Feature Learning: CNNs can autonomously learn and enhance features directly from images with minimal preprocessing. This replaces the manual feature extraction techniques of the past. This is made possible by the convolutional layer, which scans images for local features like boundaries, corners, etc.

CNNs have been utilised to attain state-of-the-art performance on key image recognition benchmarks such as ImageNet.

State-of-the-art results: CNNs have inductive biases, such as translation equivariance and locality, that make them more suitable for image data. These biases enable CNNs to manage image variations and distortions better than ANNs.

Scalability with Large Datasets: CNNs can effectively manage larger datasets than ANNs. This is especially crucial when dealing with image data, which can be extremely large and multidimensional.

Self-Attention Mechanisms: Some CNN architectures include self-attention mechanisms, which enable the model to concentrate on various image regions when making predictions. This can be advantageous for intricate image classification duties.

Pre-training on Large Datasets: CNNs can be pre-trained on large datasets and then fine-tuned on smaller task-specific datasets. This strategy can result in superior performance, particularly when the task-specific dataset is limited.

Combining CNNs with Self-Attention: Combining CNNs with forms of self-attention has generated considerable interest. This can be accomplished by enhancing feature maps for image classification or by applying self-attention to the output of a CNN. This combination can produce promising computer vision results.

Handling Medium-Resolution Images: CNNs can process medium-resolution images, whereas this may only be the case for some ANNs.

While CNNs are typically preferred for image classification, this does not preclude using ANNs for this purpose. Depending on the difficulty of the task and the quantity of available data, an ANN may perform admirably. For duties involving large, complex images and a requirement for high precision, CNNs are typically the preferred option. The choice between CNNs and ANNs for image classification will depend on the specific requirements of our assignment, such as the size and nature of our dataset, the computational resources at our disposal, and the required level of accuracy.

CNNs are preferred over ANNs for image classification tasks due to their ability to capture spatial relationships between pixels, their parameter sharing and translation invariance properties, and their ability to learn hierarchical features. These characteristics make CNNs more efficient, robust, and accurate than traditional ANNs for image classification.

b) What are the advantages and disadvantages of using vision transformers for image classification?

Vision Transformers (ViTs) is a comparatively recent development in the field of artificial neural networks that have garnered prominence due to their remarkable performance on various computer vision tasks. They apply the transformer architecture, which was originally developed for natural language processing, to image data and, more particularly, to image classification tasks.

Here are the pros and cons of employing Vision Transformers for image classification:

Advantages:

Scalability: ViTs have demonstrated remarkable scalability, efficiently processing images of various sizes and resolutions. Larger datasets and more robust computational resources enable them to outperform conventional convolutional neural networks (CNNs). In addition, they are more scalable in terms of model size and dataset size, effectively utilising increased computational resources by scaling up the model size.

Global Context: Contrary to CNNs, which rely on local connectivity to acquire spatial features, ViTs have a global receptive field, allowing them to capture long-range dependencies and relationships among various image regions. This can lead to more precisely depicting and comprehending complex scenes and objects.

Simplified Architecture: ViTs have a simpler architecture than CNNs due to the absence of convolutional layers and other specialised components, including pooling layers. This simplified architecture makes ViTs easier to comprehend, implement, and modify.

Transfer Learning: ViTs have demonstrated robust transfer learning capabilities, with pre-trained models generalising well to a variety of downstream tasks, including object detection, semantic segmentation, and instance segmentation. They are more transferable and can be pre-trained on massive datasets and fine-tuned on subsequent tasks, resulting in excellent performance.

Data Efficiency: Vision Transformers are more data-efficient on large-scale datasets. Compared to CNNs, they perform well with fewer data points.

Disadvantages:

Computational Resources: ViTs can be computationally demanding, particularly for high-resolution images, because they require attention mechanisms that scale quadratically with the number of input elements (i.e., image patches). This makes them less suitable for deployment on devices with limited resources or in environments with strict latency requirements.

Reliance on Large Datasets and Pre-training: ViTs typically require large data sets and pre-training to achieve competitive performance. This pre-training requirement can restrict their applicability in scenarios with limited data and computational resources.

Limited Inductive Bias: ViTs have limited inductive bias compared to CNNs, designed to exploit images' spatial structure. This lack of inductive bias may lower the performance of certain image classification tasks, particularly when training data is limited.

Model Size: ViTs can be quite large, especially when scaled up to manage high-resolution images or to attain cutting-edge performance. The scale of the model can complicate memory requirements and deployment on devices or environments with limited resources.

Lack of Translation Invariance: Unlike CNNs, ViTs lack translation invariance, which can be detrimental to certain tasks.

Pre-processing Overhead: Vision transformers require images to be partitioned into non-overlapping regions and compressed into a sequence of tokens, which can result in some pre-processing overhead compared to traditional CNN-based methods.

Vision Transformers offer numerous advantages for image classification, including scalability, global context modelling, and transfer learning capabilities. Nonetheless, they have several limitations, such as high computational resource requirements, large model sizes, and preprocessing overhead. Whether vision transformers or other methods are used for image classification depends on the nature of the task, the available resources, and the desired performance.

c) In the Hands-On exercise in Module 6, Lesson 2, Videos/Topics 3 and 4, we train a CNN model and train a classifier for CIFAR 10 image classification. Extend that code to implement the image classification using a vision transformer (ViT) on the test images.

Refer to the File for code: IMG_Classification_VIT(F).ipynb

d) Comment on the accuracy achieved in comparison to the CNN model.

Comparing the performance of the Convolutional Neural Network (CNN) and the Vision Transformer (ViT), we can see that the CNN model achieved greater accuracy on the test set (72% vs. 52%).

The potential causes for this disparity are:

Model Architecture: CNNs and Vision Transformers handle image data in fundamentally distinct ways. CNNs utilise convolutional layers to exploit the spatial locality of image pixels and have become a popular choice for image classification tasks due to their efficiency and effectiveness. Vision Transformers, on the other hand, which have been effective in NLP tasks, deconstruct the image into a sequence of patches and process them as a sequence, enabling global comprehension of the image. To outperform CNNs, however, they require larger datasets and more computational capacity.

Dataset Size and Complexity: Size and Complexity of the Dataset Despite being a standard benchmark, the CIFAR-10 dataset is relatively modest and low-resolution. CNNs typically perform well on such data due to their ability to efficiently capture local characteristics. ViTs, on the other hand, typically perform better with larger, higher-resolution images in which global dependencies become more crucial. The relatively inferior performance of the ViT model may be attributable to the small size and simplicity of the CIFAR-10 dataset.

Pre-training: The ViT model has been pre-trained, which typically contributes to greater accuracy. Nevertheless, pre-training is typically performed on large, high-resolution datasets (e.g., ImageNet), which are quite distinct from CIFAR-10. The disparity between the pre-training data and the objective task data may be one of the causes of the ViT model's poor performance.

Both models exhibit symptoms of overfitting, as indicated by the discrepancy between training and validation accuracy. However, the overfitting in the ViT model appears to be more pronounced. Overfitting could be mitigated by incorporating regularisation (e.g., dropout or weight decay), augmenting data, or employing early stopping.

While the Vision Transformer is a powerful model and has achieved state-of-the-art results on many tasks, this does not necessarily imply that it will outperform CNNs on all types of datasets, particularly on small and low-resolution datasets such as CIFAR-10. Less complex models, such as CNNs, can be more effective and efficient for these types of tasks. When selecting a model architecture, it is essential to grasp the characteristics of our specific task and data set, as demonstrated by these results.

References:

Rose, D. (2018, May 15). Artificial Intelligence for Business: What You Need to Know about Machine Learning and Neural Networks.

Monti, F. (2011). Regression Analysis of Count Data (Vol. 53). Cambridge University Press.

Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied Logistic Regression (Vol. 398). John Wiley & Sons.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Flach, P. (2012). Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

Müller, A. C., & Guido, S. (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media, Inc.

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359.

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? Advances in Neural Information Processing Systems, 27, 3320-3328.

Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1717-1724.

Razavian, A. S., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: An astounding baseline for recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 512-519.

Tzeng, E., Hoffman, J., Saenko, K., & Darrell, T. (2017). Adversarial discriminative domain adaptation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 7167-7176.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 1, 4171-4186.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. arXiv preprint arXiv:1907.11692.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 38-45

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097-1105.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770-778.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. Proceedings of the International Conference on Learning Representations.

Touvron, H., Cord, M., Girard, M., Sablayrolles, A., & Jégou, H. (2021). Training data-efficient image transformers & distillation through attention. Proceedings of the International Conference on Learning Representations.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-end object detection with transformers. Proceedings of the European Conference on Computer Vision, 2137-2150.

Stankovic, S., Stankovic, M., & Kostic, M. (2021). Robust Design of Artificial Neural Networks by Taguchi Method. Entropy, 23(7), 854.