Experiment 2: Exploratory Data Analysis on E-Commerce Data

Exploratory Data Analysis:
Exploratory Data Analysis (EDA) is the process of visualizing and analysing data to extract insights from it. In other words, EDA is the process of summarizing important characteristics of data in order to gain better understanding of the dataset.
1. Maximize insight into a data set.

2. Uncover underlying structure.

3. Extract important variables.

4. Detect outliers and anomalies.

5. Test underlying assumptions.

6. Develop parsimonious models.

7.Determine optimal factor settings.

Dataset features:
InvoiceNo *(invoice_num)*: A number assigned to each transaction
StockCode *(stock_code)*: Product code
Description *(description)*: Product name
Quantity *(quantity)*: Number of products purchased for each transaction
InvoiceDate *(invoice_date)*: Timestamp for each transaction
UnitPrice *(unit_price)*: Product price per unit
CustomerID (*cust_id*): Unique identifier each customer
Country (*country*): Country name

Experiment 3: Download the any dataset from UCI or Data.org or from any other data repositories and Implement Single and multilayer perceptron on a dataset.

A. MLP Classifier

B. MLP Regressor

Display the loss graph for the same. Also generate classification report in terms of confusion matrix, precision, recall, and f1 score.

After completion of this experiment students will be able to:

1. · To learn about classification and regression
2. To learn MLP and backpropagation
3. To demonstrate and analyse the results

Aim: Download the any dataset from UCI or Data.org or from any other data repositories and Implement Single and multilayer perceptron on a dataset.

A. MLP Classifier

B. MLP Regressor

Display the loss graph for the same. Also generate classification report in terms of confusion matrix, precision, recall, and f1 score

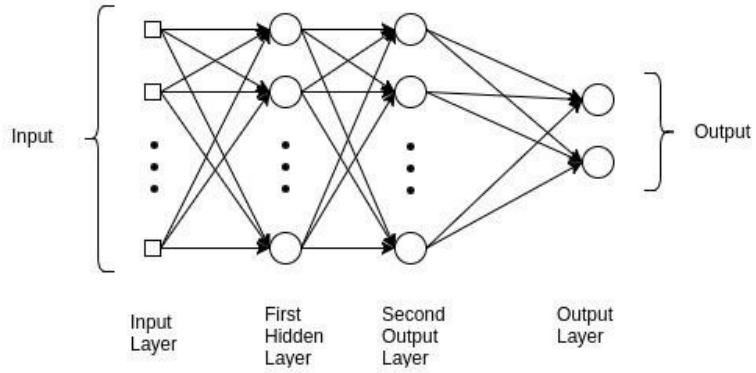
Theory:

Regression:
A regression problem is when the output variable is a real or continuous value, such as "salary" or "weight". Many different models can be used, the simplest isthe linear regression. It tries to fit data with the best hyper-plane which goes through the points.

Classification:
A classification problem is when the output variable is a category, such as "red"or "blue" or "disease" and "no disease". A classification model attempts to drawsome conclusion from observed values. Given one or more inputs a classificationmodel will try to predict the value of one or more outcomes.For example, when filtering emails "spam" or "not spam", when looking at transaction data, "fraudulent", or "authorized".

Multilayer Perceptron:
In the Multilayer perceptron, there can more than one linear layer (combinationsof **neurons**). If we take the simple example the three-layer network, first layer will be the *input layer* and last will be *output layer* and middle layer will be called *hidden layer.* We feed our input data into

the input layer and take the output from the output layer. We can increase the number of the hidden layer as much as we want, to make the model more complex according to our task.



BackPropagation Algorithm:

The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases).

In other words, **backpropagation aims to minimize the cost function by adjusting network's weights and biases.** The level of adjustment is determinedby the gradients of the cost function with respect to those parameters.

Activation function

Activation functions also known non- linearity, describe the input-output relations in a non-linear way. This gives the model power to be more flexible indescribing arbitrary relations. Here are some popular activation functions Sigmoid, Relu, and TanH. I will describe these in my next blog.

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Precision = TP/TP+FP

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label? Wehave got recall of 0.631 which is good for this model as it's above 0.5.

Recall = TP/TP+FN

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitivelyit is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall. In our case, F1 score is 0.701.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)Confusion matrix

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized withcount values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification modelis confused when it makes predictions. It gives us insight not only into the errorsbeing made by a classifier but more importantly the types of errors that are being made.

	Class 1 Predicted	*Class 2 Predicted*
Class 1 Actual	**TP**	**FN**
Class 2 Actual	**FP**	**TN**

Conclusion: Thus we have successfully completed the implementation of Multilayer perceptron.

Experiment 4: Develop a Bayesian classifier

After completion of this experiment students will be able to:

1. To learn bayes theorem
2. To implement Bayesian classifier

Aim: Develop a Bayesian classifier on a dataset

Theory:

Bayes Theorem

Bayes' Theorem is a way of finding a probability when we know certain otherprobabilities.

The formula is:

P(A|B) = *P(A) P(B|A)***P(B)**

Which tells us:	how often A happens *given that B happens*, written **P(A|B)**,
When we know:	how often B happens *given that A happens*, written **P(B|A)**
	and how likely A is on its own, written **P(A)**
	and how likely B is on its own, written **P(B)**

Bayes Classifier with example

In machine learning, **naïve Bayes classifiers** are a family of simple "probabilisticclassifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models.[1] But they could be coupled with Kerneldensity estimation and achieve higher accuracy levels.

Naive Bayes is a simple technique for constructing classifiers: models that assignclass labels to problem instances, represented as vectors of feature values,where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based ona common principle: all naive Bayes classifiers assume that the value of a
particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

Experiment 5: Prepare the Correlated dataset of your own example or Download the any one of the UCI ML data Set in which find the correlated data and Find the best fit line for the data.

Aim: Prepare the Correlated dataset of your own example or Download the any one of the UCI ML data Set in which find the correlated data and Find the best fit line for the data

Theory: Linear regression is a statistical approach for modelling relationship between a dependent variable with a given set of independent variables.

Simple linear regression is an approach for predicting a **response** using a **single feature**.

It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as afunction of the feature or independent variable(x).

Multiple linear regression attempts to model the relationship between **two or more features** and a response by fitting a linear equation to observed data.
Clearly, it is nothing but an extension of Simple linear regression.

Logistic regression:

Logistic regression is another technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems (problems with two class values). In this post you will discover the logistic regression algorithm for machine learning.
Logistic regression is named for the function used at the core of the method, the logistic function.
The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S- shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.
1 / (1 + e^-value)
Where e is the base of the natural logarithms (Euler's number or the EXP() function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformedinto the range 0 and 1 using the logistic function.

Correlated datasets: They follow the property of Correlation

Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate together. A positive correlation indicates the extent to which those variables increase or decrease in parallel; a negative correlation indicates the extent to which one variable increases as the other decreases. A correlation coefficient is a statistical measure of the degree to which changesto the value of one variable predict change to the value of another. When the fluctuation of one variable reliably predicts a similar fluctuation in another variable, there's often a tendency to think that means that the change in one causes the change in the other. However, correlation does not imply causation. There may be, for example, an unknown factor that influences both variables similarly.

Experiment 6: Using inbuilt dataset of Breast cancer from scikit learn Implement PCA algorithm

After completion of this experiment students will be able to:

- 1.To learn about dimensionality reduction techniques
2. To implement principle component analysis.

Aim: Using inbuilt dataset of Breast cancer from scikit learn Implement PCA algorithm

Theory:

Curse of Dimensionality

The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. The expression was coined by Richard E. Bellman when considering problems in dynamic programming.[1][2]

Cursed phenomena occur in domains such as numerical analysis, sampling, combinatorics, machine learning, data mining and databases. The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient.

PCA

Given a collection of points in two, three, or higher dimensional space, a "best fitting" line can be defined as one that minimizes the average squared distance from a point to the line. The next best-fitting line can be similarly chosen from directions perpendicular to the first. Repeating this process yields an orthogonal basis in which different individual dimensions of the data are uncorrelated. These basis vectors are called principal components, and several related procedures principal component analysis (PCA).

PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It is often used to visualize genetic distance and relatedness between populations. PCA is either done in the following 2 steps:

1. calculating the data covariance (or correlation) matrix of the original data
2. performing eigenvalue decomposition on the covariance matrix

LDA

Linear discriminant analysis (LDA), normal discriminant analysis (NDA), or discriminant function analysis is a generalization of Fisher's linear discriminant, a method used in statistics,

pattern recognition, and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification.
LDA is closely related to analysis of variance (ANOVA) and regression analysis, which also attempt to express one dependent variable as a linear combination of other features or measurements.

Difference between PCA AND LDA
LDA => This method identifies components (i.e., linear combination of the observed variables) that maximize class separation (i.e. between-class variance) when such prior information is available (i.e., supervised). E.g., you have a training set containing a variable specifying the class of each observation.
PCA => Aims to find components that account for maximum variance in the data (including error and within-variable variance). Unlike LDA, it does not take into account class membership (i.e., unsupervised), and is used when such information is not available. Importantly, both LDA and PCA do not require any prior notion of how the variables are related among themselves, and the resulting components can not be interpreted in terms of an underlying construct.

Conclusion
Thus we have successfully completed the implementation of the Principle component Analysis Algorithm.

Experiment 7: Implement decision tree classification/regression technique for given dataset

Developed model should be able to answer the given queries.

After completion of this experiment students will be able to:

1. To learn about decision trees
2. To implement decision trees and compare results

Aim: Implement decision tree classification/regression technique for given dataset

Developed model should be able to answer the given queries

Theory:

Decision Tree:

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Different types:

Decision trees used in data mining are of two main types:

- Classification tree analysis is when the predicted outcome is the class (discrete) to which the data belongs.
- Regression tree analysis is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

The term Classification And Regression Tree (CART) analysis is an umbrella term used to refer to both of the above procedures, first introduced by Breiman et al. in 1984.[4] Trees used for regression and trees used for classification have some similarities - but also some differences, such as the procedure used to determine where to split.[4]

Some techniques, often called ensemble methods, construct more than one decision tree:

- Boosted trees Incrementally building an ensemble by training each new instance to emphasize the training instances previously mis-modeled. A typical example is AdaBoost. These can be used for regression-type and classification-type problems.[5][6]
- Bootstrap aggregated (or bagged) decision trees, an early ensemble method, builds multiple decision trees by repeatedly resampling training data with replacement, and voting the trees for a consensus prediction.[7]

o A random forest classifier is a specific type of bootstrap aggregating

- Rotation forest – in which every decision tree is trained by first applying principal

component analysis (PCA) on a random subset of the input features.[8]
A special case of a decision tree is a decision list,[9] which is a one-sided decision tree, so that every internal node has exactly 1 leaf node and exactly 1 internal node as a child (except for the bottommost node, whose only child is a single leaf node). While less expressive, decision lists are arguably easier to understand than general decision trees due to their added sparsity, permit non-greedy learning methods[10] and monotonic constraints to be imposed.[11]
Notable decision tree algorithms include:

- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)[4]
- Chi-square automatic interaction detection (CHAID). Performs multi-level splits when computing classification trees.[12]
- MARS: extends decision trees to handle numerical data better.
- Conditional Inference Trees. Statistics-based approach that uses non- parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning.[13][14]

ID3 and CART were invented independently at around the same time (between 1970 and 1980)[citation needed], yet follow a similar approach for learning a decision tree from training tuples.

Conclusion

Thus we have successfully completed the implementation of decision tree classifier.

Experiment No 8: Implement SVM Classifier or Regression for given dataset

1. To learn SVM and kernel functions
2. To implement SVM classifier

Aim: Implement SVM Classifier or Regression for given dataset

Theory:

SVM: support-vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Kernel function

In machine learning, kernel methods are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets. For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified feature map: in contrast, kernel methods require only a user-specified kernel, i.e., a similarity function over pairs of data points in raw representation. Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick".[1] Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

Kernel trick

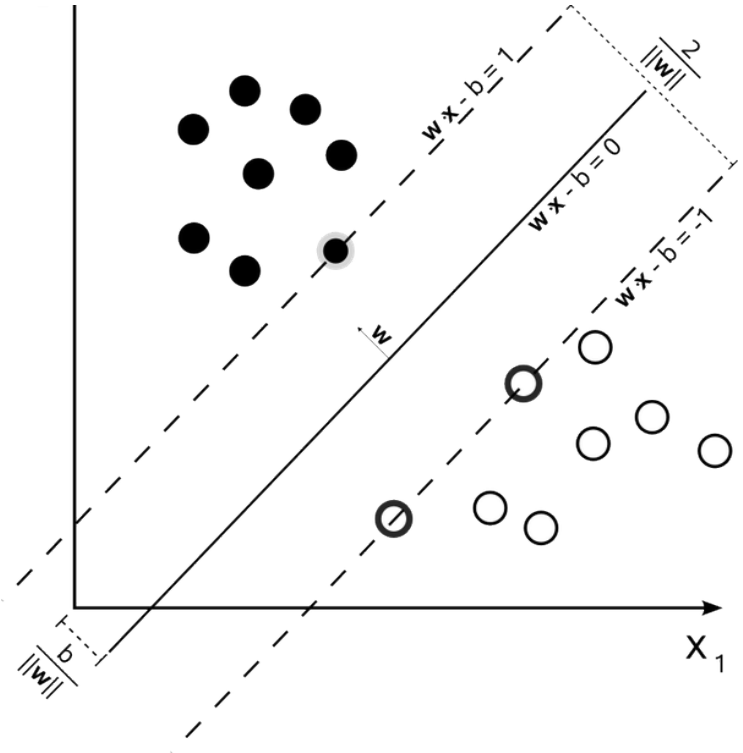
The kernel trick seems to be one of the most confusing concepts in statistics and machine learning; it first appears to be genuine mathematical sorcery, not to mention the problem of lexical ambiguity (does kernel refer to: a non-parametric way to estimate a probability density (statistics), the set of vectors v for which a linear transformation T maps to the zero vector — i.e. T(v) = 0 (linear algebra), the set of elements in a group G that are mapped to the identity element by a homomorphism between groups (group theory), the core of a computer operating system (computer science), or something to do with the seeds of nuts or fruit?).

The kernel trick also illustrates some fundamental ideas about different ways to represent data and how machine learning algorithms “see” these different data representations. And finally, the seeming mathematical sleight of hand in the kernel trick just begs one to further explore what it actually means.

Significance of SVM: It is capable of doing both classification and regression. In this post I'll focus on using SVM for classification. In particular I'll be focusing on non-linear SVM, or SVM using a non-linear kernel. Non-linear SVM means that the boundary that the algorithm calculates doesn't have to be a straight line. The benefit is that you can capture much more complex relationships between your datapoints without having to perform difficult transformations on your own. The downside is that the training time is much longer as it's much more computationally intensive.

Originally Answered: what is the purpose of support vector in SVM?

A support vector machine attempts to find the line that "best" separates two classes of points. By "best", we mean the line that results in the largest margin between the two classes. The points that lie on this margin are the support vectors.



Here we have three support vectors.

The nice thing about acknowledging these support vectors is that we can then formulate the problem of finding the "maximum-margin hyperplane" (the line that best separates the two classes) as an optimization problem that only considers these support vectors. So we can effectively throw out the vast majority of our data, which makes the classification process go much faster than, say,a neural network. More importantly, by presenting the problem in terms of the support vectors (the so-called dual form), we can apply what's called the kernel trick to effectively transform the SVM into a non-linear classifier.

Experiment 9: Implement K means algorithm for multidimensional data for Cars or Wine dataset from UCI repository

After completion of this experiment students will be able to:

· 1. To learn unsupervised learning

2. To implement K means algorithm.

Aim: Implement K means algorithm for multidimensional data for Cars or Wine dataset from UCI repository

Theory:

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labelled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.

Common clustering algorithms include:

- Hierarchical clustering: builds a multilevel hierarchy of clusters by creating a cluster tree
- k-Means clustering: partitions data into k distinct clusters based on distance to the centroid of a cluster
- Gaussian mixture models: models clusters as a mixture of multivariate normal density components
- Self-organizing maps: uses neural networks that learn the topology and distribution of the data
- Hidden Markov models: uses observed data to recover the sequence of states

Unsupervised learning methods are used in bioinformatics for sequence analysis and genetic clustering; in data mining for sequence and pattern mining; in medical imaging for image segmentation; and in computer vision for object recognition.

Clustering and its types:

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

Let's understand this with an example. Suppose, you are the head of a rental store and wish to understand preferences of your costumers to scale up your business. Is it possible for you to look at details of each costumer and devise a unique business strategy for each one of them? Definitely not. But, what you can do is to cluster all of your costumers into say 10 groups based on their purchasing habits and use a separate strategy for costumers in each of these 10 groups. And this is what we call clustering.

A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other, for example, a hierarchy of clusters embedded in each other. Clustering's can be roughly distinguished as:

- Hard clustering: each object belongs to a cluster or not
- Soft clustering (also: fuzzy clustering): each object belongs to each cluster to a certain degree (for example, a likelihood of belonging to the cluster)

Types:

- Strict partitioning clustering: each object belongs to exactly one cluster
- Strict partitioning clustering with outliers: objects can also belong to no cluster, and are considered outliers
- Overlapping clustering (also: alternative clustering, multi-view clustering): objects may belong to more than one cluster; usually involving hard clusters
- Hierarchical clustering: objects that belong to a child cluster also belong to the parent cluster
- Subspace clustering: while an overlapping clustering, within a uniquely defined subspace, clusters are not expected to overlap

K means algorithm

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way k means algorithm works is as follows:

1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

- Compute the sum of the squared distance between data points and all centroids.
- Assign each data point to the closest cluster (centroid).
- Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

The approach kmeans follows to solve the problem is called Expectation- Maximization.

Experiment 10: Download the famous dataset of iris and Implement KNN algorithm to predict the class to which these plants belong, Calculate the performance matric and compare the error rate with K value(K value range).

After completion of this experiment students will be able to:

· 1. To learn KNN algorithm

2. To implement KNN classifier.

Aim: Download the famous dataset of iris and Implement KNN algorithm to predict the class to which these plants belong, Calculate the performance matric and compare the error rate with K value(K value range).

Theory:

1. Lazy learners

Lazy learners simply store the training data and wait until a testing data appear. When it does, classification is conducted based on the most related data in the stored training data. Compared to eager learners, lazy learners have less training time but more time in predicting.

Eager learners

Eager learners construct a classification model based on the given training data before receiving data for classification. It must be able to commit to a single hypothesis that covers the entire instance space. Due to the model construction, eager learners take a long time for train and less time to predict.

Lazy learner:

1. Just store Data set without learning from it
2. Start classifying data when it receives Test data
3. So, it takes less time learning and more time classifying data

Eager learner:

1. When it receives data set it starts classifying (learning)
2. Then it does not wait for test data to learn
3. So, it takes long time learning and less time classifying data

KNN Algorithm

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood—

calculating the distance between points on a graph.

The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbours

3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels
8. If classification, return the mode of the K labels

Experiment 11: Implement CNN for MNIST/CIFAR10 dataset using tensorflow

After completion of this experiment students will be able to:

· 1. To learn basics of deep learning

2. To learn and implement CNN

Aim: Implement CNN for MNIST/CIFAR10 dataset using tensorflow

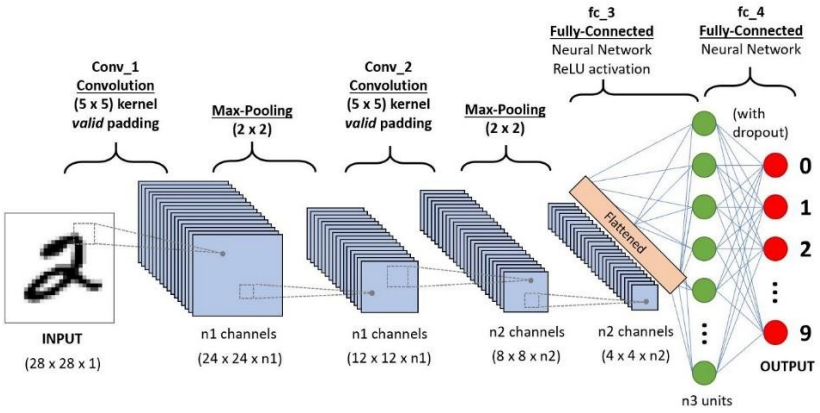
Theory:

Deep Learning

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.

Deep learning, a subset of machine learning, utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. The artificial neural networks are built like the human brain, with neuron nodes connected together like a web. While traditional programs build analysis with data in a linear way, the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach.

CNN Architecture



CNN WORKING

A Convolutional Neural Networks Introduction so to speak.

- Step 1: Convolution Operation

The first building block in our plan of attack is convolution operation. In this step, we will touch on feature detectors, which basically serve as the neural network's filters. We will also discuss feature maps, learning the parameters of such maps, how patterns are detected, the layers of detection, and how the findings are mapped out.

- Step 1(b): ReLU Layer

The second part of this step will involve the Rectified Linear Unit or ReLU. We will cover ReLU layers and explore how linearity functions in the context of Convolutional Neural Networks.

Not necessary for understanding CNN's, but there's no harm in a quick lesson to improve your skills.

- Step 2: Pooling

In this part, we'll cover pooling and will get to understand exactly how it generally works. Our nexus here, however, will be a specific type of pooling; max pooling. We'll cover various approaches, though, including mean (or sum) pooling. This part will end with a demonstration made using a visual interactive tool that will definitely sort the whole concept out for you.

- Step 3: Flattening

This will be a brief breakdown of the flattening process and how we move from pooled to flattened layers when working with Convolutional Neural Networks.

- Step 4: Full Connection

In this part, everything that we covered throughout the section will be merged together. By learning this, you'll get to envision a fuller picture of how Convolutional Neural Networks operate and how the "neurons" that are finally produced learn the classification of images.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

Load the dataset

```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

```
(50000, 32, 32, 3)
```

```
X_test.shape
```

```
(10000, 32, 32, 3)
```

Here we see there are 50000 training images and 1000 test images

```
y_train.shape
```

```
(50000, 1)
```

```
y_train[:5]
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

y_train is a 2D array, for our classification having 1D array is good enough. so we will convert this to now 1D array

```
y_train = y_train.reshape(-1,)
y_train[:5]
```

```
array([6, 9, 9, 4, 1], dtype=uint8)
```

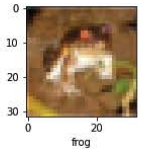
```
y_test = y_test.reshape(-1,)
```

```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

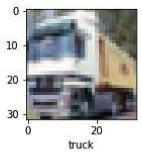
Let's plot some images to see what they are

```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

```
plot_sample(X_train, y_train, 0)
```




```
plot_sample(X_train, y_train, 1)
```



Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each value in the channel can range from 0 to 255. Hence to normalize in 0–>1 range, we need to divide it by 255

Normalizing the training data

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Build simple artificial neural network for image classification

```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 2s 2ms/step - loss: 1.8074 - accuracy: 0.3561
Epoch 2/5
1563/1563 [==============================] - 2s 1ms/step - loss: 1.6208 - accuracy: 0.4285
Epoch 3/5
1563/1563 [==============================] - 2s 2ms/step - loss: 1.5380 - accuracy: 0.4585
Epoch 4/5
1563/1563 [==============================] - 2s 2ms/step - loss: 1.4808 - accuracy: 0.4806
Epoch 5/5
1563/1563 [==============================] - 2s 2ms/step - loss: 1.4326 - accuracy: 0.4928
<tensorflow.python.keras.callbacks.History at 0x295ab873c10>
```

You can see that at the end of 5 epochs, accuracy is at around 49%

```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.45      0.53      1000
           1       0.72      0.46      0.56      1000
           2       0.33      0.46      0.39      1000
           3       0.36      0.25      0.29      1000
           4       0.44      0.37      0.40      1000
           5       0.34      0.46      0.39      1000
           6       0.56      0.47      0.51      1000
           7       0.39      0.67      0.50      1000
           8       0.64      0.60      0.62      1000
           9       0.59      0.53      0.55      1000

    accuracy                           0.47     10000
   macro avg       0.50      0.47      0.47     10000
weighted avg       0.50      0.47      0.47     10000
```

Now let us build a convolutional neural network to train our images

```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 2s 2ms/step - loss: 1.4407 - accuracy: 0.4810
Epoch 2/10
1563/1563 [==============================] - 2s 2ms/step - loss: 1.1084 - accuracy: 0.6109
Epoch 3/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.9895 - accuracy: 0.6574
Epoch 4/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.9071 - accuracy: 0.6870
Epoch 5/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.8416 - accuracy: 0.7097
Epoch 6/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.7847 - accuracy: 0.7262
Epoch 7/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.7350 - accuracy: 0.7448
Epoch 8/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.6941 - accuracy: 0.7574
Epoch 9/10
1563/1563 [==============================] - 2s 1ms/step - loss: 0.6516 - accuracy: 0.7731
Epoch 10/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.6187 - accuracy: 0.7836
<tensorflow.python.keras.callbacks.History at 0x296555783d0>
```

With CNN, at the end 5 epochs, accuracy was at around 70% which is a significant improvement over ANN. CNN's are best for image classification and gives superb accuracy. Also computation is much less compared to simple ANN as maxpooling reduces the image dimensions while still preserving the features

```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.9022 - accuracy: 0.7028
[0.9021560549736023, 0.7027999758720398]
```

```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

```
array([[4.3996371e-04, 3.4844263e-05, 1.5558505e-03, 8.8400185e-01,
        1.9452239e-04, 3.5314459e-02, 7.2777577e-02, 6.9044131e-06,
        5.6417785e-03, 3.2224660e-05],
       [8.1062522e-03, 5.0841425e-02, 1.2453231e-07, 5.3348430e-07,
        9.1728407e-07, 1.0009186e-08, 2.8985988e-07, 1.7532484e-09,
        9.4089705e-01, 1.5346886e-04],
       [1.7055811e-02, 1.1841061e-01, 4.6799007e-05, 2.7727904e-02,
        1.0848254e-03, 1.0896578e-03, 1.3575243e-04, 2.8652203e-04,
        7.8895986e-01, 4.5202184e-02],
       [3.1300801e-01, 1.1591638e-02, 1.1511055e-02, 3.9592334e-03,
        7.7280165e-03, 5.6289224e-05, 2.3531138e-04, 9.4204297e-06,
        6.5178138e-01, 1.1968113e-04],
       [1.3230885e-05, 2.1221960e-05, 9.2594400e-02, 3.3585075e-02,
        4.4722903e-01, 4.1028224e-03, 4.2241842e-01, 2.8064171e-05,
        6.6392668e-06, 1.0745022e-06]], dtype=float32)
```

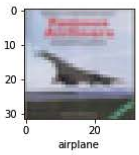
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

```
[3, 8, 8, 8, 4]
```

```
y_test[:5]
```

```
array([3, 8, 8, 0, 6], dtype=uint8)
```

```
plot_sample(X_test, y_test,3)
```




```
classes[y_classes[3]]
```

```
'ship'
```

```
classes[y_classes[3]]
```

```
'ship'
```

Double-click (or enter) to edit

Experiment No 12: Perform basic image processing using opencv

After completion of this experiment students will be able to:

· 1. To learn about images processing

2. To perform various operations on images

Aim: Perform basic image processing using opencv

Theory:

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision.[1] Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel[2]). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports some models from deep learning frameworks like TensorFlow, Torch, PyTorch (after converting to an ONNX model) and Caffe according to a defined list of supported layers.[3]. It promotes OpenVisionCapsules. [4], which is a portable format, compatible with all other formats.

Image processing operations

Digital image processing is the use of a digital computer to process digital images through an algorithm.[1][2] As a subcategory or field of digital signal processing, digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and distortion during processing. Since images are defined over two dimensions (perhaps more) digital image processing may be modeled in the form of multidimensional systems. The generation and development of digital image processing are mainly affected by three factors: first, the development of computers; second, the development of mathematics (especially the creation and improvement of discrete mathematics theory); third, the demand for a wide range of applications in environment, agriculture, military, industry and medical science has increased.

Digital image processing allows the use of much more complex algorithms, and hence, can offer both more sophisticated performance at simple tasks, and the implementation of methods which would be impossible by analogue means. In particular, digital image processing is a concrete application of, and a practical technology based on:

- Classification
- Feature extraction
- Multi-scale signal analysis
- Pattern recognition
- Projection

Some techniques which are used in digital image processing include:

- Anisotropic diffusion
- Hidden Markov models
- Image editing

- Image restoration
- Independent component analysis
- Linear filtering
- Neural networks
- Partial differential equations
- Pixelation
- Point feature matching
- Principal components analysis
- Self-organizing maps
- Wavelets OpenCV methods

Image Acquisition

OpenCV gives the flexibility to capture image directly from a pre-recorded video stream, camera input feed, or a directory path.

```
#Taking input  from a directory path img=cv2.imread('C:\Users\USER\Desktop\image.jpg',0)
# Capturing input from a video stream cap = cv2.VideoCapture(0)
```

Histogram Equalization :

Representation of intensity distribution vs no. of pixels of an image is termed as the histogram.

Equalization stretches out the intensity range in order to suit contrast levels appropriately. More resources here and here.

```
#Taking input  from  a directory path img  =  cv2.imread('wiki.jpg',0)
#Applying  Histogram  Equalization equ  =  cv2.equalizeHist(img)
#Save  the  image cv2.imwrite('res.png',equ)
```

Erosion and Dilation

Erosion and Dilation belong to the group of morphological transformations and widely used together for the treatment of noise or detection of intensity bumps. Here's an extensive resource on the same from the documentation.

```
#Taking input from a directory path img = cv2.imread('wiki.jpg',0)
#Applying Erosion img_erosion = cv2.erode(img, kernel, iterations=1)
#Applying Dilation img_dilation = cv2.dilate(img, kernel, iterations=1)
```

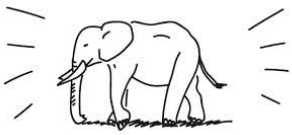
Image Denoising

Noise has a very peculiar property that its mean is zero, and this is what helps in its removal by averaging it out.

OpenCV provides four variations of this technique.

1. cv2.fastNlMeansDenoising() — works with a single grayscale images
2. cv2.fastNlMeansDenoisingColored() — works with a color image.
3. cv2.fastNlMeansDenoisingMulti() — works with image sequence captured in short period of time (grayscale images)
4. cv2.fastNlMeansDenoisingColoredMulti() — same as above, but for color images.

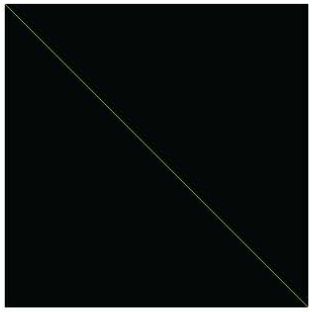
```
import cv2
from google.colab.patches import cv2_imshow
import numpy as np


image=cv2.imread("/images.png")
cv2_imshow(image)
```



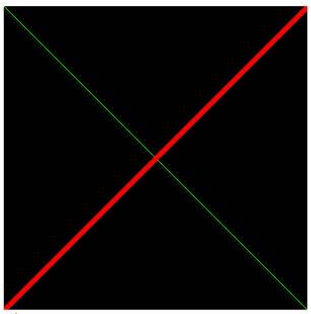
```
cv2.waitKey(0)

corner = image[0:100, 0:100]
cv2_imshow(corner)
cv2.waitKey(0)
```

```
-1

cv2_imshow(corner)
cv2.waitKey(0)
```

```
-1

canvas = np.zeros((300, 300, 3), dtype = "uint8")
#RGB(255,255,255)


green = (0, 255, 0)


cv2.line(canvas, (0, 0), (300, 300), green)
cv2_imshow(canvas)
```

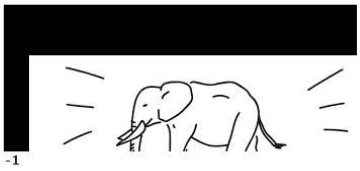


```
cv2.waitKey(0)
red = (0, 0, 255)


cv2.line(canvas, (300, 0), (0, 300), red, 3)
cv2_imshow(canvas)
cv2.waitKey(0)
```



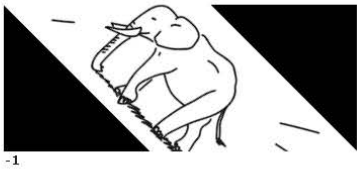
-1

```
M = np.float32([[1, 0, 25], [0, 1, 50]])


shifted = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
cv2_imshow(shifted)
cv2.waitKey(0)
```



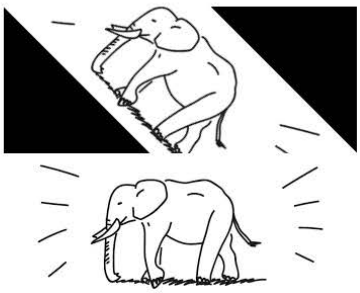
-1

```
(h, w) = image.shape[:2]


center = (w // 2, h // 2)


M = cv2.getRotationMatrix2D(center, -45, 1.0)
rotated = cv2.warpAffine(image, M,(w, h))


cv2_imshow(rotated)
cv2.waitKey(0)
```



-1

```
M = cv2.getRotationMatrix2D(center, -45, 1.0)
rotated = cv2.warpAffine(image, M, (w, h))
cv2_imshow(rotated)
cv2.waitKey(0)
cv2_imshow(image)
print(image.shape)
```



```
(145, 349, 3)


flipped = cv2.flip(image, 0)
cv2_imshow(flipped)
cv2.waitKey(0)
```

-1