**KEYWORDS**

**SVM**

Hyperplane :

In Support Vector Machine (SVM), a hyperplane is a fundamental concept used to separate different classes of data points in a feature space. SVM is a supervised machine learning algorithm that is primarily used for classification tasks.

A hyperplane is a flat affine subspace of one dimension less than the ambient space. In the context of SVM, if you have a dataset with two classes (positive and negative), and your feature space is n-dimensional, then a hyperplane is an (n-1)-dimensional flat surface that divides the feature space into two regions, one for each class.

The main idea of SVM is to find the hyperplane that best separates the data points of different classes while maximizing the margin between the closest data points (support vectors) and the hyperplane. This hyperplane is called the "maximum-margin hyperplane."

Positive Hyperplane :

In the context of SVM, the positive hyperplane is the hyperplane that separates the positive class (the class you want to predict or detect) from the negative class. It is positioned in a way that the positive class data points lie on one side of the hyperplane, and the negative class data points lie on the other side.

Positive Margin :

The goal of SVM is to find the decision boundary that maximizes the margin while minimizing classification errors. The positive margin plays a significant role in achieving this goal. It ensures that the positive class data points are well-separated from the negative class data points, reducing the likelihood of misclassification and improving the robustness of the model.

In mathematical terms, the positive margin is the distance between the decision boundary and the closest positive class data point. SVM aims to find the hyperplane that maximizes this margin, which helps in achieving better generalization to unseen data and better control over overfitting.

Negative Margin(-d) :

the negative margin (-d) in SVM refers to the distance between the decision boundary (hyperplane) and a data point belonging to the negative class. The SVM aims to find a hyperplane that maximizes the margin while correctly classifying data points. Minimizing the absolute value of the negative margin helps ensure that negative data points are not too close to the decision boundary, thus promoting a clear separation between the classes.

Negative Hyperplane :

The negative hyperplane, in conjunction with the positive hyperplane, forms the decision boundaries that allow SVM to make predictions on new, unseen data points.

To summarize, in SVM, the negative hyperplane is the decision boundary that separates the negative class from the positive class in a binary classification problem. It is an essential component of the SVM algorithm, contributing to the creation of a margin between the classes and aiding in the classification of new data points.

Eq of hyperplane = sum of negative and positive margin =D

**DECISION TREE**

Information gain :

The goal of using Information Gain in decision trees is to make the tree as compact as possible while maintaining its ability to accurately classify or predict outcomes. Attributes with high Information Gain are those that effectively separate the classes and provide the most information about the class labels in the dataset.

In summary, Information Gain is a measure used in decision trees to evaluate the effectiveness of attribute splits by quantifying how much uncertainty (entropy) is reduced in the dataset. It helps the algorithm decide which attribute to use for splitting nodes and ultimately guides the construction of the decision tree.

Ginnie index :

The Gini index (also known as the Gini impurity) is a metric used to measure the impurity or disorder of a set of data points. It is often used in decision tree algorithms, such as the CART (Classification and Regression Trees) algorithm, to determine the best splits when building the tree. The Gini index quantifies how often a randomly chosen element from the set would be incorrectly classified.

In summary, the Gini index is a fundamental concept used in decision tree algorithms to assess the impurity of data at each node and guide the process of splitting the data into subsets that result in the most homogeneous child nodes. It's an important factor in determining the best splits for building an accurate decision tree model.

**REGRESSION**

**LINEAR REGRESSION**

Variance :

In the context of linear regression, variance is a statistical concept that is related to the spread or dispersion of the data points around the regression line. Linear regression aims to find the best-fitting line that minimizes the sum of squared differences between the observed data points and the predicted values from the regression line.

In summary, in the context of linear regression, variance is a key concept that reflects the spread or dispersion of the residuals around the regression line. Minimizing the variance of residuals is a primary objective in linear regression modeling, as it indicates a better fit of the model to the data. R-squared is a metric that provides insight into how well the regression model explains the variance in the response variable.

Regressor coefficient :

In linear regression, the term "regressor coefficient" refers to the coefficients associated with the independent variables or features in the linear regression equation. Linear regression is a statistical method used for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data.

The general form of a simple linear regression equation is:

\[ y = b\_0 + b\_1 \cdot x \]

- \( y \) is the dependent variable (the variable you want to predict).

- \( x \) is the independent variable (the variable you are using to make predictions).

- \( b\_0 \) is the intercept or constant term.

- \( b\_1 \) is the coefficient associated with the independent variable \( x \), often referred to as the "regressor coefficient."

In the case of multiple linear regression, where you have more than one independent variable, the equation becomes:

\[ y = b\_0 + b\_1 \cdot x\_1 + b\_2 \cdot x\_2 + \ldots + b\_p \cdot x\_p \]

- \( x\_1, x\_2, \ldots, x\_p \) are the individual independent variables.

- \( b\_1, b\_2, \ldots, b\_p \) are the coefficients associated with the respective independent variables.

These coefficients determine the slope and direction of the relationship between the independent variables and the dependent variable. They represent the change in the dependent variable for a unit change in the corresponding independent variable, while holding other variables constant.

In the context of linear regression, the process of fitting the model involves finding the values of the coefficients (\( b\_0, b\_1, b\_2, \ldots, b\_p \)) that minimize the difference between the predicted values and the actual observed values in the training data. This is usually done using techniques like the least squares method.

Interpreting these coefficients is an important aspect of understanding the relationship between the variables in the regression model. The magnitude and sign of the regressor coefficients help you understand how changes in the independent variables impact the dependent variable.

Regularization :

Regularization in linear regression is a technique used to prevent overfitting and improve the generalization performance of the model. Linear regression aims to find the best-fitting linear relationship between the input features and the target variable. However, when the model becomes too complex, it might fit the training data very well but perform poorly on new, unseen data. Regularization helps mitigate this issue by adding a penalty term to the loss function, discouraging overly complex models.

There are two commonly used types of regularization in linear regression: Lasso (L1 regularization) and Ridge (L2 regularization). Each of these methods introduces a regularization term that is added to the linear regression's original loss function.

**LOGISTIC REGRESSION**

Euler’s Constant : 1.5690

Euler's constant, denoted as "e," is a fundamental mathematical constant that appears in various areas of mathematics and science, including calculus, number theory, and complex analysis. However, as of my last knowledge update in September 2021, Euler's constant itself is not directly related to the logistic regression algorithm in machine learning.

Logistic regression is a classification algorithm used to model the probability that a given input belongs to a particular class. It's commonly used in binary classification problems. The logistic regression model uses the logistic (or sigmoid) function to map the output of a linear combination of input features to a probability value between 0 and 1.

The logistic function used in logistic regression is defined as:

\[ f(z) = \frac{1}{1 + e^{-z}} \]

Here, "e" is Euler's constant, and "z" is the linear combination of input features. The logistic function maps the linear output (which can range from negative infinity to positive infinity) to a probability value between 0 and 1.

While Euler's constant "e" is used in the logistic function's equation, it's not a concept that is specific to logistic regression. It's a mathematical constant that has applications in various mathematical contexts. In the context of logistic regression, it's used as part of the logistic function to model the probability of a binary outcome.

Keep in mind that the field of machine learning is dynamic, and new research and developments can occur beyond my last knowledge update in September 2021. If there have been any new developments or applications involving Euler's constant and logistic regression since then, I recommend checking more recent sources for the latest information.

Confusion Matrix :

A confusion matrix is a concept commonly used to evaluate the performance of classification models, including logistic regression. It provides a clear visualization of the model's predictive accuracy by comparing the predicted class labels to the actual class labels of a dataset. In the context of logistic regression, a confusion matrix helps assess how well the model is able to classify instances into different classes.

A confusion matrix is usually organized into four cells, representing the four possible outcomes of a binary classification problem:

- \*\*True Positive (TP)\*\*: The model correctly predicts the positive class.

- \*\*True Negative (TN)\*\*: The model correctly predicts the negative class.

- \*\*False Positive (FP)\*\*: The model incorrectly predicts the positive class when the true class is negative (Type I error).

- \*\*False Negative (FN)\*\*: The model incorrectly predicts the negative class when the true class is positive (Type II error).

Here's how a confusion matrix is structured:

```

Predicted Positive Predicted Negative

Actual Positive TP FN

Actual Negative FP TN

```

From this matrix, several evaluation metrics can be derived:

1. \*\*Accuracy\*\*: Measures the proportion of correctly classified instances: (TP + TN) / (TP + TN + FP + FN).

2. \*\*Precision (Positive Predictive Value)\*\*: Measures the proportion of true positive predictions among all positive predictions: TP / (TP + FP).

3. \*\*Recall (Sensitivity, True Positive Rate)\*\*: Measures the proportion of true positive predictions among all actual positive instances: TP / (TP + FN).

4. \*\*Specificity (True Negative Rate)\*\*: Measures the proportion of true negative predictions among all actual negative instances: TN / (TN + FP).

5. \*\*F1 Score\*\*: The harmonic mean of precision and recall: 2 \* (Precision \* Recall) / (Precision + Recall).

6. \*\*False Positive Rate (FPR)\*\*: Measures the proportion of false positive predictions among all actual negative instances: FP / (FP + TN).

These metrics provide insights into different aspects of the model's performance. Precision and recall, for example, are especially important in cases where class imbalances exist.

In logistic regression, you can use a confusion matrix to understand how well the model is performing in terms of correctly classifying instances into the target classes. It helps you assess whether the model is making more false positive or false negative errors and guides you in making decisions about the model's threshold for classifying instances.

Actual Value: The true value

Predicted Value: The value from the model

True positive

False positive

False negative

True negative

**CONFUSION MATRIX**

Precision :

How often is the model right when it predicted positive.

Recall :

How often is the model predicted this when it is actually positive.

Specificity :

F1 score : (2\*precision(p)\*recall)/(p+recall)

**UNSUPERVISED LEARNING**

Classification : Grouping happens

Classification is a concept primarily associated with supervised learning, where the goal is to assign labels to data instances based on patterns learned from labeled training data. In supervised classification, the algorithm learns from a dataset where each data point is associated with a known class label, and the goal is to train a model that can accurately predict the class labels of new, unseen data.

Unsupervised learning, on the other hand, is focused on exploring patterns and relationships within data where class labels are not provided. Instead of predicting specific labels, unsupervised learning algorithms aim to discover inherent structures or groupings in the data. As a result, the concept of "classification" as it's understood in supervised learning isn't typically used in the context of unsupervised learning.

However, there is a related concept in unsupervised learning known as "clustering." Clustering is a process of grouping similar data points together in such a way that data points within the same group (cluster) are more similar to each other than to those in other clusters. Clustering doesn't involve assigning predefined labels; it's about finding natural groupings in the data.

Common unsupervised clustering algorithms include:

1. \*\*K-Means Clustering\*\*: This algorithm partitions the data into a specified number of clusters, aiming to minimize the distances between data points and the centroids (centers) of their assigned clusters.

2. \*\*Hierarchical Clustering\*\*: This approach builds a hierarchy of clusters by successively merging or splitting existing clusters based on similarity.

3. \*\*DBSCAN (Density-Based Spatial Clustering of Applications with Noise)\*\*: This algorithm groups data points based on their density within the feature space, and it can identify clusters of arbitrary shapes.

4. \*\*Gaussian Mixture Models (GMM)\*\*: GMM assumes that data points are generated from a mixture of several Gaussian distributions. It estimates these distributions to assign data points to clusters.

5. \*\*Agglomerative Clustering\*\*: This hierarchical clustering method starts with each data point as its own cluster and gradually merges them based on similarity.

In summary, while the term "classification" is not typically used in unsupervised learning, the concept of "clustering" is a fundamental component of unsupervised techniques. Clustering algorithms help in discovering patterns and groupings within data, which can provide insights and aid further analysis.

Assosiation : Pattern Matching

Association in unsupervised learning refers to the process of discovering interesting relationships, patterns, or associations among items in a dataset without any predefined labels or target variables. This type of analysis is often used to identify co-occurring items, frequent itemsets, or rules that can help understand the underlying structure of the data. Association analysis is commonly used in market basket analysis, recommendation systems, and other scenarios where understanding relationships between items is valuable.

1. **MEANS CLUSTERING**

Elbow method : Used to find the optimized value of K.

The elbow method is a heuristic used in K-means clustering to help determine the optimal number of clusters for a given dataset. K-means clustering is an unsupervised machine learning algorithm that groups similar data points into clusters, where each data point belongs to the cluster with the nearest mean (centroid).

The basic idea behind the elbow method is to plot the sum of squared distances (also known as the inertia or within-cluster sum of squares) of data points from their respective cluster centroids as a function of the number of clusters. The plot typically resembles an "elbow," and the point at which the rate of decrease in the sum of squared distances starts to slow down significantly is considered a reasonable estimate for the optimal number of clusters.

Here are the steps involved in using the elbow method for K-means clustering:

1. \*\*Choose a Range of Clusters\*\*: Decide on a range of potential cluster numbers to consider. For example, you might choose to evaluate cluster numbers from 1 to 10.

2. \*\*Compute K-means for Each Cluster Number\*\*: For each cluster number in the chosen range, perform K-means clustering and calculate the sum of squared distances for all data points to their respective cluster centroids.

3. \*\*Plot the Elbow Curve\*\*: Plot the cluster numbers on the x-axis and the corresponding sum of squared distances on the y-axis. The plot should resemble an elbow shape.

4. \*\*Identify the Elbow Point\*\*: Examine the plot and look for the "elbow" point, which is where the rate of decrease in the sum of squared distances starts to slow down. This point is often considered as a good estimate for the optimal number of clusters.

5. \*\*Choose the Optimal Number of Clusters\*\*: The number of clusters corresponding to the elbow point can be considered as the optimal number of clusters for your dataset.

It's important to note that while the elbow method provides a visual heuristic for determining the number of clusters, it's not always definitive. Sometimes, the elbow point might not be very distinct, or the optimal number of clusters might not be well-defined based solely on the elbow criterion. In such cases, other methods like silhouette analysis or domain knowledge might be needed to make a final decision on the number of clusters.

Keep in mind that the elbow method is just one of many techniques for determining the optimal number of clusters in K-means clustering, and its effectiveness can vary depending on the data and the underlying structure of the problem.

Tries to minimize the variance.

**PCA**

Cannot exceed the number of dimensions.

Variance refers to the Standard Deviation.

Orthogonal Shift : Shift the axis (if the variance both the original axis PCA is nearly equal).

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique in the field of machine learning and data analysis. It is used to transform high-dimensional data into a lower-dimensional space while preserving the most important information and variability in the data. PCA achieves this by finding a set of orthogonal axes, known as principal components, along which the data exhibits the most variance.

Here's how PCA works:

1. \*\*Data Centering\*\*: PCA begins by subtracting the mean of each feature from the data. This ensures that the data is centered around the origin.

2. \*\*Covariance Matrix\*\*: The covariance matrix of the centered data is computed. This matrix shows the relationships between different features in the dataset.

3. \*\*Eigenvalue Decomposition\*\*: The covariance matrix is then decomposed into its eigenvectors and eigenvalues. The eigenvectors are the directions in which the data varies the most, and the corresponding eigenvalues represent the variance along those directions.

4. \*\*Selecting Principal Components\*\*: The eigenvectors are sorted based on their corresponding eigenvalues in descending order. The eigenvector with the highest eigenvalue becomes the first principal component, the one with the second-highest eigenvalue becomes the second principal component, and so on.

5. \*\*Reducing Dimensions\*\*: To reduce the dimensionality, you can choose a subset of the top-k principal components that capture a desired amount of the total variance in the data. The new lower-dimensional data can be obtained by projecting the original data onto the selected principal components.

PCA is useful for several purposes:

- \*\*Dimensionality Reduction\*\*: It reduces the number of features while retaining most of the variance, which can speed up training times and reduce overfitting.

- \*\*Noise Reduction\*\*: By focusing on the directions of maximum variance, PCA can help remove noise and redundant information from the data.

- \*\*Data Visualization\*\*: PCA can project high-dimensional data into a lower-dimensional space, making it possible to visualize data in 2D or 3D plots.

- \*\*Feature Engineering\*\*: Principal components can sometimes reveal meaningful patterns or combinations of features that can be used as new features.

However, it's important to note that PCA might not always be appropriate or effective, especially if the data doesn't exhibit a clear linear structure. Additionally, interpreting the meaning of the principal components can be challenging in some cases.

Overall, PCA is a powerful tool for reducing the dimensionality of data while preserving as much relevant information as possible.

**SAMPLE CODE**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**df=pd.read\_csv("knn.csv")**

**df**

**plt.scatter(df["width"],df["height"])**

**plt.show()**

**plt.plot(df["height"],label="Height")**

**plt.plot(df["width"],label="Widht")**

**plt.legend()**

**plt.show()**

**from sklearn.model\_selection import train\_test\_split**

**X=df[['mass','width','height','color\_score']]**

**Y=df[['fruit\_label']]**

**X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,Y, test\_size= .20, random\_state=0)**

**from sklearn.neighbors import KNeighborsClassifier**

**knn=KNeighborsClassifier(n\_neighbors=1)**

**knn.fit(X\_train,y\_train)**

**#let us get the prediction using the classifier we had fit above.**

**y\_pred=knn.predict(X\_test)**

**print("real",y\_test.values)**

**print("pred",y\_pred)**

**from sklearn.metrics import accuracy\_score**

**print("Accuracy Score :",accuracy\_score(y\_test,y\_pred))**

**# Create confusion metrics to display the performance of the data**

**from sklearn.metrics import confusion\_matrix**

**confusionMatrix=confusion\_matrix(y\_test,y\_pred)**

**confusionMatrix**

**appledata=df[df['fruit\_name']=='apple']**

**orangedata=df[df['fruit\_name']=='orange']**

**lemondata=df[df['fruit\_name']=='lemon']**

**mandarindata=df[df['fruit\_name']=='mandarin']**

**plt.scatter(appledata['width'],appledata['height'],color='g',label='apple')**

**plt.scatter(orangedata['width'],orangedata['height'],color='b',label='orange')**

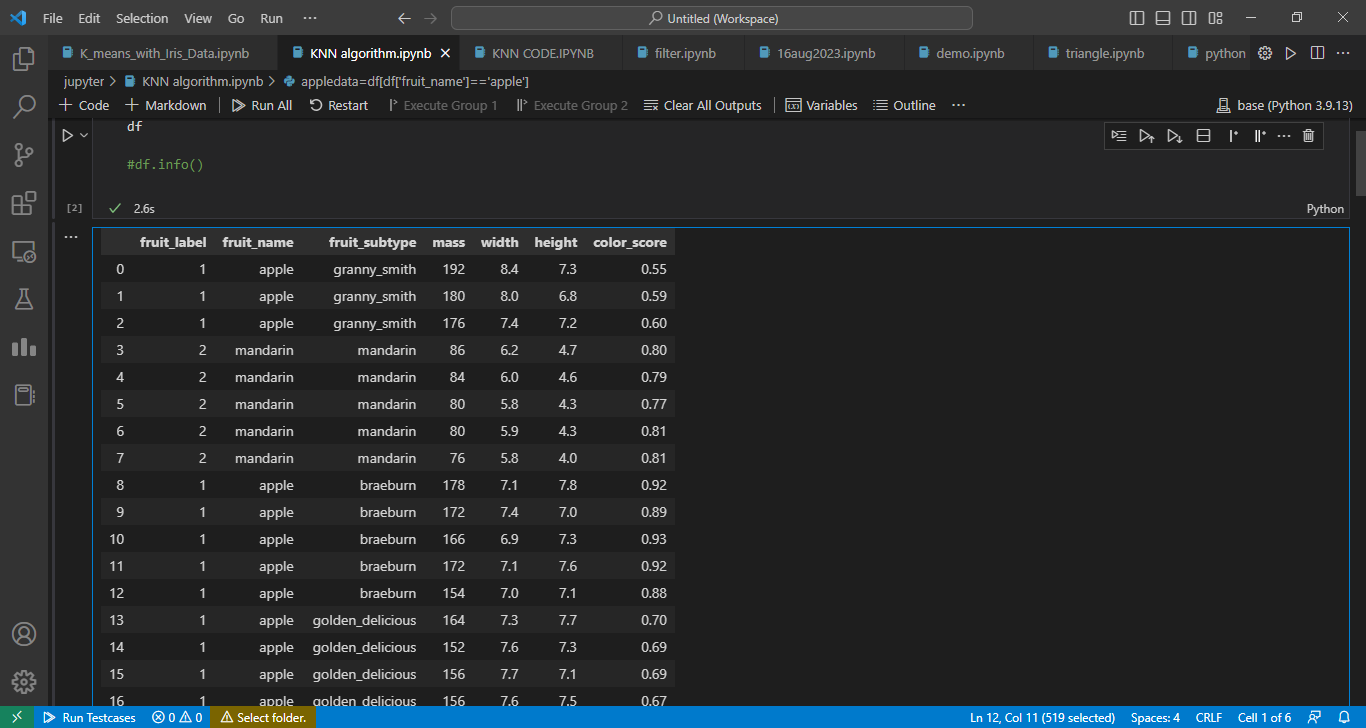
**plt.scatter(lemondata['width'],lemondata['height'],color='r',label='lemon')**

**plt.scatter(mandarindata['width'],mandarindata['height'],color='k',label='mandrin')**

**plt.legend()**

**plt.show()**

**OUTPUT**

****

