Name: Ashish Verma Course:CS773 Exam2

Solution1:-

Given

- (30, 100)
- (50, 90)
- (20, 60)
- (40, 200)
- (35, 50)
- (25, 250)

```
import pandas as pd

# Define the data
data = {
    'Age': [30, 50, 20, 40, 35, 25],
    'Salary': [100, 90, 60, 200, 50, 250]
}

# Create DataFrame
df = pd.DataFrame(data)

# Normalize the data
df['Normalized Age'] = (df['Age'] - 20) / (80 - 20)
df['Normalized Salary'] = (df['Salary'] - 20) / (300 - 20)

df
```

SNO	Age	Salary	Normalized Age	Normalized Salary
0	30	100	0.166667	0.285714
1	50	90	0.5	0.25
2	20	60	0	0.142857
3	40	200	0.333333	0.642857

4	35	50	0.25	0.107143
5	25	250	0.083333	0.821429

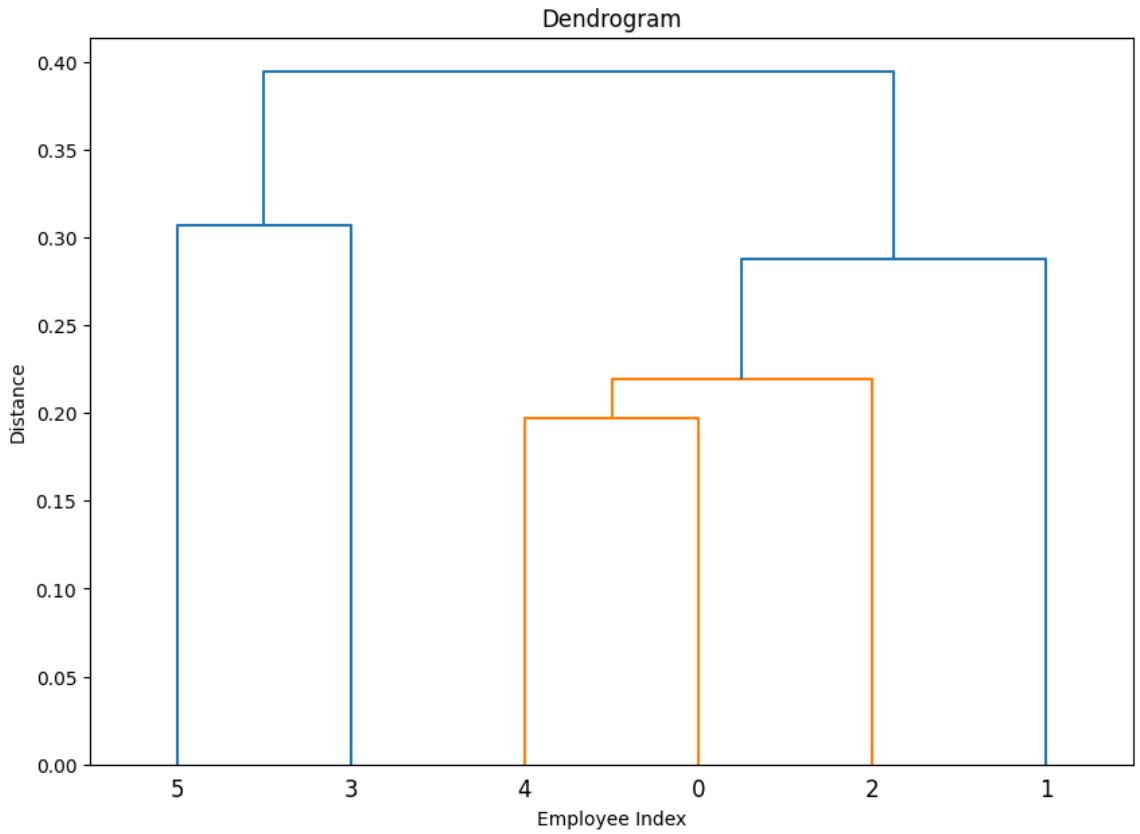
Calculate Distances

```
from scipy.spatial.distance import pdist, squareform

# Extract normalized values
X = df[['Normalized Age', 'Normalized Salary']].values

# Compute pairwise Euclidean distances
distances = pdist(X, metric='euclidean')
distance_matrix = squareform(distances)

pd.DataFrame(distance_matrix, index=df.index, columns=df.index)
```

SNO	0	1	2	3	4	5
0	0	0.335241	0.219513	0.394118	0.197059	0.542157
1	0.335241	0	0.511351	0.426749	0.287938	0.707207
2	0.219513	0.511351	0	0.600925	0.252538	0.683669
3	0.394118	0.426749	0.600925	0	0.542157	0.307226
4	0.197059	0.287938	0.252538	0.542157	0	0.733472
5	0.542157	0.707207	0.683669	0.307226	0.733472	0

Perform Hierarchical Clustering

```python
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt

# Perform hierarchical clustering
linked = sch.linkage(distances, method='single')

# Create a dendrogram
plt.figure(figsize=(10, 7))
dendrogram = sch.dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Dendrogram')
plt.xlabel('Employee Index')
plt.ylabel('Distance')
plt.show()
```



- **Cluster 1**: (30, 100), (50, 90), (35, 50), (25, 250)
- **Cluster 2**: (20, 60), (40, 200)

S

Solution2:-

Given

SE1:

- Number of documents returned: 5,000
- Number of relevant documents returned: 4,000
- Total relevant documents in the repository: 12,000

SE2:

- Number of documents returned: 25,000
- Number of relevant documents returned: 10,000
- Total relevant documents in the repository: 12,000

For SE1

Precision = 4000/5000 = 0.8

Recall = 4000/12000 = 0.333

F1-measure = 2*(0.8*0.333)/(0.8+0.333)= 0.470

For SE2

Precision = 10000/25000 = 0.4

Recall = 10000/12000 = 0.833

F1-measure = 2*(0.4*0.833)/(0.4+0.833)= 0.541

Solution3:-

Create normalized data set

```
import pandas as pd

# Define training data
training_data = {
    'Age': [35, 25, 50, 65, 22],
    'GPA': [2.0, 4.0, 3.0, 2.5, 3.0],
    'Salary': [180, 60, 200, 280, 80],
    'Prediction': ['POOR', 'G', 'G', 'AVG', 'VG']
}

# Create DataFrame
df_train = pd.DataFrame(training_data)

# Normalization
df_train['Normalized Age'] = (df_train['Age'] - 20) / (80 - 20)
df_train['Normalized GPA'] = df_train['GPA'] / 4.0
df_train['Normalized Salary'] = (df_train['Salary'] - 20) / (300 - 20)

df_train
```

SNO	Age	GPA	Salary	Prediction	Normalized Age	Normalized GPA	Normalized Salary
0	35	2	180	POOR	0.25	0.5	0.571429
1	25	4	60	G	0.083333	1	0.142857
2	50	3	200	G	0.5	0.75	0.642857
3	65	2.5	280	AVG	0.75	0.625	0.928571
4	22	3	80	VG	0.033333	0.75	0.214286

Create Unknown date set

```python
# Define unknown data
unknown_data = {
    'Age': [40, 21],
    'GPA': [3.5, 2.5],
    'Salary': [120, 90]
}

# Create DataFrame
df_unknown = pd.DataFrame(unknown_data)

# Normalization
df_unknown['Normalized Age'] = (df_unknown['Age'] - 20) / (80 - 20)
df_unknown['Normalized GPA'] = df_unknown['GPA'] / 4.0
df_unknown['Normalized Salary'] = (df_unknown['Salary'] - 20) / (300 - 20)

df_unknown
```

SNO	Age	GPA	Salary	Normalized Age	Normalized GPA	Normalized Salary
0	40	3.5	120	0.333333	0.875	0.357143
1	21	2.5	90	0.016667	0.625	0.25

Predictions

```python
import numpy as np

# Compute distances and classify
def compute_distances(df_train, df_unknown):
    results = []
    for i, row_unknown in df_unknown.iterrows():
        distances = []
        for j, row_train in df_train.iterrows():
            dist = np.sqrt(
                (row_train['Normalized Age'] - row_unknown['Normalized Age'])**2 +
                (row_train['Normalized GPA'] - row_unknown['Normalized GPA'])**2 +
                (row_train['Normalized Salary'] - row_unknown['Normalized Salary'])**2
            )
            distances.append((dist, row_train['Prediction']))

        # Sort by distance and select the nearest neighbor
        distances.sort(key=lambda x: x[0])
        nearest_prediction = distances[0][1]
        results.append(nearest_prediction)

    return results

# Apply to unknown data
predictions = compute_distances(df_train, df_unknown)
predictions
```

['G', 'VG']

Solution4:-

Given

```
data = {
    'Instance#': ['I1', 'I2', 'I3', 'I4', 'I5', 'I6', 'I7', 'I8', 'I9'],
    'Age': [25, 55, 65, 35, 25, 22, 50, 35, 24],
    'GPA': [3.0, 2.5, 4.0, 3.2, 4.0, 3.5, 2.0, 3.0, 2.7],
    'Salary': ['High', 'High', 'High', 'Med', 'Med', 'Med', 'Low', 'Low', 'Low']
}
```

```python
import pandas as pd
from sklearn.model_selection import StratifiedKFold

# Define the dataset
data = {
    'Instance#': ['I1', 'I2', 'I3', 'I4', 'I5', 'I6', 'I7', 'I8', 'I9'],
    'Age': [25, 55, 65, 35, 25, 22, 50, 35, 24],
    'GPA': [3.0, 2.5, 4.0, 3.2, 4.0, 3.5, 2.0, 3.0, 2.7],
    'Salary': ['High', 'High', 'High', 'Med', 'Med', 'Med', 'Low', 'Low', 'Low']
}

df = pd.DataFrame(data)

# Features and target
X = df[['Age', 'GPA']]
y = df['Salary']

# Define the StratifiedKFold with 3 splits
skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# Split the data
folds = list(skf.split(X, y))

# Display the splits
for fold_number, (train_index, test_index) in enumerate(folds, start=1):
    print(f"Fold {fold_number}:")

    # Training data
    X_train, y_train = X.iloc[train_index], y.iloc[train_index]
    print("Training data:")
    print(pd.concat([X_train, y_train], axis=1))

    # Testing data
    X_test, y_test = X.iloc[test_index], y.iloc[test_index]
    print("Testing data:")
    print(pd.concat([X_test, y_test], axis=1))

    print("\n" + "-"*40 + "\n")
```

Fold 1:

Training data:

	Age	GPA	Salary
1	55	2.5	High
2	65	4.0	High
3	35	3.2	Med
4	25	4.0	Med
7	35	3.0	Low
8	24	2.7	Low

Testing data:

	Age	GPA	Salary
0	25	3.0	High
5	22	3.5	Med
6	50	2.0	Low

Fold 2:

Training data:

	Age	GPA	Salary
0	25	3.0	High
2	65	4.0	High
4	25	4.0	Med
5	22	3.5	Med
6	50	2.0	Low
8	24	2.7	Low

Testing data:

	Age	GPA	Salary
1	55	2.5	High
3	35	3.2	Med
7	35	3.0	Low

Fold 3:

Training data:

	Age	GPA	Salary
0	25	3.0	High
1	55	2.5	High
3	35	3.2	Med
5	22	3.5	Med
6	50	2.0	Low

7 35 3.0 Low

Testing data:

	Age	GPA	Salary
2	65	4.0	High
4	25	4.0	Med
8	24	2.7	Low

Solution5:-

Given Data

Instance #	GPA	Age	Salary
1	3.5	25	60K
2	2.5	40	50K
3	4	35	70K
4	3	30	80K

```python
import numpy as np

# Normalization function
def normalize(value, min_value, max_value):
    return (value - min_value) / (max_value - min_value)

# Normalize the query tuple
gpa_given = normalize(3.3, 0, 4.0)
age_given = normalize(35, 20, 60)

# Instances data: (GPA, Age, Salary)
instances = [
    (normalize(3.5, 0, 4.0), normalize(25, 20, 60), 60),
    (normalize(2.5, 0, 4.0), normalize(40, 20, 60), 50),
    (normalize(4.0, 0, 4.0), normalize(35, 20, 60), 70),
    (normalize(3.0, 0, 4.0), normalize(30, 20, 60), 80)
]

# Compute distances
distances = []
for gpa, age, salary in instances:
    distance = np.sqrt((gpa - gpa_given) ** 2 + (age - age_given) ** 2)
    distances.append((distance, salary))

# Find the nearest neighbor
nearest_neighbor = min(distances, key=lambda x: x[0])
predicted_salary = nearest_neighbor[1]

print(f"Predicted Salary: ${predicted_salary}K")
```

Predicted Salary: $80K

Therefore, the nearest neighbor for the given tuple is **Instance 4** with a salary of **80K**.

Solution6:-

Steps to Employ Bagging Technique with J48 Classifier are as follow:-

1.Generate Bootstrap Samples

Bootstrapping: Create multiple bootstrap samples from the original training data. Each bootstrap sample is a random sample with replacement from the original dataset. Since you have 10,000 training instances and need to create 10 models, you will generate 10 bootstrap samples.

Sample Size: Each bootstrap sample should ideally be the same size as the original dataset (10,000 instances), though it's a random sample with replacement.

2.Train J48 Classifiers

For each bootstrap sample, train a separate J48 decision tree classifier. This means you will train 10 J48 classifiers, each on a different bootstrap sample.

3.Aggregate Predictions

Prediction: For each test instance, get predictions from all 10 trained J48 classifiers. This will give you 10 predictions for each test instance.

Voting Mechanism: Aggregate the predictions from all classifiers. The most common prediction (i.e., the mode of the predictions) is typically

used as the final output. In case of a tie, some methods might use additional heuristics.

4.Output of Bagging Given 10 Models' Results

```
from collections import Counter

# Given predictions from 10 models
predictions = ['excellent', 'poor', 'average', 'excellent', 'excellent',
               'average', 'poor', 'average', 'average', 'excellent']

# Count the occurrences of each prediction
prediction_counts = Counter(predictions)

# Find the most common prediction
most_common_prediction = prediction_counts.most_common(1)[0][0]

print(f"The output of the bagging process is: {most_common_prediction}")
```

The output of the bagging process is: **excellent**

Solution7:-

```python
import math

# Function to calculate the new weight of a training instance
def calculate_new_weight(old_weight, error):
    if error <= 0 or error >= 1:
        raise ValueError("Error must be between 0 and 1 (exclusive).")
    new_weight = old_weight * (error / (1 - error))
    return new_weight

# Function to calculate the weight of a classifier in AdaBoost
def calculate_classifier_weight(error):
    if error <= 0 or error >= 1:
        raise ValueError("Error must be between 0 and 1 (exclusive).")
    weight = 0.5 * math.log((1 - error) / error)
    return weight

# Given values
old_weight = 0.8
error_classifier = 0.2
errors = [0.05, 0.1, 0.8]

# Calculate new weight of the training instance
new_weight = calculate_new_weight(old_weight, error_classifier)
print(f"Modified weight of the training instance for the next iteration: {new_weight:.2f}")

# Calculate weights for each classifier
weights = [calculate_classifier_weight(error) for error in errors]

# Print classifier weights
for i, (error, weight) in enumerate(zip(errors, weights), 1):
    print(f"Classifier {i} with error {error:.2f}: Weight = {weight:.2f}")
```

Modified weight of the training instance for the next iteration: **0.20**

Classifier 1 with error 0.05: Weight = **1.47**

Classifier 2 with error 0.10: Weight = **1.10**

Classifier 3 with error 0.80: Weight = **-0.69**

Solution8:-

Instance 1: Education=BS, GPA=B, Experience=5 years

Salary from EQ1: 50+2×5=50+10=60

Salary from EQ2: 70+4×5=70+20=90

Salary from EQ4: 80

Average = (60+90+80)/3 =76.66

Instance 2: Education=HS, GPA=C, Experience=15 years

Salary from EQ6: 50+5×15=50+75=125

Salary from EQ7: 40

Average = (125+70)/2 = 97.5

Instance 3: Education=MS, GPA=A, Experience=0

Salary from EQ1: 50+2×0=50

Salary from EQ4: 80

Average = (50+80)/2 = 65

- Instance 1 Salary: 76.67K
- Instance 2 Salary: 82.5K
- Instance 3 Salary: 65K

Solution 9:-

Instance 1: Education=MS, GPA=B, Experience=5 years

EQ2: 30+5×5=30+25=55K

Total Salary = 30+55 = 85K

Instance 2: Education=HS, GPA=B, Experience=15 years

EQ4: 15+5×15=15+75=90K

Total Salary = 10K+20K+90K=120K

Instance 3: Education=MS, GPA=A, Experience=0

EQ1: Salary=20+2×0=20K

Total Salary = 10+20+20=50K

- Instance 1 Salary: 85K
- Instance 2 Salary: 120K
- Instance 3 Salary: 50K

Solution10:-

Given

EQ1: Salary = Age * 1.5 + Experience * 2.0 + 15

EQ2: Res_salary = Age * 0.2 + Experience * 0.15 – 5

```
# Define the functions for EQ1 and EQ2
def predict_salary(age, experience):
    # EQ1: Salary = Age * 1.5 + Experience * 2.0 + 15
    salary = age * 1.5 + experience * 2.0 + 15
    return salary

def predict_residual_salary(age, experience):
    # EQ2: Res_salary = Age * 0.2 + Experience * 0.15 - 5
    res_salary = age * 0.2 + experience * 0.15 - 5
    return res_salary

# Define instances
instances = [
    {"age": 35, "experience": 10},
    {"age": 25, "experience": 2}
]

# Calculate and print predictions
for instance in instances:
    age = instance["age"]
    experience = instance["experience"]

    salary = predict_salary(age, experience)
    res_salary = predict_residual_salary(age, experience)
    final_salary = salary + res_salary

    print(f"Instance with Age={age} and Experience={experience}:")
    print(f"  Predicted Salary = ${final_salary:.2f}K")
    print()
```

Instance with Age=35 and Experience=10:

Predicted Salary = **$91.00K**

Instance with Age=25 and Experience=2:

Predicted Salary = **$56.80K**

Solution11:-

Three-Level Authentication System are as follows:-

First Level: **Something We Know** (Knowledge-Based Authentication)

- **Password:** A traditional and widely used method where users must enter a secret password known only to them.
- **PIN (Personal Identification Number):** A short numeric code that users enter in addition to or instead of a password.
- **Security Questions:** Answers to personal questions (e.g., mother's maiden name, first pet's name) used to verify identity.

Second Level: **Something We Have** (Possession-Based Authentication)

- **Smart Cards:** Physical cards with embedded chips that users must insert into a reader or tap on a contactless reader.
- **Token Generators:** Devices or apps that generate a one-time passcode (OTP) that users enter after providing their password. These can be hardware tokens or software-based (e.g., Google Authenticator).
- **Mobile Authentication Apps:** Apps on a smartphone that provide OTPs or use push notifications for authentication (e.g., Duo Mobile, Authy).

Third Level: **Something We Are** (Biometric Authentication)

- **Fingerprint Scanners:** Devices that read and verify the user's fingerprint.
- **Facial Recognition:** Systems that use cameras to recognize and authenticate users based on facial features.
- **Iris Scanners:** Devices that scan and match the user's iris pattern.

- **Voice Recognition:** Systems that analyze and authenticate the user's voice pattern.

Benefits of a Three-Level Authentication System

1. **Enhanced Security:** By requiring multiple forms of authentication, the system significantly reduces the risk of unauthorized access. Even if one factor (e.g., a password) is compromised, the additional layers (e.g., OTP, biometric) provide further security.
2. **Mitigates Risk of Different Attack Vectors:** Each level addresses different potential vulnerabilities. For instance, passwords might be stolen or guessed, but possessing a physical token and a biometric sample adds layers of protection.
3. **Compliance and Best Practices:** Multi-factor authentication (MFA) is often a requirement for compliance with various regulations and standards, such as GDPR, HIPAA, and PCI-DSS.
4. **User Assurance:** Users gain increased confidence in the security of the systems they are accessing, knowing that multiple authentication factors are used to verify their identity.

Solution12:-

RBAC is a method of access control where permissions are assigned to roles rather than directly to users. Users are then assigned to these roles, and through their role, they inherit the permissions associated with it. This model is well-suited for environments where users have clearly defined roles and responsibilities, and it simplifies the management of access permissions.

Justification for RBAC in a University Setting

Defined Roles and Responsibilities:

- **Students:** Typically require access to course materials, grades, and registration systems but not to faculty or staff administrative functions.
- **Faculty:** Need access to course management systems, grading tools, and possibly research resources, but not to student registration or payroll systems.
- **Staff:** Handle administrative functions such as registration, payroll, and maintenance of institutional records. They need access to sensitive administrative data but not to course materials or grading systems.

Simplified Access Management:

- With RBAC, managing permissions becomes more straightforward. Instead of assigning permissions to each individual user, permissions are assigned to roles, and users are assigned to these roles. For example, all faculty members are assigned the "Faculty" role with specific permissions, and the same goes for students and staff.
- This model reduces administrative overhead and minimizes the risk of errors in assigning permissions.

Enhanced Security and Compliance:

- **Principle of Least Privilege:** RBAC ensures that users have only the access necessary to perform their job functions, reducing the risk of unauthorized access to sensitive data.
- **Regulatory Compliance:** Universities often need to comply with various regulations and standards (e.g., FERPA in the U.S. for student privacy). RBAC helps ensure that access controls align with these compliance requirements by restricting access to sensitive information to only those who need it.

Scalability:

- As the university grows, the RBAC model scales well. New roles can be created as needed (e.g., new administrative roles or specialized faculty roles), and permissions can be adjusted without changing individual user settings.

Ease of Role Management:

- When a user's role changes (e.g., a student becomes a staff member), it's easier to update their access rights by simply changing their role rather than individually updating each permission.

Role Definitions:

- **Student Role:** Access to course materials, grades, and registration systems.
- **Faculty Role:** Access to course management systems, grading tools, and research resources.
- **Staff Role:** Access to registration, payroll, and administrative systems.

Access Control Policies:

- **Course Management System:** Faculty can create and modify courses, while students can view course content.
- **Registration System:** Staff manage registrations, while students can only register or modify their own information.
- **Payroll System:** Accessible only to specific administrative staff with the appropriate role.

Solution13:-

The availability of tiny microphones, such as those found in smartphones, smart home devices (like smart speakers), or even wearable technology, raises significant privacy concerns for several reasons:

Data Collection: Companies and apps may use these microphones to collect data on users' behaviors, preferences, and conversations. This data can be used for targeted advertising or other purposes without the users' explicit consent.

Misuse of Data: If audio recordings are not properly protected, they can be accessed by malicious actors. This can result in sensitive information being leaked or used for identity theft, blackmail, or other harmful activities.

Surveillance and Eavesdropping: Tiny microphones can capture conversations and sounds from their surroundings. If these devices are hacked or misused, they can record private conversations without the consent of the individuals involved. This could lead to unauthorized surveillance or eavesdropping.

When an instructor makes a list of students enrolled in a class available to the public, there can be a privacy compromise for the individual students.

Exposure of Personal Information: The list typically includes personal details such as names, and sometimes additional information like email addresses or phone numbers. When this information is publicly accessible, it can be exploited for various purposes, including spam, unsolicited contacts, or identity theft.

Safety Risks: Publicly available information about students can lead to safety concerns. For example, individuals with malicious intent could

use this information to harass students or gain unauthorized access to other personal details.

Academic and Professional Consequences: Publicizing student information can affect students' academic and professional lives. For instance, potential employers or colleagues might access this information and make judgments based on it, potentially leading to discrimination or bias.

Solution14:-

True Positive (TP): 1650

False Positive (FP): 1350−550 = 800

False Negative (FN): 550

True Negative (TN): 550

TPR = TP/(TP+FN) = 1650/(1650+550) = 0.75 = 75%

FPR = FP/(FP+TN) = 800/(800+550) = 0.59 = 59%

Pattern Discovery: Data mining helps identify new attack patterns by analyzing historical data. It can reveal previously unknown attack vectors and behaviors, which can then be converted into new signatures.

Pattern Matching: Advanced data mining techniques can refine pattern matching algorithms to improve the accuracy of existing signatures, reducing false positives and false negatives.

Automatic Signature Generation: Data mining can automate the process of generating new signatures by discovering new attack patterns and trends from ongoing data. This helps in updating signatures more frequently and efficiently.

Adaptive Signatures: Data mining can adapt signatures to evolving attack techniques, ensuring that the IDS remains effective against new and modified threats.

Complementary Approach: Although signature-based IDS are generally good at detecting known threats, they can struggle with new or modified attacks. Data mining can complement signature-based detection with anomaly detection, identifying deviations from normal behavior that may indicate new attacks.

Behavioral Analysis: Data mining can identify abnormal patterns or deviations from standard behavior, which can help in creating signatures for previously unknown attack methods.

Clustering: Data mining techniques like clustering can group similar types of attacks or false positives, allowing for more accurate classification and better management of signatures.

Classification Algorithms: Data mining can use classification techniques to improve the decision-making process regarding whether an alert is a true positive or a false positive, thus fine-tuning the effectiveness of signatures.

Trend Identification: Data mining can analyze trends in attack data to predict future attack patterns and prepare the IDS with appropriate signatures.

Predictive Modeling: By analyzing historical attack data, data mining can predict potential vulnerabilities and suggest proactive measures and signatures to address emerging threats.

Correlation of Data: Data mining can correlate data from different sources to provide a more comprehensive view of potential threats. This helps in refining signatures by incorporating contextual information, improving the accuracy of the detection.

Enhanced Context: Understanding the context in which certain behaviors occur can help in creating more precise signatures that take into account various environmental and situational factors.

Data Visualization: Data mining can provide visualization tools that help in understanding complex attack patterns and the effectiveness of signatures. Visualizing attack data can reveal trends and relationships that are not immediately obvious.

Interpretation of Results: Data mining techniques can help interpret the results of signature-based detection by providing insights into the nature of the attacks and the performance of the signatures.

Solution 15:-

IDS1

	Predicted Normal	Predicted Intrusion
Actual Normal	440	10
Actual Intrusion	40	110

IDS2

	Predicted Normal	Predicted Intrusion

Actual Normal	415	35
Actual Intrusion	15	135

Total loss for IDS-1:

Loss_IDS1 = FN*80000+FP*30000

= 40*80000+10*30000

= **3500000**

Total loss for IDS-2:-

Loss_IDS2 = FN*80000+FP*30000

= 15*80000+35*30000

= **2250000**

IDS-2 should be purchased, as it results in a significantly lower total financial loss compared to IDS-1.

Solution 16:-

- **User Activity Logs:** Collect logs of all user activities on the database, including login times, queries executed, data accessed, and any anomalies.
- **System Logs:** Gather logs from the database management system, including error logs, transaction logs, and access control logs.
- **Network Logs:** Monitor network traffic to and from the database server to identify any unusual patterns.

- **Data Cleaning:** Remove any noise or irrelevant data from the logs. Ensure data consistency and accuracy.
- **Feature Engineering:** Extract meaningful features from the logs. These features could include the number of login attempts, duration of sessions, types of queries executed, volume of data accessed, and timestamps.
- Use visualization techniques like histograms, scatter plots, and heatmaps to understand the data distribution and identify any initial patterns or outliers.
- Conduct statistical analysis to determine the normal behavior patterns of users.
- **Clustering:** Techniques like K-Means, DBSCAN, or hierarchical clustering can group users based on their behavior. Any user whose behavior significantly deviates from the cluster norm could be flagged as suspicious.
- Principal Component Analysis (PCA): Reduce the dimensionality of the data and identify outliers based on principal components.
- If there are labeled instances of past unauthorized access attempts, supervised learning algorithms like decision trees, random forests, or support vector machines (SVM) can be trained to classify normal and suspicious activities.
- **Sequence Analysis:** Analyze the sequence of user actions to detect patterns that deviate from normal behavior. Techniques like Markov models can be useful here.
- **Time-Series Analysis:** Monitor user behavior over time and use time-series analysis to detect anomalies.
- **Rule-Based Systems:** Implement real-time monitoring systems that use predefined rules to flag suspicious activities. For example, multiple failed login attempts or access to sensitive data outside of normal working hours.

- **Machine Learning Models:** Deploy trained machine learning models to continuously analyze user behavior and detect anomalies in real-time
- **Alert Systems:** Set up alert systems to notify administrators of potential insider threats.
- **Access Control:** Implement additional access controls or temporarily suspend access for users flagged as suspicious until further investigation is completed.
- **Investigation:** Conduct thorough investigations on flagged users to determine if the anomalies were indeed unauthorized access attempts.

Solution17:-

Methods of Feature Selection

- **Correlation Coefficient:** Measure the correlation between each feature and the target variable. Features with high correlation to the target and low correlation to each other are preferred.
- **Chi-Square Test:** Used for categorical data to measure the independence between features and the target variable.
- **Recursive Feature Elimination (RFE):** Recursively removes the least important features based on the model's performance.
- **Forward/Backward Selection:** Iteratively adds or removes features based on their contribution to the model's performance.
- **Regularization:** Techniques like Lasso (L1) and Ridge (L2) regression add penalties for larger coefficients, effectively shrinking some coefficients to zero and thus performing feature selection.
- **Tree-Based Methods:** Algorithms like Random Forests and Gradient Boosting automatically provide feature importance scores.

Methods of Feature Extraction

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Autoencoders

Solution 18:-

Distance to Cluster 1

D1 = sqrt((2-3)^2+(15 -5)^2+(10 -15)^2) = 11.23

Distance to Cluster 2

D2 = sqrt((2-10)^2+(15 -15)^2+(10 – 5)^2) =9.43

The Euclidean distance between the unknown connection and Cluster 1 is approximately 11.23, and the distance to Cluster 2 is approximately 9.43. Since 9.43 is less than 11.23, the unknown connection is closer to Cluster 2.

Assumptions

1. **Equal Weighting**: Each feature (failed login attempts, file creations, sent emails) is given equal importance in the calculation of Euclidean distance.
2. **Euclidean Distance Validity**: The Euclidean distance metric is appropriate for measuring the similarity between the feature vectors in this context.
3. **Cluster Representativeness**: The centroids provided accurately represent the respective clusters.

Solution19:-

Begin by gathering all relevant access data. This includes information on which employees have access to which files or folders. Each record should indicate the user, the file or folder, and the type of access (e.g., read, write, execute).

Clean the Data: Ensure the data is free from errors, duplicates, and inconsistencies. Correct any anomalies that might affect the analysis.
Format the Data: Construct a matrix where each row represents a user and each column represents a file or folder. The entries in this matrix indicate access rights (e.g., binary values where 1 means access is granted and 0 means it is not).

Conduct an exploratory analysis to understand patterns and trends in the access data. Visual tools like heatmaps can be helpful to identify obvious clusters or outliers in access patterns.

Use clustering techniques to group users based on similar access patterns:

1. **K-Means Clustering**: This algorithm divides users into k clusters, where each cluster represents a potential role. Use methods like the Elbow Method to determine the optimal number of clusters.
2. **Hierarchical Clustering**: Build a hierarchy of clusters, which can be particularly useful if roles have a nested structure.
3. **DBSCAN**: A density-based clustering algorithm that can identify clusters of arbitrary shape and handle noise in the data.

Analyze the clusters to define roles based on the access patterns within each cluster. Assign meaningful names to these roles that reflect their access privileges, such as "HR Manager" or "Production Supervisor."

Internal Validation: Check if the roles make sense internally and align with organizational policies and structure.
External Validation: Consult with domain experts or department managers to ensure the derived roles are practical and accurate. Adjust roles based on their feedback

Refinement: Ensure roles are neither too broad nor too specific. Merge similar roles and split overly broad roles as necessary.
Access Control Policies: Clearly define the access control policies for each role to maintain the principle of least privilege.

Deploy the derived roles in the organization's access control system. Ensure that these roles are used consistently across all access control points.

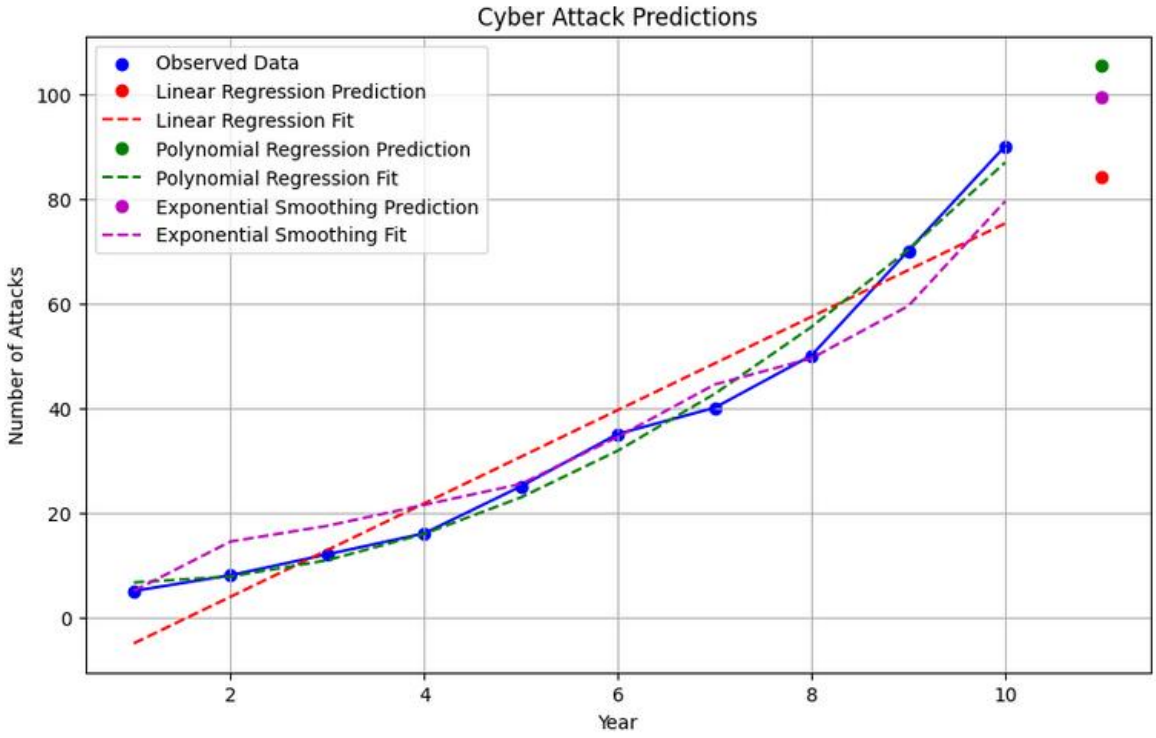
Regularly review and update the roles to reflect changes in the organization or evolving access needs. Continuous monitoring helps maintain the relevance and security of the role-based access control system.

Document the process thoroughly, including the rationale behind each role and the methodologies used. Provide a report summarizing the derived roles and the expected benefits of the new role-based access control system.

Solution20:-

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Given data
years = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
attacks = np.array([5, 8, 12, 16, 25, 35, 40, 50, 70, 90])

# Linear Regression
linear_model = LinearRegression()
linear_model.fit(years.reshape(-1, 1), attacks)
linear_prediction = linear_model.predict([[11]])

# Polynomial Regression (degree 2)
poly_features = PolynomialFeatures(degree=2)
years_poly = poly_features.fit_transform(years.reshape(-1, 1))
poly_model = LinearRegression()
poly_model.fit(years_poly, attacks)
poly_prediction = poly_model.predict(poly_features.transform([[11]]))

# Exponential Smoothing
exp_smoothing_model = ExponentialSmoothing(attacks, trend="add", seasonal=None)
exp_smoothing_fit = exp_smoothing_model.fit()
exp_smoothing_prediction = exp_smoothing_fit.predict(start=10, end=10)

# Plotting the data and predictions
plt.figure(figsize=(10, 6))
plt.scatter(years, attacks, color='blue', label='Observed Data')
plt.plot(years, attacks, color='blue')

# Linear Regression Prediction
plt.plot([11], linear_prediction, 'ro', label='Linear Regression Prediction')
plt.plot(years, linear_model.predict(years.reshape(-1, 1)), 'r--', label='Linear Regression Fit')

# Polynomial Regression Prediction
plt.plot([11], poly_prediction, 'go', label='Polynomial Regression Prediction')
plt.plot(years, poly_model.predict(years_poly), 'g--', label='Polynomial Regression Fit')

# Exponential Smoothing Prediction
plt.plot([11], exp_smoothing_prediction, 'mo', label='Exponential Smoothing Prediction')
exp_smoothing_fit_fittedvalues = exp_smoothing_fit.fittedvalues
plt.plot(years, exp_smoothing_fit_fittedvalues, 'm--', label='Exponential Smoothing Fit')

plt.xlabel('Year')
plt.ylabel('Number of Attacks')
plt.legend()
plt.title('Cyber Attack Predictions')
plt.grid(True)
plt.show()

print(f"Linear Regression Prediction for YR11: {linear_prediction[0]:.2f}")
print(f"Polynomial Regression Prediction for YR11: {poly_prediction[0]:.2f}")
print(f"Exponential Smoothing Prediction for YR11: {exp_smoothing_prediction[0]:.2f}")
```

- Linear Regression Prediction for YR11: **84.13**
- Polynomial Regression Prediction for YR11: **105.47**
- Exponential Smoothing Prediction for YR11: **99.44**



Solution21:-

Data Masking

- **Age Masking**: Replace actual ages with a range or a broader category.
- **Salary Masking**: Replace exact salaries with ranges or approximate values.

Name	Age	Salary ($K)
Kim	20-30	60-80
John	30-40	70-90
Jane	40-50	50-60
Joe	40-50	90-110

Jill	20-30	40-60

Aggregation

- **Age Aggregation**: Group ages into broader ranges.
- **Salary Aggregation**: Group salaries into ranges.

Age Range	**Average Salary ($K)**
20-29	60
30-39	75
40-49	57.5
50-59	100

K-Anonymity

Age and **Salary** values are generalized to ensure each combination appears at least k times. K =2

Name	**Age Range**	**Salary Range ($K)**
Kim	20-30	60-80
John	30-40	70-90
Jane	40-50	50-60
Joe	40-50	90-110
Jill	20-30	40-60

Data Perturbation

- **Age Perturbation**: Add or subtract a random value within a small range.
- **Salary Perturbation**: Add or subtract a random value within a range

Assuming random perturbations within ±5 for age and ±10K for salary

Name	Age (Perturbed)	Salary ($K) (Perturbed)
Kim	27	65
John	33	78
Jane	47	58
Joe	48	95
Jill	32	52

Noise Addition

- **Age Noise**: Add random noise from a uniform distribution.
- **Salary Noise**: Add random noise from a uniform distribution.

Assume noise range for age: ±5 years, for salary: ±10K

Name	Age (Noisy)	Salary ($K) (Noisy)
Kim	27	65
John	36	85
Jane	44	56
Joe	52	102
Jill	31	49

Solution22:-

```python
import numpy as np

# Given data
true_salaries = {
    'Kim': 30,
    'John': 75,
    'Jane': 50,
    'Joe': 40,
    'Jill': 30
}

# Given average and perturbation range
average_perturbed_salary = 45
number_of_employees = 6
total_perturbed_sum = average_perturbed_salary * number_of_employees
true_salaries_sum = sum(true_salaries.values())
total_perturbation = total_perturbed_sum - true_salaries_sum

# Jane makes $15K more than Kim
jane_more_than_kim = 15

# Perturbation range
perturbation_range = (-20, 20)

# Function to generate modified salaries
def generate_modified_salaries(true_salaries, perturbation_range):
    modified_salaries = {}
    for name, true_salary in true_salaries.items():
        perturbation = np.random.randint(perturbation_range[0], perturbation_range[1] + 1)
        modified_salaries[name] = true_salary + perturbation
    return modified_salaries

# Function to estimate Kim's salary
def estimate_kim_salary(true_salaries, jane_more_than_kim, perturbation_range):
    kim_salaries = []
    jane_salaries = []

    for _ in range(10000):  # Generate a large number of samples
        modified_salaries = generate_modified_salaries(true_salaries, perturbation_range)

        kim_salary = modified_salaries['Kim']
        jane_salary = modified_salaries['Jane']

        if jane_salary - kim_salary == jane_more_than_kim:
            kim_salaries.append(kim_salary)
            jane_salaries.append(jane_salary)

    # Compute the range for Kim's salary
    kim_salary_min = min(kim_salaries) if kim_salaries else None
    kim_salary_max = max(kim_salaries) if kim_salaries else None

    return kim_salary_min, kim_salary_max

# Estimate Kim's salary range
kim_salary_min, kim_salary_max = estimate_kim_salary(true_salaries, jane_more_than_kim, perturbation_range)

print(f"Estimated Kim's Salary Range: ${kim_salary_min}K to ${kim_salary_max}K")
```

Estimated Kim's Salary Range: $15K to $50K

Solution23:-

Sort the data by Salary

Name	Salary ($K)
Corey	28
Kim	33
Jing	40
King	55
Asha	70
Curtis	75
Joe	95
Bob	100
Jane	120
Jackey	120
Bill	140

Group the Data for 3-Anonymization

Group 1

- Corey, Kim, Jing
- Salaries: 28, 33, 40
- Generalized Salary: 28-40

Group 2

- King, Asha, Curtis
- Salaries: 55, 70, 75
- Generalized Salary: 55-75

Group 3

- Joe, Bob, Jane, Jackey, Bill

- Salaries: 95, 100, 120, 120, 140
- Generalized Salary: 95-140

Generalizations to the Original Table

Name	Generalized Salary ($K)
Corey	28-40
Kim	28-40
Jing	28-40
King	55-75
Asha	55-75
Curtis	55-75
Joe	95-140
Bob	95-140
Jane	95-140
Jackey	95-140
Bill	95-140

Solution24:-

Perceived and Actual Privacy:

- Users often perceive social media as a private space to express their opinions and share personal content. However, the reality is that these platforms are public by default unless users take specific actions to enhance their privacy settings.
- Even with strict privacy settings, posts can be shared or screenshotted, making it difficult to maintain complete control over one's content.

Terms of Service:

- Social media platforms have terms of service that users agree to, which often include clauses about data usage and privacy. Users should be aware that their information and content might be accessible to more people than they intend.

Public Representation:

- Employees are often seen as representatives of their employers, even outside of work hours. Public posts that reflect poorly on the employer can lead to reputational damage, prompting employers to take action.
- High-profile roles or industries that emphasize public trust (e.g., healthcare, education, law) might have stricter expectations for employees' conduct on social media.

Workplace Policies:

- Many organizations have social media policies outlining acceptable behavior and potential repercussions for violating these policies. Employees should be familiar with these guidelines.
- Disciplinary actions are usually more justifiable if the employee's posts are discriminatory, harassing, or otherwise violate company values or laws.

Free Speech vs. Consequences:

- While employees have the right to free speech, they are not free from the consequences of that speech. Employers also have the right to protect their brand and workplace culture.
- Posts that are not work-related but still offensive or harmful can impact workplace dynamics and public perception.

Legal Protections:

- Different countries have varying legal protections for employees regarding off-duty conduct. For example, in the United States, some states have laws protecting employees from being fired for legal off-duty activities, but these protections are not absolute.
- Employees should understand their local laws and how they apply to social media use.

The privacy offered by social media platforms is often limited and can be misunderstood by users. Employers may have valid reasons for disciplining employees based on their social media posts, especially if those posts conflict with the company's values or public image. However, there is a fine line between protecting the company's interests and infringing on employees' rights to express themselves outside of work. Striking a balance requires clear policies, employee awareness, and a nuanced understanding of privacy and free speech implications in the digital age.

Solution25:-

Global Support = Support1 +Support2 = 600+800 = 1400

Global Accuracy = (Support1*Accuracy1+Support2* Accuracy2)/Global Support

Global Accuracy = (600*0.75+800*0.5)/1400 = 0.607

Hence Global Support = 1400 and Global Accuracy = 60.71% approximately.