|  |  |  |  |
| --- | --- | --- | --- |
| **1 SIMILARITY MEASURES**  import pd np  from sklearn.preprocessing import StandardScaler  laptop\_data.head() laptop\_data.info()  laptop\_data.describe() len(laptop\_data)  laptop\_data.isnull().sum()  laptop\_data['Rating'] = np.random.randint(1,6 , size=len(laptop\_data)) laptop\_data  laptop\_data.to\_csv("New Laptop data.csv", index=False)  laptop\_data = pd.read\_csv("/content/New Laptop data.csv")  laptop\_data.head() index1 = 0 index2 = 1  row1 = laptop\_data.iloc[index1][['Ram', 'Weight', 'Price', 'Rating']].values  row2 = laptop\_data.iloc[index2][['Ram', 'Weight', 'Price', 'Rating']].values  euclidean\_dist = np.sqrt(np.sum((row1 - row2) \*\* 2))  manhattan\_dist = np.sum(np.abs(row1 - row2))  dot\_product = np.dot(row1, row2)  magnitude1 = np.sqrt(np.sum(row1 \*\* 2))  magnitude2 = np.sqrt(np.sum(row2 \*\* 2))  cosine\_sim = dot\_product / (magnitude1 \* magnitude2)  print(f"\nEuclidean Distance between these products: {euclidean\_dist:.4f}")  print(f"Manhattan Distance between these products: {manhattan\_dist:.4f}")  print(f"Cosine Similarity between these products: {cosine\_sim:.4f}")  if euclidean\_dist < 5:      print("\nSuggestion: These products are quite similar in features and price!")  else:      print("\nSuggestion: These products are Dissimilar.") | **2 DIMENSION REDUCTION TECHNIQUES**  import pd np plt sns  from sklearn.preprocessing import StandardScaler  laptop\_data.head()  laptop\_data.to\_csv("New Laptop data.csv", index=False)  print("Data saved to New Laptop data.csv")  laptop\_data = pd.read\_csv("/content/New Laptop data.csv")  def preprocess\_data(laptop\_data):      df['Cpu\_brand'] = df['Cpu\_brand'].astype('category').cat.codes      df['Gpu\_brand'] = df['Gpu\_brand'].astype('category').cat.codes      df['Os'] = df['Os'].astype('category').cat.codes      return df  processed\_df = preprocess\_data(laptop\_data)  features = ['Ram', 'Weight', 'Price', 'TouchScreen', 'Ips', 'Ppi', 'Cpu\_brand', 'HDD', 'SSD', 'Rating']  x = processed\_df[features].values  x\_mean = np.mean(x, axis=0)  x\_std = np.std(x, axis=0)  x\_standardized = (x - x\_mean) / x\_std  cov\_matrix = np.cov(x\_standardized.T)  eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)  sorted\_index = np.argsort(eigenvalues)[::-1]  sorted\_eigenvalues = eigenvalues[sorted\_index]  sorted\_eigenvectors = eigenvectors[:, sorted\_index]  n\_components = 2  eigenvectors\_subset = sorted\_eigenvectors[:, 0:n\_components]  principal\_components = np.dot(x\_standardized, eigenvectors\_subset)  pca\_df = pd.DataFrame(data=principal\_components, columns=['Principal Component 1', 'Principal Component 2'])  final\_df = pd.concat([pca\_df, df[['Company', 'TypeName']]], axis=1)  print(final\_df)  eigenvectors\_df = pd.DataFrame(      eigenvectors\_subset,      columns=[f'PC{i+1}' for i in range(n\_components)],      index=features )  plt.figure(figsize=(12, 8))  sns.heatmap(eigenvectors\_df, annot=True, cmap='coolwarm', center=0)  plt.title('Eigenvectors (Loadings) for Selected Principal Components')  plt.xlabel('Principal Components')  plt.ylabel('Features') plt.show() | **7 ACCURACY METRICS**  import pd plt  from sklearn.model\_selection import train\_test\_split  from sklearn.ensemble import RandomForestClassifier  from sklearn.preprocessing import LabelEncoder  from sklearn.metrics import roc\_curve, auc, confusion\_matrix, classification\_report, roc\_auc\_score  from sklearn.model\_selection import RandomizedSearchCV  label\_encoder = LabelEncoder()  data['Company'] = label\_encoder.fit\_transform(data['Company'])  data['TypeName'] = label\_encoder.fit\_transform(data['TypeName'])  data['Cpu\_brand'] = label\_encoder.fit\_transform(data['Cpu\_brand'])  data['Gpu\_brand'] = label\_encoder.fit\_transform(data['Gpu\_brand'])  data['Os'] = label\_encoder.fit\_transform(data['Os'])  data['Username'] = label\_encoder.fit\_transform(data['Username'])  X = data.drop(columns=['Rating']) y = data['Rating']  y\_binary = y.apply(lambda x: 1 if x >= 4 else 0)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_binary, test\_size=0.3, random\_state=42)  rf\_model = RandomForestClassifier(random\_state=42)  rf\_model.fit(X\_train, y\_train)  y\_pred\_proba = rf\_model.predict\_proba(X\_test)[:, 1]  y\_pred = rf\_model.predict(X\_test)  fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)  roc\_auc = auc(fpr, tpr)  plt.figure(figsize=(8, 6))  plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')  plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  plt.xlim([0.0, 1.0]) plt.ylim([0.0, 1.05])  plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate')  plt.title('Receiver Operating Characteristic (ROC)')  plt.legend(loc="lower right") plt.show()  print("Classification Report:\n", classification\_report(y\_test, y\_pred))  print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))  param\_dist = {      'n\_estimators': [50, 100, 200],      'max\_depth': [10, 20, 30, None],      'min\_samples\_split': [2, 5, 10],      'min\_samples\_leaf': [1, 2, 4],      'bootstrap': [True, False]  } | **6 Attacks In Recommendation Systems**  import pd gr  from sklearn.feature\_extraction.text import TfidfVectorizer  from sklearn.metrics.pairwise import linear\_kernel  df.fillna('', inplace=True)  def content\_based\_filtering(target\_username, top\_n=5):      df['combined\_features'] = df['Company'] + ' ' + df['TypeName'] + ' ' + df['Cpu\_brand'] + ' ' + df['Gpu\_brand'] + ' ' + df['Os']      tfidf = TfidfVectorizer(stop\_words='english')      tfidf\_matrix = tfidf.fit\_transform(df['combined\_features'])      cosine\_sim = linear\_kernel(tfidf\_matrix, tfidf\_matrix)      if target\_username not in df['Username'].values:          return "Username not found in the dataset."      target\_index = df[df['Username'] == target\_username].index[0]      sim\_scores = list(enumerate(cosine\_sim[target\_index])      sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)      top\_indices = [i[0] for i in sim\_scores[1:top\_n + 1]]      return df.iloc[top\_indices][['Username', 'Company', 'TypeName', 'Cpu\_brand', 'Gpu\_brand', 'Os', 'Price']  def collaborative\_filtering(target\_username, top\_n=5):      user\_laptop\_matrix = df.pivot\_table(index='Username', columns='Company', values='Price', fill\_value=0)      if target\_username not in user\_laptop\_matrix.index:          return "Username not found in the dataset.”      cosine\_sim = linear\_kernel(user\_laptop\_matrix, user\_laptop\_matrix)      target\_index = user\_laptop\_matrix.index.get\_loc(target\_username)      sim\_scores = list(enumerate(cosine\_sim[target\_index]))      sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)     top\_indices = [i[0] for i in sim\_scores[1:top\_n + 1]]      top\_users = user\_laptop\_matrix.index[top\_indices]      target\_user\_laptops = set(df[df['Username'] == target\_username]['Company'])      top\_laptops = df[(df['Username'].isin(top\_users)) & (~df['Company'].isin(target\_user\_laptops))]      return top\_laptops[['Username', 'Company', 'TypeName', 'Cpu\_brand', 'Gpu\_brand', 'Os', 'Price']].head(top\_n)  def get\_recommendations(username):      content\_recs = content\_based\_filtering(username, top\_n=5)      collab\_recs = collaborative\_filtering(username, top\_n=5)       return content\_recs, collab\_recs |
|  | **8 HYBRID RECOMMENDATION**  **#CONTENT BASED**  from sklearn.feature\_extraction.text import TfidfVectorizer  from sklearn.metrics.pairwise import linear\_kernel np  'Gpu\_brand', and 'Os'  data['combined\_features'] = data['Company'] + ' ' + data['TypeName'] + ' ' + data['Cpu\_brand'] + ' ' + data['Gpu\_brand'] + ' ' + data['Os']  vectorizer = TfidfVectorizer(stop\_words='english')  feature\_matrix = vectorizer.fit\_transform(data['combined\_features'])  cosine\_sim = linear\_kernel(feature\_matrix, feature\_matrix)  def get\_content\_recommendations(title, cosine\_sim=cosine\_sim):      idx = data.index[data['Company'] == title].tolist()[0]  sim\_scores = list(enumerate(cosine\_sim[idx]))  sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)      sim\_scores = sim\_scores[1:6]      laptop\_indices = [i[0] for i in sim\_scores]  return data.iloc[laptop\_indices]  get\_content\_recommendations('Acer')  **#COLLABORATIVE BASED**  from sklearn.metrics.pairwise import cosine\_similarity  from sklearn.preprocessing import LabelEncoder  label\_encoder = LabelEncoder()  data['user\_id'] = label\_encoder.fit\_transform(data['Username'])  user\_item\_matrix = data.pivot\_table(index='user\_id', columns='Company', values='Price', aggfunc='mean').fillna(0)  user\_similarity = cosine\_similarity(user\_item\_matrix)  def get\_collaborative\_recommendations(user\_name, user\_similarity=user\_similarity):      user\_idx = label\_encoder.transform([user\_name])[0]      sim\_scores = list(enumerate(user\_similarity[user\_idx]))      sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)      sim\_scores = sim\_scores[1:6]      user\_indices = [i[0] for i in sim\_scores]      similar\_users\_laptops = data[data['user\_id'].isin(user\_indices)]['Company'].unique()      return similar\_users\_laptops  get\_collaborative\_recommendations('Harish')  **#HYBRID RECOMMENDATION SYSTEM**  def get\_hybrid\_recommendations(user\_name, title, alpha=0.7, top\_n=5):      content\_recommendations = get\_content\_recommendations(title, top\_n=top\_n)      content\_recommendations\_set = set(content\_recommendations['Company'])      collaborative\_recommendations = get\_collaborative\_recommendations(user\_name, top\_n=top\_n)      collaborative\_recommendations\_set = set(collaborative\_recommendations)      hybrid\_recommendations = list(content\_recommendations\_set.intersection(collaborative\_recommendations\_set))      if not hybrid\_recommendations:          # Combine both sets with weighted averaging (alpha) | **8 HYBRID RECOMMENDATION**  import pandas as pd  from sklearn.feature\_extraction.text import TfidfVectorizer  from sklearn.metrics.pairwise import linear\_kernel, cosine\_similarity  from sklearn.preprocessing import LabelEncoder  data['combined\_features'] = (data['Company'] + ' ' + data['TypeName'] + ' ' + data['Cpu\_brand'] + ' ' + data['Gpu\_brand'] + ' ' + data['Os'] + ' ' + data['Ram'].astype(str) + 'GB ' +                               data['TouchScreen'].astype(str) + ' ' +                               data['Ips'].astype(str) + ' ' +                               data['Ppi'].astype(str) + 'PPI ' +                               data['Price'].astype(str) + 'USD ' +                               data['Weight'].astype(str) + 'kg')  vectorizer = TfidfVectorizer(stop\_words='english')  feature\_matrix = vectorizer.fit\_transform(data['combined\_features'])  cosine\_sim = linear\_kernel(feature\_matrix, feature\_matrix)  label\_encoder = LabelEncoder()  data['user\_id'] = label\_encoder.fit\_transform(data['Username'])  user\_item\_matrix = data.pivot\_table(index='user\_id', columns='Company', values='Price', aggfunc='mean').fillna(0)  user\_similarity = cosine\_similarity(user\_item\_matrix)  #**CONTENT BASED**  def get\_content\_recommendations(title, top\_n=5):      idx = data.index[data['Company'] == title].tolist()[0]      sim\_scores = list(enumerate(cosine\_sim[idx]))      sim\_scores = sorted(sim\_scores[1:], key=lambda x: x[1], reverse=True)[:top\_n]      laptop\_indices = [i[0] for i in sim\_scores]      recommendations = data.iloc[laptop\_indices].drop\_duplicates(subset=['Company', 'TypeName'])      return recommendations  **#COLLABORATIVE BASED**  def get\_collaborative\_recommendations(user\_name, top\_n=5):      user\_idx = label\_encoder.transform([user\_name])[0]      sim\_scores = list(enumerate(user\_similarity[user\_idx]))      sim\_scores = sorted(sim\_scores[1:], key=lambda x: x[1], reverse=True)[:top\_n]      user\_indices = [i[0] for i in sim\_scores]      similar\_users\_laptops = data[data['user\_id'].isin(user\_indices)]['Company'].unique()      return similar\_users\_laptops  **#POPULARITY BASED**  def get\_popularity\_recommendations(top\_n=5):      popular\_laptops = data.groupby('Company').size().sort\_values(ascending=False).index[:top\_n]      return list(popular\_laptops)  **#ITEM BASED FILTERING**  item\_sim\_matrix = cosine\_similarity(feature\_matrix)  def get\_item\_based\_recommendations(title, top\_n=5):      idx = data.index[data['Company'] == title].tolist()[0]      sim\_scores = list(enumerate(item\_sim\_matrix[idx]))      sim\_scores = sorted(sim\_scores[1:], key=lambda x: x[1], reverse=True)[:top\_n]      item\_indices = [i[0] for i in sim\_scores] |  |
| 6 interface = gr.Interface(      fn=get\_recommendations,      inputs=gr.Textbox(lines=1, placeholder="Enter Username", label="Username"),      outputs=[gr.Dataframe(label="Content-Based Recommendations"),               gr.Dataframe(label="Collaborative Filtering Recommendations")],      title="Laptop Recommendation System",      description="Enter a username to get personalized laptop recommendations based on content-based and collaborative filtering." interface.launch()  #ATTACK ON CONTENT BASED FILTERING  def profile\_injection\_attack(df, num\_fake\_profiles=10, target\_company='Apple', target\_type='Ultrabook'):      fake\_profiles = []      for i in range(num\_fake\_profiles):          fake\_profile = {              'Ram': np.random.choice([8, 16]),              'Weight': np.random.uniform(1.0, 2.0),        }          fake\_profiles.append(fake\_profile)      return pd.DataFrame(fake\_profiles)  def attribute\_manipulation\_attack(df, target\_company='Apple', target\_type='Ultrabook', new\_price=500.0):      df.loc[(df['Company'] == target\_company) & (df['TypeName'] == target\_type), 'Price'] = new\_price      return df  fake\_profiles\_df = profile\_injection\_attack(df)  manipulated\_df = attribute\_manipulation\_attack(df.copy())  augmented\_df\_profile\_injection = pd.concat([df, fake\_profiles\_df], ignore\_index=True)  print("Profile Injection Attack Dataset:")  print(augmented\_df\_profile\_injection.tail(10))  print("\nAttribute Manipulation Attack Dataset:")  print(manipulated\_df.tail(10))  fake\_profiles\_df manipulated\_df  #ATTACK ON COLLABORATIVE BASED FILTERING  def sybil\_attack(df, num\_fake\_profiles=10, target\_company='Apple', target\_type='Ultrabook'):      fake\_profiles = []      for i in range(num\_fake\_profiles):          fake\_profile = {}          fake\_profiles.append(fake\_profile)      return pd.DataFrame(fake\_profiles)  def rating\_injection\_attack(df, num\_fake\_ratings=10, target\_company='Apple', target\_type='Ultrabook'):      fake\_ratings = []      for i in range(num\_fake\_ratings):          fake\_rating = {}          fake\_ratings.append(fake\_rating)      return pd.DataFrame(fake\_ratings)  sybil\_profiles\_df = sybil\_attack(df)  fake\_ratings\_df = rating\_injection\_attack(df)  augmented\_df\_sybil = pd.concat([df, sybil\_profiles\_df], ignore\_index=True)  augmented\_df\_ratings = pd.concat([df, fake\_ratings\_df], ignore\_index=True)  print("Sybil Attack Dataset:")  print(augmented\_df\_sybil.tail(10))  print("\nRating Injection Attack Dataset:")  print(augmented\_df\_ratings.tail(10)) | 7 rf\_random = RandomizedSearchCV(estimator=rf\_model, param\_distributions=param\_dist,                                 n\_iter=50, cv=3, verbose=1, random\_state=42, n\_jobs=-1)  rf\_random.fit(X\_train, y\_train)  best\_rf\_model = rf\_random.best\_estimator\_  y\_pred\_proba\_best = best\_rf\_model.predict\_proba(X\_test)[:, 1]  y\_pred\_best = best\_rf\_model.predict(X\_test)  fpr\_best, tpr\_best, \_ = roc\_curve(y\_test, y\_pred\_proba\_best)  roc\_auc\_best = auc(fpr\_best, tpr\_best)  plt.figure(figsize=(8, 6))  plt.plot(fpr\_best, tpr\_best, color='green', lw=2, label=f'Fine-Tuned ROC (AUC = {roc\_auc\_best:.2f})')  plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  plt.xlim([0.0, 1.0]) plt.ylim([0.0, 1.05])  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('Fine-Tuned ROC Curve')  plt.legend(loc="lower right") plt.show()  print("Fine-Tuned Classification Report:\n", classification\_report(y\_test, y\_pred\_best))  print("Fine-Tuned Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_best)) | 2 def encode\_user\_input(cpu\_brand, gpu\_brand, os):      cpu\_mapping = {name: idx for idx, name in enumerate(df['Cpu\_brand'].unique())}      gpu\_mapping = {name: idx for idx, name in enumerate(df['Gpu\_brand'].unique())}      os\_mapping = {name: idx for idx, name in enumerate(df['Os'].unique())}      encoded\_cpu = cpu\_mapping.get(cpu\_brand, -1)      encoded\_gpu = gpu\_mapping.get(gpu\_brand, -1)      encoded\_os = os\_mapping.get(os, -1)      if -1 in [encoded\_cpu, encoded\_gpu, encoded\_os]:          raise ValueError("Invalid input provided for CPU brand, GPU brand, or OS.")      return encoded\_cpu, encoded\_gpu, encoded\_os  def standardize\_input(user\_input, x\_mean, x\_std):      return (user\_input - x\_mean) / x\_std  def recommend\_laptop(final\_df, user\_input\_standardized):      user\_principal\_components = np.dot(user\_input\_standardized, eigenvectors\_subset)      distances = np.linalg.norm(final\_df[['Principal Component 1', 'Principal Component 2']].values - user\_principal\_components, axis=1)      recommended\_index = np.argmin(distances)      return final\_df.iloc[recommended\_index][['Company', 'TypeName']] | **3 USER PROFILE LEARNING**  import pd np  from sklearn.metrics.pairwise import cosine\_similarity  from sklearn.preprocessing import StandardScaler  categorical\_columns = ['Company', 'TypeName', 'Cpu\_brand', 'Gpu\_brand', 'Os']  df = pd.get\_dummies(df, columns=categorical\_columns)  numerical\_columns = ['Ram', 'Weight', 'Price', 'TouchScreen', 'Ips', 'Ppi', 'HDD', 'SSD']  scaler = StandardScaler()  df[numerical\_columns] = scaler.fit\_transform(df[numerical\_columns])  user\_profiles = df.groupby('Username').mean()  user\_profiles.head()  similarity\_matrix = cosine\_similarity(user\_profiles)  similarity\_df = pd.DataFrame(similarity\_matrix, index=user\_profiles.index, columns=user\_profiles.index)  similarity\_df  **def recommend\_preferences(user, similarity\_df, user\_profiles, top\_n=1):**       if user not in similarity\_df.index:          print(f"User '{user}' not found.")          return None      similar\_users = similarity\_df[user].sort\_values(ascending=False).index[1:top\_n+1]      recommendations = user\_profiles.loc[similar\_users].mean()      return recommendations  user\_input = input("Enter your username: ")  recommendations = recommend\_preferences(user\_input, similarity\_df, user\_profiles, top\_n=1)  if recommendations is not None:      print(f"\nRecommended Preferences for {user\_input}:\n")      print(recommendations)` |
|  | item\_recommendations = data.iloc[item\_indices]['Company'].unique()      return list(item\_recommendations)  **#HYBRID RECOMMENDATION SYSTEM**  def get\_hybrid\_recommendations(user\_name, title, alpha=0.5, beta=0.3, gamma=0.1, delta=0.1, top\_n=5):      content\_recommendations = get\_content\_recommendations(title, top\_n=top\_n)      content\_recommendations\_set = set(content\_recommendations['Company'])      collaborative\_recommendations = get\_collaborative\_recommendations(user\_name, top\_n=top\_n)      collaborative\_recommendations\_set = set(collaborative\_recommendations)      popularity\_recommendations = get\_popularity\_recommendations(top\_n=top\_n)      popularity\_recommendations\_set = set(popularity\_recommendations)      item\_based\_recommendations = get\_item\_based\_recommendations(title, top\_n=top\_n)      item\_based\_recommendations\_set = set(item\_based\_recommendations)      hybrid\_recommendations = list(content\_recommendations\_set)[:int(alpha \* top\_n)] + \                               list(collaborative\_recommendations\_set)[:int(beta \* top\_n)] + \                               list(popularity\_recommendations\_set)[:int(gamma \* top\_n)] + \                               list(item\_based\_recommendations\_set)[:int(delta \* top\_n)]      hybrid\_recommendations = list(dict.fromkeys(hybrid\_recommendations))[:top\_n]      print(f"Content-based: {content\_recommendations\_set}")      print(f"Collaborative: {collaborative\_recommendations\_set}")      print(f"Popularity-based: {popularity\_recommendations\_set}")      print(f"Item-based: {item\_based\_recommendations\_set}")      print(f"Hybrid Recommendations: {hybrid\_recommendations}")      return hybrid\_recommendations  recommendations = get\_hybrid\_recommendations('Harish', 'Apple', alpha=0.4, beta=0.3, gamma=0.2, delta=0.1, top\_n=5)  print("Final Hybrid Recommendations:", recommendations) | combined\_recommendations = list(content\_recommendations\_set.union(collaborative\_recommendations\_set))          content\_weighted = int(top\_n \* alpha)          collaborative\_weighted = top\_n - content\_weighted         hybrid\_recommendations = list(content\_recommendations\_set)[:content\_weighted] + \                                   list(collaborative\_recommendations\_set)[:collaborative\_weighted]      print(f"Hybrid Recommendations (User: {user\_name}, Title: {title}): {hybrid\_recommendations}")      return hybrid\_recommendations  recommendations = get\_hybrid\_recommendations('Harish', 'Acer', alpha=0.7, top\_n=5)  print("Hybrid Recommendations:", recommendations) |  |