jupyter Phase 4 unsupervised model.ipynb Last Checkpoint: 3 days ago

File Edit View Insert Run Kernel Settings Help Trusted

Code Jupyterlab Python 3 (ipykernel)

[1]: # Read and Redirect Stdout
```
[2025-06-26 10:40:02] INFO    Using
                              /home/jaqen/Desktop/Desktop/CS_Manapal/Capstone/Develop/venv/lib/python3.10/site-packages/dm-pretrainedsentiment-project-0.0.1_jaqen-py3.10.egg.egg configuration.
[2025-06-26 10:40:02] INFO    Registerd item key: 'Bitcoin_Address'                                    __init__.py:40
[2025-06-26 10:40:02] INFO    Registerd item key: 'Email_Address'                                      __init__.py:40
[2025-06-26 10:40:02] INFO    Registerd item key: 'IP_Address'                                         __init__.py:40
[2025-06-26 10:40:02] INFO    Using configuration file
                              C:/Users/jaqen/Desktop/Desktop/CS_Manapal/Capstone/Develop/venv/lib/python3.10/site-packages/dm-pretrainedsentiment-project-0.0.1_jaqen-py3.10.egg.egg/dm_pretrainedsentiment/data/config.yaml
[2025-06-26 10:40:02] WARNING C:/Users/jaqen/Desktop/Desktop/CS_Manapal/Capstone/Develop/venv/lib/python3.10/site-packages/nltk/corpus/__init__.py:165:
                              UserWarning: Lazy loading of corpora through configurations in nltk.data.path is
                              incompatible with the future version you are currently using.
                              warnings.warn(
[2025-06-26 10:40:02] INFO    Failed to loading enumerous usage data with the sole purpose of improving plugin.py:155
                              the product. No personal data or IP addresses are stored on our side. If
                              you want to opt-out, set the value of the environment variable or
                              environment variable, or create a .telemetry file in the current
                              working directory or in your home directory. Read more at
                              https://docs.dask.org/en/latest/configuration-reference.html
[2025-06-26 10:40:02] INFO    Dask project Manapal                                                     __init__.py:104
[2025-06-26 10:40:02] INFO    Defined global variable 'context', 'session', 'catalog' and              __init__.py:105
                              'pipeline'
[2025-06-26 10:40:02] INFO    Registerd line magic 'run_kl'                                            __init__.py:109
```

[2]: context

[3]: %reload_ext kedro_viz
```
[2025-06-26 10:40:02] INFO    Reloading kedro_viz (TarpaulinSaint)                                     data_catalog.py:180
```

[4]: catalog

[5]: %kedro_viz launch-packaged-pipelines --port=4141 --host=127.0.0.1 --autoreload --no-browser --project-path C:/Users/jaqen/Desktop/Desktop/CS_Manapal/Capstone/Develop/dm_pretrainedsentiment_project --env=local --pipeline=data_science --load-versions=preprocessed_data:2025-06-26T09.57.00.000Z,kmeans_model:2025-06-26T09.57.00.000Z,pca_model:2025-06-26T09.57.00.000Z,scaled_data:2025-06-26T09.57.00.000Z --parameters='{"kmeans":{"n_clusters":5,"random_state":42},"pca":{"n_components":3}}'

```
[2025-06-26 10:40:02] INFO    Reloading kedro_viz (TarpaulinSaint)                                     data_catalog.py:180
```

[6]: import pandas as pd
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, IncrementalPCA
from sklearn.metrics import silhouette_score, silhouette_samples
from sklearn.cluster import KMeans, MiniBatchKMeans
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.cluster.hierarchy import dendrogram, linkage, fcluster
from mpl_toolkits.mplot3d import mpl_toolkits
```

[7]: %reload_ext kedro_viz
```
[2025-06-26 10:40:02] INFO    Reloading kedro_viz (TarpaulinSaint)                                     data_catalog.py:180
```

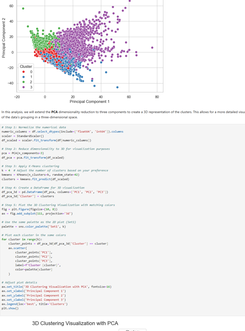
In this analysis we will apply **Unsupervised Clustering** to **scaled subset** of the dataset to create **clusters**. We will then visualize the distribution of these clusters to understand how the data points are grouped.

[8]: # Step 1: Take a sample of the dataset (adjust size as needed)
```
# Using 10% of the data for faster processing, or adjust based on your dataset
sample_indices = pd.sample(df_scaled_pca_features.index, frac=0.1, random_state=42)
df_sample_scaled = df_scaled[sample_indices]

# Step 2: Create dendrogram
Z = linkage(df_sample_scaled, method='ward')

# Plot dendrogram
plt.figure(figsize=(12, 8))
dendrogram(Z, truncate_mode='lastp', p=30)
plt.title('Cluster Distribution (Sample)')
plt.xlabel('Cluster')
plt.ylabel('Count')
plt.grid(True)
plt.show()
```

[9]: [2025-06-26 10:40:02] WARNING C:\Users\jaqen\AppData\Local\Temp\ipykernel_30960\2748733443.py:20: warnings.warn(f"Ignoring unknown kwarg 'truncate_mode'")

In this analysis we will apply **Principal Component Analysis (PCA)** to reduce the dimensionality of the dataset to two principal components. This will allow us to visualize the clusters identified by the **K-Means** algorithm in a two-dimensional space.

[10]: # Set up plotting style
```
sns.set(style="darkgrid")

# Ensure the original DataFrame is loaded as 'df'
# Filter only numerical columns
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
df_numeric = df[numeric_cols]

# Scale the data (standardization)
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_numeric)

# Apply PCA with 2 components for visualization (3 components)
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)

# Create a DataFrame with the principal components
df_pca = pd.DataFrame(df_pca, columns=['PC1', 'PC2'])

# Apply K-Means
k = 4 # Define the number of clusters
kmeans = KMeans(n_clusters=k, random_state=42)
clusters = kmeans.fit_predict(df_scaled)

# Add cluster labels to the PCA DataFrame
df_pca['Cluster'] = clusters

# Visualize the clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(
    data=df_pca,
    x='PC1',
    y='PC2',
    hue='Cluster',
    palette='tab10',
    s=50,
    legend='full'
)
plt.title('Clustering with K-Means and PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster', loc='best')
plt.grid(True)
plt.show()
```



In this analysis, we will extend the **PCA** dimensionality reduction to three components to create a 3D representation of the clusters. This allows for a more detailed visualization of the data's grouping in a three-dimensional space.

[11]: # Step 1: Normalize the numerical data
```
numeric_columns = df_scaled_pca_features.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
df_numeric = df_scaled_pca_features[numeric_columns]

# Step 2: Apply PCA with 3 components for 3D visualization purposes
pca = PCA(n_components=3)
df_pca = pca.fit_transform(df_scaled)

# Step 3: Apply K-Means clustering
# Using 4 clusters as a reasonable starting point based on your preference
kmeans = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(df_scaled)

# Step 4: Create a DataFrame for 3D visualization
df_pca_3d = pd.DataFrame(df_pca, columns=['PC1', 'PC2', 'PC3'])
df_pca_3d['Cluster'] = clusters

# Step 5: Plot the 3D clustering visualization with matching colors
fig = go.Figure(go.Scatter3d(x=df_pca_3d['PC1'],
    y=df_pca_3d['PC2'],
    z=df_pca_3d['PC3'],
    mode='markers',
    marker=dict(size=5,
                color=df_pca_3d['Cluster'] * 10,
                colorscale='Viridis',
                opacity=0.8),
    text=df_pca_3d['Cluster'],
    hoverinfo='text'))

# Use the same palette as the 2D plot (tab10)
palette = sns.color_palette('tab10', 4)

# Plot each cluster in the same colors
for cluster in range(4):
    cluster_points = df_pca_3d[df_pca_3d['Cluster'] == cluster]
    ax.scatter(
        cluster_points['PC1'],
        cluster_points['PC2'],
        cluster_points['PC3'],
        c=[palette[cluster]],
        label=f'Cluster {cluster}',
        s=50,
        alpha=0.8
    )

# Adjust plot details
ax.set_title('3D Clustering Visualization with PCA', fontsize=16)
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.legend(loc='best', title='Clusters')
plt.show()
```

In this analysis, we will evaluate the quality of the clustering using the **Silhouette Score**. The Silhouette Score measures how similar each point is to its own cluster compared to other clusters. A higher score indicates that the clusters are well-separated, while a lower score suggests that the clustering might not be optimal.

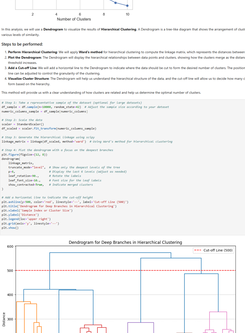
Steps to be performed:

1. **Compute the Silhouette Score (average):** This will provide an overall measure of how well separated the clusters are.
2. **Calculate Silhouette Scores for each point:** We will compute individual Silhouette Scores for each data point to understand how well each point fits within its assigned cluster.
3. **Plot the Silhouette Score:** The plot will display the distribution of Silhouette Scores for each cluster, along with a red dashed line indicating the average Silhouette Score across all clusters.
4. **Cluster Statistics:** For each cluster, we will calculate and display statistics such as the mean, median, minimum, and maximum Silhouette Scores.

This will allow us to assess the effectiveness of the clustering algorithm and determine if the clusters are well-defined or if adjustments are needed.

[12]: # Step 1: Filter numerical columns
```
numeric_columns = df_scaled_pca_features.select_dtypes(include=['float64', 'int64']).columns
df_numeric = df[numeric_columns]

# Step 2: Scale the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_numeric)

# Step 3: Perform clustering using MiniBatchKMeans
from sklearn.cluster import MiniBatchKMeans
mini_kmeans = MiniBatchKMeans(n_clusters=4, random_state=42, batch_size=1000)
clusters = mini_kmeans.fit_predict(df_scaled)

# Step 4: Calculate the Silhouette Score (average)
silhouette_avg = silhouette_score(df_scaled, clusters)
print("Average Silhouette Score: {:.2f}".format(silhouette_avg))

# Step 5: Calculate Silhouette Scores for each point
silhouette_vals = silhouette_samples(df_scaled, clusters)

# Step 6: Initialize a DataFrame to store statistics for each cluster
cluster_stats = {}

# Plot Silhouette Scores
plt.figure(figsize=(8, 6))
y_lower = 0
for i in range(4):
    # Aggregate silhouette scores for the current cluster
    cluster_silhouette_vals = silhouette_vals[clusters == i]
    cluster_silhouette_vals.sort()
    y_upper = y_lower + len(cluster_silhouette_vals)

    # Bar plot for the silhouette scores of the current cluster
    plt.barh(
        range(y_lower, y_upper),
        cluster_silhouette_vals,
        edgecolor='none',
        height=1
    )

    # Update the lower bound for the next cluster
    y_lower = y_upper

    # Calculate statistics for the cluster
    cluster_stats[i] = {
        'Cluster': i,
        'Mean': cluster_silhouette_vals.mean(),
        'Median': np.median(cluster_silhouette_vals),
        'Min': cluster_silhouette_vals.min(),
        'Max': cluster_silhouette_vals.max(),
        'Count': len(cluster_silhouette_vals)
    }

# Add a red dashed line to indicate the average silhouette score
plt.axvline(x=silhouette_avg, color='red', linestyle='--', label='Average Silhouette Score')

# Adjust plot details
plt.title('Silhouette Plot for Clusters')
plt.xlabel('Silhouette Score')
plt.ylabel('Data Points')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

# Convert cluster statistics to a DataFrame and display
cluster_stats_df = pd.DataFrame(cluster_stats).T
print("Cluster Silhouette Statistics:")
print(cluster_stats_df)
```

```
Average Silhouette Score: 0.27
```

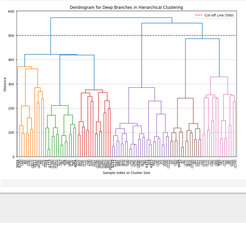



```
Cluster Silhouette Statistics:
   Cluster      Mean    Median       Min       Max  Count
0        0  0.270000  0.270000 -0.400000  0.800000      1
1        1  0.270000  0.270000 -0.400000  0.800000      1
2        2  0.270000  0.270000 -0.400000  0.800000      1
3        3  0.270000  0.270000 -0.400000  0.800000      1
```

In this analysis, we will use the **Elbow Method** to determine the optimal number of clusters for our dataset. The Elbow Method helps identify the point at which increasing the number of clusters no longer significantly improves the clustering quality.

This method will help us visually assess the total number of clusters to use for the K-Means algorithm.

[13]: # Step 1: Filter numerical columns
```
numeric_columns = df_scaled_pca_features.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
df_numeric = df_scaled_pca_features[numeric_columns]

# Step 2: Scale data
# Using standard scaler for consistent results
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_numeric)

# Step 3: Calculate inertia for different numbers of clusters
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow method
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.grid(True)

# Find the elbow point (simple method)
# This is a simple way to find the elbow point
from numpy import diff, argmax
second_derivative = diff(diff(inertia))
elbow_point = argmax(second_derivative) + 2
print("Suggested number of clusters: {}".format(elbow_point))
```



In this analysis, we will use a **Dendrogram** to visualize the results of **Hierarchical Clustering**. A Dendrogram is a tree-like diagram that shows the arrangement of clusters at various levels of similarity.

Steps to be performed:

1. **Perform Hierarchical Clustering:** We will apply **Ward's method** for hierarchical clustering to compute the linkage matrix, which represents the distances between clusters.
2. **Plot the Dendrogram:** The Dendrogram will display the hierarchical relationships between data points and clusters, showing how the clusters merge as the distance threshold increases.
3. **Add a Cut-off Line:** We will add a horizontal line to the Dendrogram to indicate where the data should be cut to form the desired number of clusters. The position of the line can be adjusted to control the granularity of the clustering.
4. **Visualize Cluster Branches:** The Dendrogram will help us understand the hierarchical structure of the data, and the cut-off line will allow us to decide how many clusters to form based on the hierarchy.

This method will provide us with a clear understanding of how clusters are related and help us determine the optimal number of clusters.

[14]: # Step 1: Take a representative sample of the dataset (adjust size based on your memory)
```
df_sample = df.sample(n=5000, random_state=42) # Adjust the sample size according to your dataset
numeric_columns = df_sample.select_dtypes(include=['float64', 'int64'])

# Step 2: Scale the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(numeric_columns)

# Step 3: Generate the hierarchical linkage using ward method
linkage_matrix = linkage(df_scaled, method='ward')

# Step 4: Plot the dendrogram with a focus on the deepest branches
plt.figure(figsize=(15, 8))
dendrogram(
    linkage_matrix,
    truncate_mode='lastp',  # Show only the deepest branches
    p=30,                   # Display the last 30 clusters (adjust as needed)
    leaf_rotation=90.,      # Rotate the labels
    leaf_font_size=10.,     # Adjust font size
    show_contracted=True,   # Indicate merged clusters
)

# Add a horizontal line to indicate the cut-off height
plt.axhline(y=500, color='red', linestyle='--', label='Cut-off Line (500)')
plt.title('Dendrogram for Deep Branches in Hierarchical Clustering')
plt.xlabel('Sample Index or Cluster Size')
plt.ylabel('Distance')
plt.legend(loc='upper right')
plt.grid(True, axis='y')
plt.show()
```