Clustering For Dataset Exploration

Created	@October 22, 2024 10:57 PM
Class	Unsupervised Learning with Python

Introduction to Unsupervised Learning in Python

Prerequisites

- Understanding of basic machine learning concepts.

- Familiarity with Python and libraries like `scikit-learn` and `matplotlib`.

Key Details

- Focus: **Unsupervised Learning** for discovering patterns in data without labeled outcomes.
- Techniques: **Clustering** and **Dimensionality Reduction**.
- Tools: Python libraries like `scikit-learn` and `matplotlib`.

What is Unsupervised Learning?

Unsupervised learning is a type of machine learning used to uncover patterns in data without predefined labels. It aims to discover hidden structures or relationships within the data. Unlike supervised learning, where models are trained with labeled data to make predictions, unsupervised learning works with unlabeled data to identify patterns.

Examples of Unsupervised Learning:

1. **Clustering**: Identifying natural groups of customers based on purchase history.
2. **Dimension Reduction**: Finding patterns and correlations in high-dimensional data and compressing it into a simpler form.

Mathematical Representation:

For unsupervised learning, the task is to map the data X to patterns Z:

$$X \mapsto Z$$

Clustering

Clustering is one of the most common techniques in unsupervised learning. It involves grouping data points that share similar characteristics.

k-Means Clustering

k-Means is a simple and widely used clustering algorithm that partitions the dataset into **k** clusters.

1. The algorithm assigns each data point to the cluster with the nearest centroid.
2. Centroids are updated iteratively to minimize the within-cluster variance.
3. New data points are assigned to the cluster of the nearest centroid using the distance measure.

Algorithmic Steps:

1. Initialize k random centroids.
2. Assign each data point to its nearest centroid.
3. Recompute the centroid of each cluster as the mean of all points in the cluster.
4. Repeat steps 2 and 3 until convergence.

Mathematically:

1. Compute the Euclidean distance between each data point and the centroids:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

2. Assign each point to the cluster of the nearest centroid.
3. Recalculate the centroids:

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

where
μ_j is the new centroid and C_j is the set of points assigned to cluster j.

Using k-Means with `scikit-learn`

```
from sklearn.cluster import KMeans
```

```
# Fit the model
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

# Predict the clusters for new samples
new_samples = np.array([[5.0, 3.2], [4.5, 2.5]])
predictions = kmeans.predict(new_samples)
print(predictions)
```

Cluster Visualization with `matplotlib`

To visualize clusters, scatter plots are commonly used. For example, using the Iris dataset:

```
import matplotlib.pyplot as plt

# Example of scatter plot for sepal length vs petal length
sepal_length = X[:, 0]
petal_length = X[:, 2]

# Plotting with cluster labels
plt.scatter(sepal_length, petal_length, c=kmeans.labels_)
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.title('Iris Clusters')
plt.show()
```

Dimensionality Reduction

Dimensionality reduction is a technique used to reduce the number of features in the dataset while preserving the structure of the data. This helps in visualizing complex, high-dimensional data and speeds up computation.

Principal Component Analysis (PCA)

One of the most common dimensionality reduction techniques is **Principal Component Analysis (PCA)**, which transforms the data into a lower-dimensional space by identifying the directions (principal components) that capture the most variance in the data.

Mathematically:

The PCA transformation is given by:

$$Z = XW$$

Where
W are the principal components (eigenvectors of the covariance matrix of X).

PCA Example with `scikit-learn`

```
from sklearn.decomposition import PCA

# Reduce to 2 dimensions
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Plot the reduced dataset
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=labels)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.show()
```

Evaluating Clustering Results

Once clustering is performed, it is important to evaluate the quality of the clusters.

Metrics for Clustering Evaluation:

1. **Inertia**: Measures how tightly the data points are clustered around their centroids. Lower values indicate better clustering.

$$\text{Inertia} = \sum_{i=1}^{n} \sum_{j=1}^{k} \left\| x_i - \mu_j \right\|^2$$

2. **Silhouette Score**: Measures how similar a data point is to its own cluster compared to other clusters. The value ranges from -1 to 1, where higher values indicate better-defined clusters.

```
from sklearn.metrics import silhouette_score

# Calculate silhouette score
score = silhouette_score(X, kmeans.labels_)
print(f'Silhouette Score: {score}')
```

Visualization of Clustering

Using a scatter plot to visualize clustering results is a crucial step in evaluating the performance of the clustering algorithm. For example, visualizing k-Means clusters using sepal length vs. petal length:

```
import matplotlib.pyplot as plt

# Scatter plot
plt.scatter(sepal_length, petal_length, c=kmeans.labels_, cma
p='viridis')
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.title('Cluster Visualization')
plt.show()
```

Conclusion

- **Unsupervised Learning** is a powerful tool for finding patterns in data without the need for labels.
- **Clustering** methods like k-Means allow us to discover natural groupings in the data.
- **Dimensionality Reduction** techniques like PCA help in simplifying complex datasets while preserving important structures.
- Proper evaluation of clustering using metrics like inertia and silhouette score is essential to ensure meaningful results.

Evaluating Clustering Quality in K-means

Prerequisites

- Previous knowledge: K-means clustering implementation on iris dataset
- Required libraries: pandas, scikit-learn

Key Details

- Focus: Methods to evaluate clustering quality
- Dataset: Iris dataset with 3 species
- Main approaches:
    1. Direct comparison with known labels
    2. Inertia-based evaluation

Cluster Evaluation Methods

1. Cross-tabulation Analysis

- Compares clusters with actual species/labels
- Uses a contingency table to show correspondence

Implementation with Pandas

```
import pandas as pd

# Create DataFrame with cluster labels and species
df = pd.DataFrame({
    'cluster_labels': kmeans.labels_,
    'species': iris_species
})

# Generate cross-tabulation
cross_tab = pd.crosstab(df['cluster_labels'], df['species'])
```

Example Output Format:

```
species      setosa  versicolor  virginica
cluster
0                 0          48          2
1                50           0          0
2                 0           2         48
```

2. Inertia-based Evaluation

- Measures cluster tightness
- Lower values indicate better clustering

Mathematical Formulation

The inertia (I) is defined as:

$$I = \sum_{i=1}^{n} \min_{\mu_j \in C}(||x_i - \mu_j||^2)$$

Where:

- x_i is the i^{th} sample
- μ_j is the centroid of cluster j
- C is the set of all cluster centroids

- $||x_i - \mu_j||^2$ is the squared Euclidean distance

Implementation

```
from sklearn.cluster import KMeans

# Calculate inertia for different k values
inertias = []
k_range = range(1, 10)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)

# Visualize elbow curve
import matplotlib.pyplot as plt

plt.plot(k_range, inertias, 'bx-')
plt.xlabel('k')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```

Choosing Optimal Number of Clusters

Elbow Method

1. Plot inertia vs. number of clusters
2. Look for "elbow point" where inertia decrease slows
3. Select corresponding k value

Mathematical Interpretation

The rate of change in inertia can be expressed as:

$$\Delta I_k = I_k - I_{k+1}$$

The elbow point occurs where:

$$\frac{\Delta I_k}{\Delta I_{k+1}} \approx 1$$

Key Takeaways

- Cross-tabulation provides insights when true labels are available
- Inertia measures within-cluster sample spread
- Lower inertia indicates tighter clusters
- Optimal k balances low inertia with cluster count
- Elbow method helps identify optimal k

Mathematical Concept Integration

Relationship between Cross-tabulation and Inertia

The correlation between cross-tabulation accuracy and inertia can be expressed as:

$$\text{correlation} = -\rho(A, I)$$

Where:

- A is the clustering accuracy from cross-tabulation
- I is the inertia
- ρ is the correlation coefficient

A strong negative correlation indicates that as inertia decreases, clustering accuracy tends to increase.

Feature Scaling in K-means Clustering

Prerequisites

- Understanding of K-means clustering

- Basic knowledge of variance and data distributions

Key Details

- Focus: Importance of feature scaling in clustering
- Dataset: Piedmont wines (178 samples, 3 varieties)
- Features: Chemical composition and visual properties

Feature Variance Problem

Issue

- Features with different variances affect clustering disproportionately
- Higher variance features dominate the clustering process

Mathematical Representation

For a feature X, variance is calculated as:

$\text{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$

Where:

- n is the number of samples
- x_i is the i^{th} observation
- μ is the mean of the feature

Standardization Solution

StandardScaler Implementation

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Create and fit the scaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Apply KMeans to scaled data
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
```

Mathematical Transformation

For each feature X, StandardScaler performs:

$X_{scaled} = \frac{X-\mu}{\sigma}$

Where:

- μ is the mean of the feature
- σ is the standard deviation
- Resulting features have $\mu = 0$ and $\sigma = 1$

Pipeline Implementation

Using scikit-learn Pipeline

```
from sklearn.pipeline import make_pipeline

# Create pipeline
pipeline = make_pipeline(
    StandardScaler(),
    KMeans(n_clusters=3, random_state=42)
)

# Fit and predict
clusters = pipeline.fit_predict(X)

# Evaluate with cross-tabulation
cross_tab = pd.crosstab(clusters, wine_varieties)
```

Pipeline Flow Mathematics

Given input data X:

1. Standardization: $X_{std} = \frac{X-\mu}{\sigma}$
2. K-means clustering on X_{std}:
   - Minimize $\sum_{i=1}^{n} \min_{\mu_j \in C}(||x_{std_i} - \mu_j||^2)$

Evaluation Methods

Cross-tabulation Analysis

```
import pandas as pd

# Compare clusters with wine varieties
cross_tab = pd.crosstab(
    clusters,
    wine_varieties,
    margins=True
)
```

Improvement Metrics

Clustering accuracy can be measured as:

$$\text{accuracy} = \frac{\text{correct\_assignments}}{\text{total\_samples}}$$

Where correct assignments are determined by the majority class in each cluster.

Alternative Scaling Methods

1. MaxAbsScaler:

$$X_{scaled} = \frac{X}{\max(|X|)}$$

2. Normalizer:

$$X_{normalized} = \frac{X}{||X||}$$

Key Takeaways

- Feature scaling is crucial for balanced clustering
- StandardScaler transforms features to mean=0, variance=1
- Pipelines streamline preprocessing and clustering
- Proper scaling significantly improves cluster-class correspondence
- Multiple scaling options available based on data characteristics

Mathematical Concept Integration

Impact of Scaling on Distance Metrics

The Euclidean distance between points p and q after scaling:

$$d_{scaled}(p, q) = \sqrt{\sum_{i=1}^{n} (\frac{p_i - \mu_i}{\sigma_i} - \frac{q_i - \mu_i}{\sigma_i})^2}$$

This ensures that:

1. All features contribute equally to distance calculations
2. Clustering is not biased by feature magnitudes
3. Results better reflect natural data groupings