



# Project AI

# supervized ML




Supervised learning is a type of machine learning where the computer is taught using examples. These examples have both the input (what you know) and the output (what you want to predict). The goal is for the computer to learn how to predict the output based on the input.





# process :

1. Data Collection: Gather a labeled dataset, where each example consists of an input-output pair.
  2. Data Preprocessing: Clean the data, handle missing values, normalize or scale features, and possibly perform feature engineering.
  3. Train-Test Split: Split the dataset into training and testing sets to evaluate the model's performance.
  4. Model Selection: Choose a suitable algorithm for the problem at hand (e.g., linear regression for regression tasks, decision trees for classification tasks).
  5. Training: Use the training data to train the model, adjusting the model's parameters to minimize the error between predicted and actual outputs.
  6. Evaluation: Assess the model's performance on the test set using appropriate metrics (e.g., accuracy, precision, recall for classification; mean squared error, R-squared for regression).
  7. Hyperparameter Tuning: Optimize the model's hyperparameters to improve performance.
  8. Prediction: Use the trained model to make predictions on new, unseen data.
- 

# Algorithm used in this project :

for classifying images of cats and dogs, we extract features from the images and then use traditional machine learning algorithms such as Decision tree and k-Nearest Neighbors (k-NN).

here is a steps for what we do ..

## 1- import libraries:

```
1 import os
2 import numpy as np
3 from tensorflow.keras.preprocessing.image import load_img, img_to_array
4 from sklearn.preprocessing import LabelEncoder
5 from sklearn.model_selection import train_test_split
6 from sklearn.neighbors import KNeighborsClassifier
7 from sklearn.tree import DecisionTreeClassifier
8 from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
9 import matplotlib.pyplot as plt
```

Your paragraph text

## 2- Load and Preprocess Images:

```
1 # Set image parameters
2 img_height = 128
3 img_width = 128
4 channels = 3
5 num_classes = 2 # Adjusted to match the number of classes in lab dictionary
6
7 # Dictionary to label the images
8 lab = {'dogs': 0, 'cats': 1}
9
10 def load_images_from_folder(folder_path, label_dict):
11     data = []
12     labels = []
13     print(f"Loading images from folder: {folder_path}")
14     for folder in os.listdir(folder_path):
15         folder_path_full = os.path.join(folder_path, folder)
16         if not os.path.isdir(folder_path_full):
17             print(f"Skipping {folder_path_full} as it is not a directory.")
18             continue
19         print(f"Processing folder: {folder}")
20         for filename in os.listdir(folder_path_full):
21             file_path = os.path.join(folder_path_full, filename)
22             if os.path.isfile(file_path):
23                 if filename.lower().endswith(('png', '.jpg', '.jpeg', '.bmp', '.gif')):
24                     try:
25                         img = load_img(file_path, target_size=(img_height, img_width))
26                         img_array = img_to_array(img)
27                         data.append(img_array.flatten()) # Flatten the image array
28                         labels.append(label_dict[folder])
29                     except Exception as e:
```

### 3-Train decision tree Model and k-nn model and evaluate them :

```
69 dt_model = DecisionTreeClassifier() # You can adjust parameters as needed
70 dt_model.fit(x_train, y_train)
71
72 # Predict and evaluate Decision Tree Classifier
73 y_pred_dt = dt_model.predict(x_test)
74
75 print("----Decision Tree Classifier evaluation:-----")
76 print("Accuracy:", accuracy_score(y_test, y_pred_dt))
77 print("Confusion matrix:")
78 print(confusion_matrix(y_test, y_pred_dt))
79 print("Classification report:")
80 print(classification_report(y_test, y_pred_dt))
81
82
83 # Define and train K-Neighbors Classifier
84 knn_model = KNeighborsClassifier(n_neighbors=5) # You can adjust `n_neighbors`
85 knn_model.fit(x_train, y_train)
86
87 # Predict and evaluate K-Neighbors Classifier
88 y_pred_knn = knn_model.predict(x_test)
89 print("----K-Neighbors Classifier evaluation:-----")
90 print("Accuracy:", accuracy_score(y_test, y_pred_knn))
91 print("Confusion matrix:")
92 print(confusion_matrix(y_test, y_pred_knn))
93 print("Classification report:")
94 print(classification_report(y_test, y_pred_knn))
```

Sample Image label: 1

----Decision Tree Classifier evaluation:-----

Accuracy: 0.5034562211981567

Confusion matrix:

		0	1
0	299	328	
1	103	138	

Classification report:

	precision	recall	f1-score	support
0	0.74	0.48	0.58	627
1	0.30	0.57	0.39	241
accuracy			0.50	868
macro avg	0.52	0.52	0.49	868
weighted avg	0.62	0.50	0.53	868

----K-Neighbors Classifier evaluation:-----

Accuracy: 0.4735023041474654

Confusion matrix:

		0	1
0	222	405	
1	52	189	

Classification report:

	precision	recall	f1-score	support
0	0.81	0.35	0.49	627
1	0.32	0.78	0.45	241
accuracy			0.47	868
macro avg	0.56	0.57	0.47	868
weighted avg	0.67	0.47	0.48	868



```
File Edit Search Source Run Debug Consoles Projects Tools View Help
F:\ERLAL_Project\CarAndDog\temp.py

1 import os
2 import numpy as np
3 from tensorflow.keras.preprocessing.image import load_img, img_to_array
4 from sklearn.preprocessing import LabelEncoder
5 from sklearn.model_selection import train_test_split
6 from sklearn.neighbors import KNeighborsClassifier
7 from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
8 import matplotlib.pyplot as plt
9
10 # Set image parameters
11 img_height = 128
12 img_width = 128
13 channels = 3
14 num_classes = 2 # Adjusted to match the number of classes in lab dictionary
15
16 # Dictionary to label the images
17 lab = {'dogs': 0, 'cats': 1}
18
19 def load_images_from_folder(folder_path, label_dict):
20     data = []
21     labels = []
22     print(f"Loading images from folders: {folder_path}")
23     for folder in os.listdir(folder_path):
24         folder_path_full = os.path.join(folder_path, folder)
25         if not os.path.isdir(folder_path_full):
26             print(f"Skipping {folder_path_full} as it is not a directory.")
27             continue
28         print(f"Processing folder: {folder}")
29         for filename in os.listdir(folder_path_full):
30             file_path = os.path.join(folder_path_full, filename)
31             if os.path.isfile(file_path):
32                 if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):
33                     try:
34                         img = load_img(file_path, target_size=(img_height, img_width))
35                         data.append(img_array(img).flatten()) # Flatten the image array
36                         labels.append(label_dict[folder])
37                     except Exception as e:
38                         print(f"Error loading image {file_path}: {e}")
39
40 except Exception as e:
41     print(f"Error loading image {file_path}: {e}")
42 else:
43     print(f"Skipping non-image file {file_path}.")
44 print(f"Loaded {len(data)} images from {folder_path}")
45 return np.array(data) / 255.0, np.array(labels)
46
47 # Load training data
48 dataset_path1 = 'Data/Train'
49 x_train, y_train = load_images_from_folder(dataset_path1, lab)
50
51 # Load testing data
52 dataset_path2 = 'Data/Test'
53 x_test, y_test = load_images_from_folder(dataset_path2, lab)
54
55 # Check if data is loaded correctly
56 print(f"Training data shape: {x_train.shape}")
57 print(f"Training labels shape: {y_train.shape}")
58 print(f"Testing data shape: {x_test.shape}")
59 print(f"Testing labels shape: {y_test.shape}")
60
61 # Ensure that the data is not empty
62 if x_train.size == 0 or y_train.size == 0 or x_test.size == 0 or y_test.size == 0:
63     raise ValueError("One of the data arrays is empty. Please check your data loading process.")
64
65 # Debug output
66 print(f"Sample image data shape: {x_train[0].shape}")
67 print(f"Sample image label: {y_train[0]}")
68
69 # Define and train Decision Tree Classifier
70 dt_model = DecisionTreeClassifier() # You can adjust parameters as needed
71 dt_model.fit(x_train, y_train)
72
73 # Predict and evaluate Decision Tree Classifier
74 y_pred_dt = dt_model.predict(x_test)
75
76 print("----Decision Tree Classifier evaluation:-----")
77 print("Accuracy:", accuracy_score(y_test, y_pred_dt))
78 print("Confusion matrix:")
79 print(confusion_matrix(y_test, y_pred_dt))
80 print("Classification report:")
81 print(classification_report(y_test, y_pred_dt))
82
83 # Define and train K-Neighbors Classifier
84 knn_model = KNeighborsClassifier(n_neighbors=5) # You can adjust 'n_neighbors'
85 knn_model.fit(x_train, y_train)
86
87 # Predict and evaluate K-Neighbors Classifier
88 y_pred_knn = knn_model.predict(x_test)
89
90 print("----K-Neighbors Classifier evaluation:-----")
91 print("Accuracy:", accuracy_score(y_test, y_pred_knn))
92 print("Confusion matrix:")
93 print(confusion_matrix(y_test, y_pred_knn))
94 print("Classification report:")
95 print(classification_report(y_test, y_pred_knn))
```

Name	Type	Size	Value
channels	int	1	3

```
Sample image label: 1
----Decision Tree Classifier evaluation:-----
Accuracy: 0.5834562211981567
Confusion matrix:
[[299 328]
 [103 138]]
Classification report:
precision    recall  f1-score   support

0       0.74      0.48      0.58      627
1       0.30      0.57      0.39      241

accuracy      0.52      0.52      0.50      868
macro avg     0.52      0.52      0.49      868
weighted avg  0.62      0.50      0.53      868

----K-Neighbors Classifier evaluation:-----
Accuracy: 0.4735023041474654
Confusion matrix:
[[222 465]
 [ 52 189]]
Classification report:
precision    recall  f1-score   support

0       0.81      0.35      0.49      627
1       0.32      0.78      0.45      241

accuracy      0.56      0.57      0.47      868
macro avg     0.67      0.47      0.48      868
weighted avg  0.67      0.47      0.48      868

In [4]:
```

```
File Edit Search Source Run Debug Consoles Projects Tools View Help
F:\ERLAL_Project\CarAndDog\temp.py

39
40 except Exception as e:
41     print(f"Error loading image {file_path}: {e}")
42 else:
43     print(f"Skipping non-image file {file_path}.")
44 print(f"Loaded {len(data)} images from {folder_path}")
45 return np.array(data) / 255.0, np.array(labels)
46
47 # Load training data
48 dataset_path1 = 'Data/Train'
49 x_train, y_train = load_images_from_folder(dataset_path1, lab)
50
51 # Load testing data
52 dataset_path2 = 'Data/Test'
53 x_test, y_test = load_images_from_folder(dataset_path2, lab)
54
55 # Check if data is loaded correctly
56 print(f"Training data shape: {x_train.shape}")
57 print(f"Training labels shape: {y_train.shape}")
58 print(f"Testing data shape: {x_test.shape}")
59 print(f"Testing labels shape: {y_test.shape}")
60
61 # Ensure that the data is not empty
62 if x_train.size == 0 or y_train.size == 0 or x_test.size == 0 or y_test.size == 0:
63     raise ValueError("One of the data arrays is empty. Please check your data loading process.")
64
65 # Debug output
66 print(f"Sample image data shape: {x_train[0].shape}")
67 print(f"Sample image label: {y_train[0]}")
68
69 # Define and train Decision Tree Classifier
70 dt_model = DecisionTreeClassifier() # You can adjust parameters as needed
71 dt_model.fit(x_train, y_train)
72
73 # Predict and evaluate Decision Tree Classifier
74 y_pred_dt = dt_model.predict(x_test)
75
76 print("----Decision Tree Classifier evaluation:-----")
77 print("Accuracy:", accuracy_score(y_test, y_pred_dt))
78 print("Confusion matrix:")
79 print(confusion_matrix(y_test, y_pred_dt))
80 print("Classification report:")
81 print(classification_report(y_test, y_pred_dt))
82
83 # Define and train K-Neighbors Classifier
84 knn_model = KNeighborsClassifier(n_neighbors=5) # You can adjust 'n_neighbors'
85 knn_model.fit(x_train, y_train)
86
87 # Predict and evaluate K-Neighbors Classifier
88 y_pred_knn = knn_model.predict(x_test)
89
90 print("----K-Neighbors Classifier evaluation:-----")
91 print("Accuracy:", accuracy_score(y_test, y_pred_knn))
92 print("Confusion matrix:")
93 print(confusion_matrix(y_test, y_pred_knn))
94 print("Classification report:")
95 print(classification_report(y_test, y_pred_knn))
```

Name	Type	Size	Value
channels	int	1	3

```
Sample image label: 1
----Decision Tree Classifier evaluation:-----
Accuracy: 0.5834562211981567
Confusion matrix:
[[299 328]
 [103 138]]
Classification report:
precision    recall  f1-score   support

0       0.74      0.48      0.58      627
1       0.30      0.57      0.39      241

accuracy      0.52      0.52      0.50      868
macro avg     0.52      0.52      0.49      868
weighted avg  0.62      0.50      0.53      868

----K-Neighbors Classifier evaluation:-----
Accuracy: 0.4735023041474654
Confusion matrix:
[[222 465]
 [ 52 189]]
Classification report:
precision    recall  f1-score   support

0       0.81      0.35      0.49      627
1       0.32      0.78      0.45      241

accuracy      0.56      0.57      0.47      868
macro avg     0.67      0.47      0.48      868
weighted avg  0.67      0.47      0.48      868

In [4]:
```

```
File Edit Search Source Run Debug Consoles Projects Tools View Help
F:\ERLAL_Project\CarAndDog\temp.py

56 print(f"Training labels shape: {y_train.shape}")
57 print(f"Testing labels shape: {y_test.shape}")
58 print(f"Testing labels shape: {y_test.shape}")
59
60 # Ensure that the data is not empty
61 if x_train.size == 0 or y_train.size == 0 or x_test.size == 0 or y_test.size == 0:
62     raise ValueError("One of the data arrays is empty. Please check your data loading process.")
63
64 # Debug output
65 print(f"Sample image data shape: {x_train[0].shape}")
66 print(f"Sample image label: {y_train[0]}")
67
68 # Define and train Decision Tree Classifier
69 dt_model = DecisionTreeClassifier() # You can adjust parameters as needed
70 dt_model.fit(x_train, y_train)
71
72 # Predict and evaluate Decision Tree Classifier
73 y_pred_dt = dt_model.predict(x_test)
74
75 print("----Decision Tree Classifier evaluation:-----")
76 print("Accuracy:", accuracy_score(y_test, y_pred_dt))
77 print("Confusion matrix:")
78 print(confusion_matrix(y_test, y_pred_dt))
79 print("Classification report:")
80 print(classification_report(y_test, y_pred_dt))
81
82 # Define and train K-Neighbors Classifier
83 knn_model = KNeighborsClassifier(n_neighbors=5) # You can adjust 'n_neighbors'
84 knn_model.fit(x_train, y_train)
85
86 # Predict and evaluate K-Neighbors Classifier
87 y_pred_knn = knn_model.predict(x_test)
88
89 print("----K-Neighbors Classifier evaluation:-----")
90 print("Accuracy:", accuracy_score(y_test, y_pred_knn))
91 print("Confusion matrix:")
92 print(confusion_matrix(y_test, y_pred_knn))
93 print("Classification report:")
94 print(classification_report(y_test, y_pred_knn))
```

Name	Type	Size	Value
channels	int	1	3
dataset_path1	str	10	Data/Train
dataset_path2	str	9	Data/Test
dt_model	tree_classes.DecisionTreeClassifier	1	DecisionTreeCL...
img_height	int	1	128
img_width	int	1	128
knn_model	neighbors_classification.KNeighborsClassifier	1	KNeighborsClas...
lab	dict	2	{'dogs': 0, 'cats': 1}
num_classes	int	1	2
x_test	Array of float32	(868, 49152)	[16.8352941 8...
x_train	Array of float32	(3341, 49152)	[10.998839 0...
y_pred_dt	Array of int32	(868,)	[0 1 0 ... 1 1]
y_pred_knn	Array of int32	(868,)	[1 1 1 ... 0 0]
y_test	Array of int32	(868,)	[1 1 1 ... 0 0]
y_train	Array of int32	(3341,)	[1 1 1 ... 0 0]

```
Sample image label: 1
----Decision Tree Classifier evaluation:-----
Accuracy: 0.5834562211981567
Confusion matrix:
[[299 328]
 [103 138]]
Classification report:
```

# Unsupervised ML

Used Histogram of Oriented Gradients (HOG) for feature extraction, Principal Component Analysis (PCA) for dimensionality reduction, and KMeans clustering for grouping similar images. The project evaluates the clustering results against the ground truth labels and visualizes the clusters and classification results. (How ever because the images are complex this approach wasn't the best for this case)

## Data Preprocessing

### Image Loading and Preprocessing

Images are loaded from separate folders for training and testing datasets. Each image is resized to 64x64 pixels and normalized by dividing pixel values by 255.0. The labels are assigned as 0 for dogs and 1 for cats.

```
12 # Function to load images from a folder and preprocess them
13 def load_images_and_labels_from_folder(folder, image_size=(64, 64)):
14     images = []
15     labels = []
16     for label, subfolder in enumerate(['dogs', 'cats']):
17         subfolder_path = os.path.join(folder, subfolder)
18         for filename in os.listdir(subfolder_path):
19             if filename.endswith(".jpg") or filename.endswith(".png"):
20                 img_path = os.path.join(subfolder_path, filename)
21                 img = io.imread(img_path)
22                 if img is not None:
23                     img = cv2.resize(img, image_size)
24                     img = img / 255.0 # Normalize the image
25                     images.append(img)
26                     labels.append(label) # 0 for dogs, 1 for cats
27     return np.array(images), np.array(labels)
28
```

## HOG Feature Extraction

HOG features are extracted from the grayscale images. The HOG descriptor captures the edge and gradient structure of the images, which is useful for distinguishing between different object categories.

python

```
29 # Extract HOG features from images
30 def extract_hog_features(images, pixels_per_cell=(16, 16), cells_per_block=(2, 2),
    visualize=False):
31     hog_features = []
32     for img in images:
33         # Convert to grayscale
34         img_gray = color.rgb2gray(img)
35         if visualize:
36             features, _ = hog(img_gray, pixels_per_cell=pixels_per_cell,
37                               cells_per_block=cells_per_block, visualize=visualize)
38         else:
39             features = hog(img_gray, pixels_per_cell=pixels_per_cell,
40                             cells_per_block=cells_per_block, visualize=visualize)
41         hog_features.append(features)
42     return np.array(hog_features)
43
```

## Standardization and Scaling

The extracted HOG features are standardized using StandardScaler and optionally scaled to a [0, 1] range using MinMaxScaler.

```
60 # Standardize the HOG features
61 scaler_standard = StandardScaler()
62 train_hog_features_standard_scaled = scaler_standard.fit_transform(train_hog_features)
63 test_hog_features_standard_scaled = scaler_standard.transform(test_hog_features)
64
65 # Apply MinMaxScaler
66 scaler_minmax = MinMaxScaler(feature_range=(0, 1))
67 train_hog_features_minmax_scaled = scaler_minmax.fit_transform(train_hog_features)
68 test_hog_features_minmax_scaled = scaler_minmax.transform(test_hog_features)
69
```

## Dimensionality Reduction

PCA is applied to the standardized HOG features to reduce dimensionality and retain 50 principal components. This step helps in reducing the computational cost and noise while preserving the essential features for clustering.

```
70 # Apply PCA for feature extraction
71 pca = PCA(n_components=50)
72 train_hog_features_pca = pca.fit_transform(train_hog_features_standard_scaled)
73 test_hog_features_pca = pca.transform(test_hog_features_standard_scaled)
74 print(f"Training PCA shape: {train_hog_features_pca.shape}")
75 print(f"Testing PCA shape: {test_hog_features_pca.shape}")
76
```



## Clustering

### Determining Optimal Number of Clusters

The elbow method is used to determine the optimal number of clusters by plotting the within-cluster sum of squares (inertia) for different values of k.

```
77 # Determine the optimal number of clusters using the elbow method
78 inertia = []
79 for i in range(1, 11):
80     kmeans = KMeans(n_clusters=i, init='k-means++', n_init=10, random_state=0)
81     kmeans.fit(train_hog_features_pca)
82     inertia.append(kmeans.inertia_)
83
84 # Plot the elbow graph
85 plt.figure(figsize=(10, 6))
86 plt.plot(range(1, 11), inertia, marker='o')
87 plt.title('Elbow Method For Optimal k')
88 plt.xlabel('Number of clusters')
89 plt.ylabel('Within-cluster Sum of Squares')
90 plt.show()
```

### KMeans Clustering

Based on the elbow graph, the optimal number of clusters is chosen, and KMeans clustering is applied to the PCA-transformed features.

```
92 # Choose the optimal number of clusters based on the elbow graph
93 optimal_clusters = 2 # Set this to the number of clusters identified by the elbow method
94
95 # Apply KMeans clustering with the optimal number of clusters
96 kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', n_init=10, random_state=0)
97 train_clusters = kmeans.fit_predict(train_hog_features_pca)
98 test_clusters = kmeans.predict(test_hog_features_pca)
99
```

### Evaluation

The clustering results are evaluated against the ground truth labels using confusion matrices and classification reports. Clusters are relabeled to align with the actual labels for better interpretability.

```
100 # Print evaluation metrics
101 print("Ground truth clusters:\n", test_labels)
102 print("Estimated clusters:\n", test_clusters)
103 print("Confusion matrix:\n", confusion_matrix(test_labels, test_clusters))
104 print("Classification report:\n", classification_report(test_labels, test_clusters))
105
106 # Relabel clusters
107 relabel_clusters = [1 if cluster == 0 else 0 for cluster in test_clusters]
108
109 print("Relabel clusters:\n", relabel_clusters)
110 print("Confusion matrix (relabelled):\n", confusion_matrix(test_labels, relabel_clusters))
111 print("Classification report (relabelled):\n", classification_report(test_labels,
112                                relabel_clusters))
```

## Visualization

### PCA Component Plot

The PCA components are plotted to visualize the clustering results

```
113 # Function to plot PCA components
114 def plot_pca_components(images_pca, labels, title):
115     plt.figure(figsize=(8, 6))
116     scatter = plt.scatter(images_pca[:, 0], images_pca[:, 1], c=labels, cmap='viridis',
117                           alpha=0.5)
117     plt.title(title)
118     plt.xlabel('PCA Component 1')
119     plt.ylabel('PCA Component 2')
120     plt.colorbar(scatter, label='Cluster')
121     plt.show()
122
123 # Plot PCA components of training data
124 plot_pca_components(train_hog_features_pca, train_clusters, 'KMeans Clustering Results
125                    (Training Data)')
```

### Image Plot with Cluster Labels

A subset of test images is plotted with their predicted cluster labels.

```
126 # Function to plot images with cluster labels
127 def plot_images(images, labels, title, image_size=(64, 64)):
128     fig, axes = plt.subplots(2, 5, figsize=(12, 6))
129     fig.suptitle(title)
130     for i, ax in enumerate(axes.flat):
131         if i < len(images):
132             ax.imshow(images[i].reshape(image_size + (3,)))
133             ax.set_title(f'Cluster {labels[i]}')
134             ax.axis('off')
135     plt.show()
136
137 # Plot images with predicted clusters
138 plot_images(test_images, test_clusters, 'KMeans Clustering Results (Test Data)')
139
```

### 2D PCA Visualization

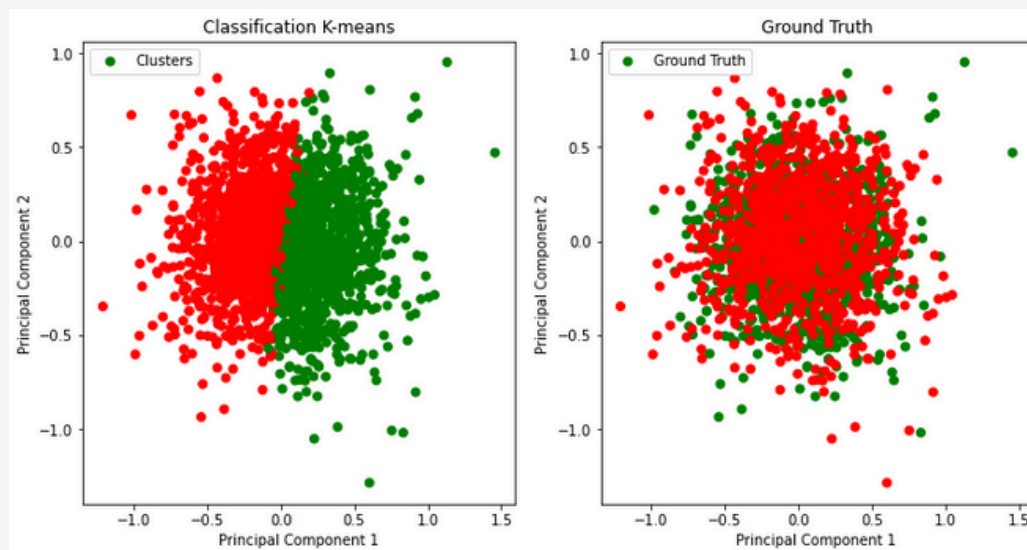
PCA is performed again to reduce the dimensionality to 2D for visualization of clustering results and ground truth labels.

```
140 # Perform PCA to reduce dimensionality to 2D for visualization
141 pca_2d = PCA(n_components=2)
142 pca_result = pca_2d.fit_transform(test_hog_features) # Use HOG features for PCA
143 visualization
144 # Visualize the clustering results
145 colormap = np.array(['green', 'red'])
146
147 plt.figure(figsize=(12, 6))
148
149 plt.subplot(1, 2, 1)
150 plt.title('Classification K-means')
151 plt.scatter(pca_result[:, 0], pca_result[:, 1], c=colormap[relabel_clusters],
152             label='Clusters')
153 plt.xlabel('Principal Component 1')
154 plt.ylabel('Principal Component 2')
155 plt.legend()
156
157 plt.subplot(1, 2, 2)
158 plt.title('Ground Truth')
159 plt.scatter(pca_result[:, 0], pca_result[:, 1], c=colormap[test_labels], label='Ground
160             Truth')
161 plt.xlabel('Principal Component 1')
162 plt.ylabel('Principal Component 2')
163 plt.legend()
164 plt.show()
165
```

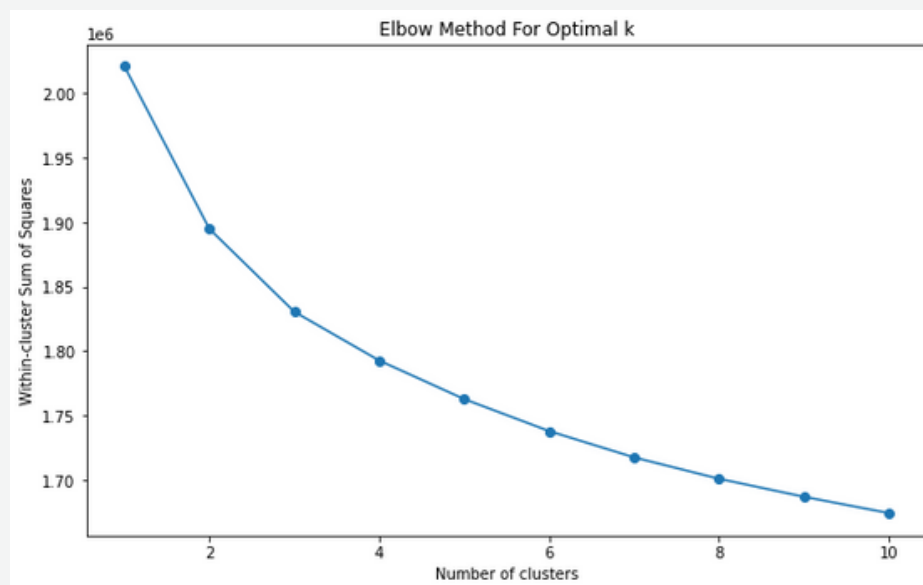


Figures are displayed in the Plots pane by default. To make them also appear inline in the console, you need to uncheck "Mute inline plotting" under the options menu of Plots.

## Classification KMeans and Ground Truth



## Elbow method for optimal K



## KMeans Clustering Results (Training Data)





## KMeans Clustering Results (Test Data)

KMeans Clustering Results (Test Data)



## Conclusion

The application of unsupervised learning techniques for image classification. The combination of HOG features, PCA, and KMeans clustering provides a method for grouping similar images without the need for labeled training data. The results, evaluated through confusion matrices and visualizations, show the effectiveness of this approach in distinguishing between images of dogs and cats. Future improvements could include experimenting with different feature extraction methods, clustering algorithms, and further hyperparameter tuning to enhance performance.

# Deep Learning

Deep learning is a subset of machine learning that uses multilayered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain..

## Data Preprocessing

### Image Loading and Preprocessing

Images are loaded from separate folders for training and testing datasets. Each image is resized to 64x64 pixels and normalized by dividing pixel values by 255.0. The labels are assigned as 0 for dogs and 1 for cats.

```
# Load training data
train_data = []
train_labels = []
train_base_path = 'C:/Users/t460/Downloads/archive/training_set/training_set'

for folder in os.listdir(train_base_path):
    folder_path = os.path.join(train_base_path, folder)
    if os.path.isdir(folder_path):
        for filename in os.listdir(folder_path):
            img_path = os.path.join(folder_path, filename)
            try:
                img = load_img(img_path, target_size=(img_height, img_width))
                img_array = img_to_array(img)
                train_data.append(img_array)
                train_labels.append(label[folder])
            except Exception as e:
                print(f"Error loading image {img_path}: {e}")
```

This section of the code focuses on preparing and normalizing the training data for a machine learning model. It converts the training data into a NumPy array and normalizes the pixel values by dividing by 255.0, which scales the values to a range between 0 and 1. This is a common preprocessing step for image data to ensure that the model trains more effectively. Additionally, the labels for the training data are also converted into a NumPy array. The code then proceeds to load the test data and labels from the specified directory path, preparing them for evaluation purposes.

```
29
30 x_train = np.array(train_data) / 255.0 # Normalize pixel values
31 y_train = np.array(train_labels)
32
33 # Load test data
34 test_data = []
35 test_labels = []
36 test_base_path = 'C:/Users/t460/Downloads/archive/test_set/test_set'
37
```

This section of the code is responsible for loading and preprocessing the test data for the machine learning model. It iterates through each folder in the specified test data directory. For each folder, it checks if the path is a directory and then iterates through each image file within that directory. Each image is loaded and resized to the specified dimensions (`img\_height` and `img\_width`). The image is then converted into an array and appended to the `test\_data` list. Correspondingly, the label for the image, determined by the folder name, is appended to the `test\_labels` list. If there is an error loading an image, it prints an error message. Finally, the test data and labels are converted into NumPy arrays, and the pixel values of the test data are normalized by dividing by 255.0, ensuring they are in the range of 0 to 1.

```
37
38 for folder in os.listdir(test_base_path):
39     folder_path = os.path.join(test_base_path, folder)
40     if os.path.isdir(folder_path):
41         for filename in os.listdir(folder_path):
42             img_path = os.path.join(folder_path, filename)
43             try:
44                 img = load_img(img_path, target_size=(img_height, img_width))
45                 img_array = img_to_array(img)
46                 test_data.append(img_array)
47                 test_labels.append(folder)
48             except Exception as e:
49                 print(f"Error Loading image {img_path}: {e}")
50
51 x_test = np.array(test_data) / 255.0 # Normalize pixel values
52 y_test = np.array(test_labels)
53
```

This section of the code defines, trains, and evaluates a Convolutional Neural Network (CNN) model for image classification:

### 1. Model Definition:

- The model is built using convolutional (Conv2D) and pooling (MaxPooling2D) layers to extract features from images.
- A flattening layer (Flatten) is used to convert the 2D matrix data to a vector, followed by dense (Dense) layers for classification.

### 2. Model Compilation:

- The model is compiled with the `sparse_categorical_crossentropy` loss function and `adam` optimizer.

### 3. Model Training:

- The model is trained on the training data for 10 epochs with a batch size of 32, using 20% of the data for validation.

### 4. Model Evaluation:

- The model's accuracy is evaluated on the test data, and the test accuracy is printed.

```
53 # Define the model
54 model = Sequential()
55 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)))
56 model.add(MaxPooling2D((2, 2)))
57 model.add(Conv2D(64, (3, 3), activation='relu'))
58 model.add(MaxPooling2D((2, 2)))
59 model.add(Conv2D(128, (3, 3), activation='relu'))
60 model.add(MaxPooling2D((2, 2)))
61 model.add(Flatten())
62 model.add(Dense(128, activation='relu'))
63 model.add(Dense(num_classes, activation='softmax'))
64 model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
65 # Print model summary
66 model.summary()
67 # Train the model
68 model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
69 # Evaluate the model
70 test_loss, test_acc = model.evaluate(x_test, y_test)
71 print(f"Test accuracy: {test_acc}")
```

The accuracy

The screenshot shows a Jupyter Notebook environment. The left pane displays the code for loading training and test data. The right pane shows the Variable Explorer and the Console output.

**Variable Explorer:**

Name	Type	Size	Value
train_labels	list	...	[0, 0, 0, 0, 0, 0, 0, 0, 0, ...]
x_test	Array of float32	(2023, 128, 128, 3)	[[[0.04313726 0.02745098 0. ... [0.5137255 0.44313726 0. ...
x_train	Array of float32	(8005, 128, 128, 3)	[[[0.15686275 0.1764706 0. ... [0.16078432 0.1764706 0. ...
y_test	Array of int32	(2023,)	[0 0 0 ... 1 1 1]
y_train	Array of int32	(8005,)	[0 0 0 ... 1 1 1]

**Console Output:**

```
Epoch 10/10
201/201 ----- 102s 508ms/step - accuracy: 0.9867 - loss: 0.0418 -
val_accuracy: 0.6889 - val_loss: 1.5034
64/64 ----- 8s 119ms/step - accuracy: 0.8236 - loss: 0.7841
Test accuracy: 0.7760751247406006
```



# Our Team

**shahd waleed**

**Bassant Tamer**

**Salma Osama**

**Fayza Mahmoud**