Shape recognition

Functions

Image feature functions used:

- eccentricity: Eccentricity is a measure of how elongated an object or shape is. It quantifies the shape of an object by calculating the ratio of the distances between the foci and the major axis length. A value close to 0 indicates a more circular shape, while a value close to 1 indicates a more elongated shape.
- area_convex: Area convex represents the total area of the convex hull of an object. The convex hull is the smallest convex polygon that contains all the points of the object.
- perimeter: Perimeter refers to the total length of the boundary or contour of an object. It measures the distance around the object's boundary.
- solidity: Solidity is the ratio of the object's area to the area of its convex hull. It provides a measure of how solid or compact the shape is. A value of 1 indicates a completely solid shape, while a value less than 1 indicates holes or concavities in the shape.
- euler_number: Euler number is a topological property that characterizes the connectivity and number of holes in an object. It is calculated based on the number of objects, holes, and tunnels present in the image.
- area_bbox: Area bbox represents the area of the bounding box enclosing the object. The bounding box is the smallest rectangle that fully contains the object.
- equivalent_diameter_area: Equivalent diameter is a measure of the characteristic size or diameter of the object. It is the diameter of a circle with the same area as the object.
- orientation: Orientation refers to the angle at which the major axis of the object is tilted. It provides information about the object's rotation with respect to a reference axis.
- solidity (mentioned twice): This seems to be a duplicate mention in the code. It is likely a typo or an oversight.

These image feature functions are used to extract various properties or characteristics of objects or regions within the images. They capture information related to shape, size, compactness, and orientation, which can be useful for image analysis, classification, and other computer vision tasks.

The Leave-One-Out (LOO) cross-validator is a technique used for model evaluation and performance estimation. It is a special case of k-fold cross-validation, where k is set to the total number of samples in the dataset.

Let’s explain shortly how It works. For each data point in the dataset, the LOO cross-validator creates a training set containing all the data points except the one being evaluated (left out), model is trained on the training set. The left-out data point is used as the test set to evaluate the trained model. First and secound step are repeated for each data point in the dataset, resulting in as many iterations as the number of samples in the dataset. The performance of the model is evaluated by aggregating the results from each iteration, such as computing the average accuracy or other metrics.

Advantages of LOO cross-validation:

- It provides an unbiased estimate of model performance because it uses all available data points for testing.
- It maximizes the training data used for model fitting, which can lead to better model performance.

Disadvantages of LOO cross-validation:

- It can be computationally expensive, especially for large datasets, as it requires training and testing the model for each data point.
- It can be sensitive to outliers since each sample is individually left out, and the model may overfit or underfit on specific outliers.

LOO cross-validation is commonly used when the dataset size is relatively small, and obtaining an accurate estimate of model performance is crucial. It helps to assess the generalization capability of the model by simulating the scenario of testing on unseen data.

The image features and labels are split into training and test sets using train_test_split from the sklearn.model_selection module. The test set size is set to 20% of the data, and the random state is fixed for reproducibility.

A loop is executed from 1 to 9 to train and evaluate a K-Nearest Neighbors classifier for each number of neighbors. The classifier is created using KNeighborsClassifier from the sklearn.neighbors module. It is trained on the training data (X_train and y_train), and predictions are made on the test data (X_test).

The display_random_image function is defined to display a random image from the test set along with the predicted class label. It takes a dataset parameter and uses random selection and indexing to choose an image. It displays the image using plt.imshow from the matplotlib library and predicts the class label using the trained classifier.

Libraries

os: Provides a way to use operating system dependent functionality, such as accessing file paths and directories.

numpy (np): A library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of mathematical functions.

PIL (Image): The Python Imaging Library adds image processing capabilities to your Python interpreter.

sklearn.model_selection: Provides functions for splitting data into training and test sets and generating cross-validation folds.

sklearn.neighbors: Provides functionality for k-nearest neighbors classification.

sklearn.preprocessing: Provides tools for scaling and preprocessing the data.

sklearn.metrics: Provides functions for evaluating model performance, such as accuracy score.

imghdr: Provides functions for determining the type of image file based on its content.

skimage.measure (regionprops): Provides functions for measuring properties of labeled image regions.

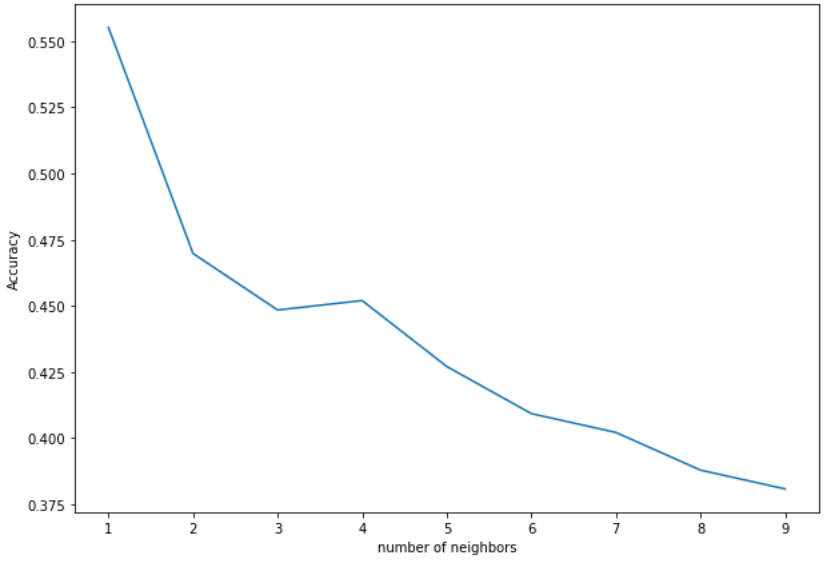
random: Provides functions for generating pseudo-random numbers.

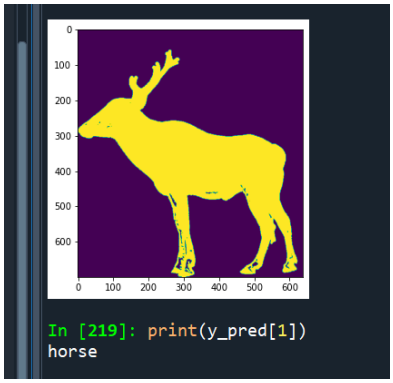
time: Provides various time-related functions.

matplotlib.pyplot (plt): A collection of command style functions that make Matplotlib work like MATLAB.

Results:

We have already listed 9 image features that we used and with them we achieved a precision result of 0.5551601423487544, we started with combinations of 2 image extractors (e.g. Eccentricty and area convex gave us a result of 0.341637), the more features we added, the result with repaired more. We also trained the data on different numbers of neighbors in the range from 1 to 9. The results are visible in the table.






```
Accuracy: 0.5551601423487544 , for  1  Neigbors.
Accuracy: 0.4697508896797153 , for  2  Neigbors.
Accuracy: 0.4483985765124555 , for  3  Neigbors.
Accuracy: 0.45195729537366547 , for  4  Neigbors.
Accuracy: 0.42704626334519574 , for  5  Neigbors.
Accuracy: 0.4092526690391459 , for  6  Neigbors.
Accuracy: 0.40213523131672596 , for  7  Neigbors.
Accuracy: 0.3879003558718861 , for  8  Neigbors.
Accuracy: 0.3807829181494662 , for  9  Neigbors.
```