lab1

5/4/24

Write a Python program to import and Export data using Pandas library functions

Importing

```
import pandas as pd
df = pd.read_csv("/content/austinHousingData.csv")
df.head()
```

Exporting

```
url = "archive.ics.uci.edu/ml/machine-learning-
                                    databases"
colnames = ("sepal-length", "sepal-width", "petal
            length", "petal-width", "class")
iris_data = pd.read_csv(url, names=colnames)
iris_data.head()
iris_data.to_csv("exported.csv")
```

2) Data preprocessing

i) Import dataset using pandas

ii) perform dataset shape & analysis

iii) use is_null() function from pandas to analyse missing values.

iv) Drop or fill missing values according to your use case example dropna() and fill().

v) we can generate dummy variables. (is a binary variable that indicates whether a separate categorical variable that indicates a specific values)

12/4/24 Lab 2:

Decision Tree Algorithm:

- Create a Root node for the tree.
- If all examples are positive, return the Single Node tree Root, with label = +.
- If all Examples are negative, return the Single-node tree Root, with label = -
- If Attributes is empty, Return the Single node tree Root, with label = most common value of Target_attribute in Examples.
- otherwise Begin
  - A ← the attribute from attribute that ~~best~~ * classifies Examples
  - The decision attribute for Root ← A
  - For each possible value, v_i of A.
    - Add a new branch below Root, corresponding to the test $A = v_i$.
    - let examples v_i, be the Subset of Examples that have value v_i for A.
    - ~~If~~ If examples v_i, is empty
      - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples.

- Else below this new branch add the subtree ID3 (Examples v_i, Target attribute, Attributes $-\{A\}$).

- End
- Return Root

* The best attribute is the one with highest information gain.

output:

Entropy of the entire dataset : 0.94

① Highest information gain = outlook
= 0.246

② highest information gain = rainy = 0.971
But attribute is windy.

Lab3:

KNN

Algorithm:

1) Input: unlabled instance x_{test} to which the class table needs to be predicted.
2) calculate Euclidean distance.
3) Find nearest neighbour: select K instance with the smallest distance to x_{test}
4) For classification:
   - Majority vote: If its classification test, count the occurance of each class table among the K-nearest neighbour.
   - Assign table: Assign the class table with the highest count of the predicted table for x_{test}.

OUTPUT:

```
model.predict([[7.7, 2.6, 6.9, 2.3]])
- array(['Iris Virginica'], dtype='<U14')
```

Linear regression:

Algorithm:

Training phase:

1) Input: Training data with one predictor (x) & target variable (y)

2) compute means

3) Estimate coefficients:

$$\beta_1 = \frac{cov(x,y)}{var(x)}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

Prediction phase:

1) Input: new x_{test}

2) compute prediction: use $y_{pred} = \beta_0 + \beta_1 x_{test}$ to predict y_{pred}.

OUTPUT:

model.predict = ([[2]])

- Array ([107.82727115])

logistic regression:

Algorithm:

Training Phase:

1) Initialize parameter: Satart with random or zero volumes for weights (w) & bias (b).

2) compute predictions: calculate predictions using the equation

$\hat{y} = a(\theta^T x + b)$, where a is the signed function.

3) compute loss: Measure the error between predictions & actual taste using binary cross-entropy loss.

4) update parameter: adjust weights & bias using gradient descent to initialize.

5) repeat: Iterate step 2-4 until convergence or a max. no of iterations

~~Prediction Ph~~

~~Output~~

8/5/24

Week-4

Support Vector Machine:

Algorithm

1) Define Kernel functions
   Ex: $K(x_1, x_2) = x_1 \cdot x_2$
2) Solve the quadratic programming problem to find the α value.
3) Complete weight and bias.
4) Identify the support vectors.
5) Make predictions.

output:

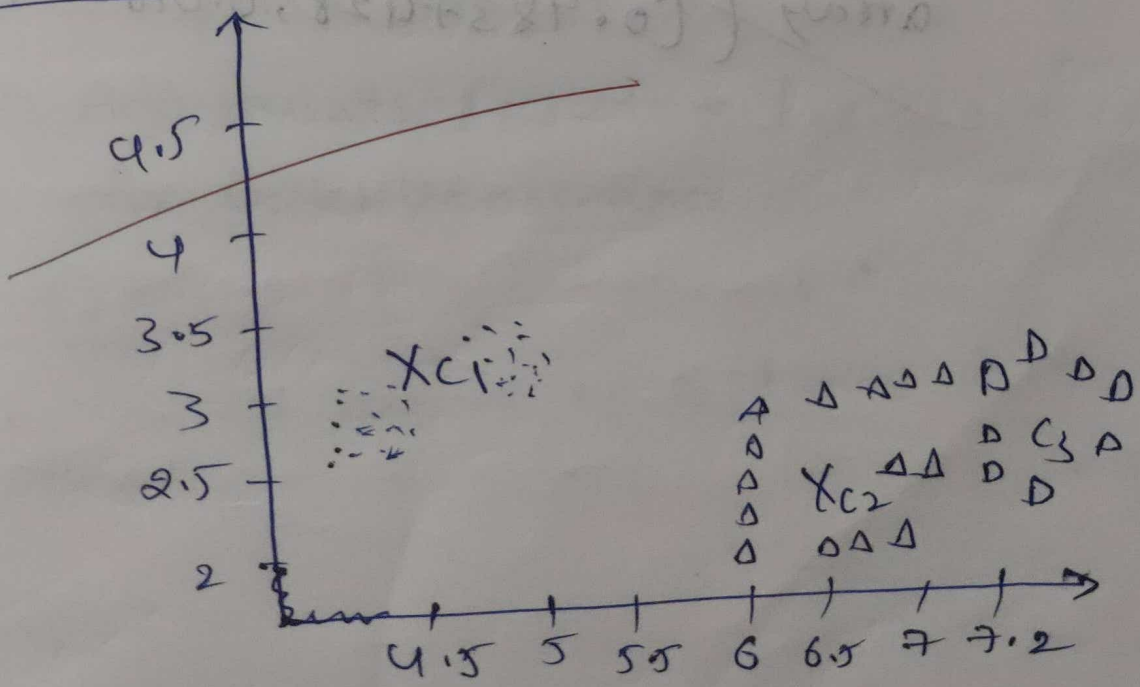
→ Model = svm(x)
model.fit(x_train, y_train)
Predictions = Model.predict(x_test)
accuracy(y_test, Predictions)
0.962300

Model.predict([[-0.47069, -0.1604 ...
0.19695]]) array(0)

K Means clustering Algorithm:

1. Select the number K to decide the no. of clusters.
2. Select random K points or centroids.
3. Assign each point to the closest centroid, which will form the predefined K cluster.
4. calculate the variance and place a new centroid of each cluster.
5. Repeat step 3, reassign the centroid.
6. If any re assignment go to step-4 else go to finish.
7. Model is Ready.

output:



3) principal component Analysis

Algorithm

1) Calculate Mean.
2) calculation of covariance Matrix.
3) Eigen values of covariance Matrix.
4) computation of the eigen vector - unit eigen vectors.
5) computation of first principal components.
6) Geometric meaning of first principal components.

Output:

pca -

pca.explained_variance_ratio.

array ([0.9837428, 0.01620498]).

Lab 5:

ANN with Backpropogation:

Algorithm:

1) Initialization: Randomly set weights and biases.
2) Forward pass: compute output using current weights and biases.
3) Calculate Error: Find the difference between predicted and actual output.
4) Backward pass: Update weights and biases to minimize error using gradient descent.
5) Repeat: Iterate steps 2-4 for multiple epochs.
6) Evaluation: Assess performance on a separate dataset.
7) Adjustments: Fine-tune hyperparameters for better performance.

Output:

MLP correct answers while testing: 8/9
(Accuracy = 0.88)

Random forest:

Algorithm:

1) Initialization:
   - choose number of trees and random feature subsets.

2) Tree construction:
   - For each tree:
     - Randomly select data with replacement and features.
     - Grow tree recursively with random feature selection at each split.

3) Prediction:
   - Aggregate predictions from all trees.

4) Optional:
   - Calculate feature importance.
   - Fine-tune hyperparameters.

5) Evaluation:
   - Assess performance on validation data.

output:

accuracy = 0.93

Adaboost Algorithm:

1) Initialization:
   - Initialize weights for each training sample uniformly
   - choose the base learner (ex: decisionTree)

2) for each iteration t:
   - Train the base learner on the training data with current sample weights.
   - compute the weighted error of the base learner.
   - compute the weight of the base learner in the final ensemble.

3) update sample weights:
   - Increase weights of misclassified samples.
   - Decrease weights of correctly classified sample

4) Normalize sample weights.

5) Repeat 2-4 for a fixed number of iterations or untill a stopping criterion is met.

6) Final prediction:

- combine predictions of all weak learners using their respective weights.

7) Evaluation:

- Assess the performance of the AdaBoost ensemble on a separate validation dataset.

Output:

Accuracy = 0.944

ESD
31/5/24