Indian Institute of Technology Dharwad

CS214: Artificial Intelligence Laboratory

Lab 5: K-Nearest Neighbors, Bayes, and Naive Bayes Classifier

You are given the Wisconsin Diagnostic Breast Cancer (WDBC) dataset. The dataset features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe the characteristics of the cell nuclei present in the image. A few of the images can be found at http://www.cs.wisc.edu/ street/images/. The separating plane (separating plane in the 3-D space of Worst Area, Worst Smoothness and Mean Texture) was obtained using Multisurface Method-Tree (MSM-T) [1], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes. The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in [2].

The dataset details

- Number of attributes: 31 (Diagnosis, 30 real-valued input features)
- Attribute information
  1. Diagnosis (1 = malignant, 0 = benign) (Attribute 1)
  2. Ten real-valued features are computed for each cell nucleus: (Attribute 2-31)
     (a) radius (mean of distances from center to points on the perimeter)
     (b) texture (standard deviation of gray-scale values)
     (c) perimeter
     (d) area
     (e) smoothness (local variation in radius lengths)
     (f) compactness ($perimeter^2/area - 1.0$)
     (g) concavity (severity of concave portions of the contour)
     (h) concave points (number of concave portions of the contour)
     (i) symmetry
     (j) fractal dimension ("coastline approximation" - 1)

  The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, Attribute 2 is Mean Radius, Attribute 12 is Radius SE, Attribute 22 is Worst Radius.

We are providing you with the prepared datasets (train, validation, and test datasets) in the csv files.

1. Original Dataset
   - train data: ***WDBC_Train.csv***
   - validation data: ***WDBC_Validation.csv***
   - test data: ***WDBC_Test.csv***

2. PCA transformed dataset for two components ($l = 2$)
    - train data: ***WDBC_PCA2_Train.csv***
    - validation data: ***WDBC_PCA2_Validation.csv***
    - test data: ***WDBC_PCA2_Test.csv***
3. PCA transformed dataset for two components ($l = 10$)
    - train data: ***WDBC_PCA10_Train.csv***
    - validation data: ***WDBC_PCA10_Validation.csv***
    - test data: ***WDBC_PCA10_Test.csv***

Write a Python program to complete the following tasks.

Part A: K-Nearest Neighbors (KNN) classification on original data

Perform the KNN classification and performance metric evaluation as listed below.

1. Load train, validation, and test data from ***WDBC_Train.csv***, ***WDBC_Validation.csv***, and ***WDBC_Test.csv***, respectively.
2. Implement KNN classification with $K = 7$.
3. Evaluate and compare the performance metrics (listed below) on validation data and test data.
    - Confusion matrix
    - Accuracy, Precision, Recall, and F1-score

Part B: Implementation of Bayes classifier (Gaussian Distribution) on original data

The Bayes Classifier assumes that features follow a Gaussian distribution and applies Bayes' Theorem for classification.

1. Load train, validation, and test data from ***WDBC_Train.csv***, ***WDBC_Validation.csv***, and ***WDBC_Test.csv***, respectively.
2. Prepare the input feature vectors and target/class vectors for train, validation, and test data. The input features vectors contain the records/tuples without the class label attribute and the target/class vectors contain the class label attribute.
3. Separate the training features vectors into two classes (Benign = 0, Malignant = 1).
4. Compute the mean vector and covariance matrix for each class (exclude the class labeled attributes i.e., 'diagnosis'). Also, compute the prior probability for each class, $P(C_i) = \frac{\text{\# samples in class i}}{\text{Total training samples}}$ where, C_i represents the i^{th} class.

5. Now, for each tuple/record in the test data (exclude the class labeled attributes i.e., 'diagnosis'), compute the likelihood for each class. The likelihood can be found using the mean vector and covariance matrix computed on the training dataset for each class. (**Use *multivariate_normal* from *scipy* and get the probability density function (pdf)**)

6. Next, compute posterior probability: $P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{P(\mathbf{x})}$. Here, $P(C_i|\mathbf{x})$: Posterior Probability, $p(\mathbf{x}|C_i)$: Likelyhood, $P(C_i)$: Prior, and $P(\mathbf{x})$: Total Probability.

7. Finally, for each tuple/record in the validation and test data, assign a class labeled with respect to the highest posterior probability.

8. Now that you have the actual and predicted class labels for all the validation and test data, evaluate the following performance metrics for both.

   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part C: Implementation of Naive Bayes classifier (Gaussian Distribution) on original data

The Naive Bayes Classifier assumes that features follow a Gaussian distribution and all features are independent.

1. Load train, validation, and test data from ***WDBC_Train.csv***, ***WDBC_Validation.csv***, and ***WDBC_Test.csv***, respectively.

2. Prepare the input feature vectors and target/class vectors for train, validation, and test data. The input features vectors contain the records/tuples without the class label attribute and the target/class vectors contain the class label attribute.

3. Now, to implement the Naive Bayes classifier, use *GaussianNB()* from *Scikit-learn*. The parameter estimation or model training should be performed using the training features vectors and target/class vectors (use the *fit* function from *GaussianNB()*).

4. Now use the *predict* function from *GaussianNB()* to predict the class levels of validation and test data.

5. Now that you have the actual and predicted class labels for all the validation and test data, evaluate the following performance metrics for both.

   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part D: K-Nearest Neighbors (KNN) classification on standardized data

Perform the KNN classification and performance metric evaluation as listed below.

1. Load train, validation, and test data from ***WDBC_Scaled_Train.csv***, ***WDBC_Scaled_-Validation.csv***, and ***WDBC_Scaled_Test.csv***, respectively.

2. Implement KNN classification with $K = 1$.
3. Evaluate and compare the performance metrics (listed below) on validation data and test data.
    - Confusion matrix
    - Accuracy, Precision, Recall, and F1-score

Part E: K-Nearest Neighbors (KNN) classification on PCA transformed ($l = 2$) data

Perform the KNN classification and performance metric evaluation as listed below.

1. Load train, validation, and test data from ***WDBC_PCA2_Train.csv***, ***WDBC_PCA2-_Validation.csv***, and ***WDBC_PCA2_Test.csv***, respectively.
2. Implement KNN classification with $K = 11$.
3. Evaluate and compare the performance metrics (listed below) on validation data and test data.
    - Confusion matrix
    - Accuracy, Precision, Recall, and F1-score

Part F: Implementation of Bayes classifier (Gaussian Distribution) on PCA transformed ($l = 2$) data

The Bayes Classifier assumes that features follow a Gaussian distribution and applies Bayes' Theorem for classification.

1. Load train, validation, and test data from ***WDBC_PCA2_Train.csv***, ***WDBC_PCA2-_Validation.csv***, and ***WDBC_PCA2_Test.csv***, respectively.
2. Prepare the input feature vectors and target/class vectors for train, validation, and test data. The input features vectors contain the records/tuples without the class label attribute and the target/class vectors contain the class label attribute.
3. Separate the training features vectors into two classes (Benign = 0, Malignant = 1).
4. Compute the mean vector and covariance matrix for each class (exclude the class labeled attributes i.e., 'diagnosis'). Also, compute the prior probability for each class, $P(C_i) = \frac{\text{\# samples in class i}}{\text{Total training samples}}$ where, C_i represents the i^{th} class.
5. Now, for each tuple/record in the test data (exclude the class labeled attributes i.e., 'diagnosis'), compute the likelihood for each class. The likelihood can be found using the mean vector and covariance matrix computed on the training dataset for each class. (**Use *multivariate_normal* from *scipy* and get the probability density function (pdf)**)
6. Next, compute posterior probability: $P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{P(\mathbf{x})}$. Here, $P(C_i|\mathbf{x})$: Posterior Probability, $p(\mathbf{x}|C_i)$: Likelyhood, $P(C_i)$: Prior, and $P(\mathbf{x})$: Total Probability.

7. Finally, for each tuple/record in the validation and test data, assign a class labeled with respect to the highest posterior probability.

8. Now that you have the actual and predicted class labels for all the validation and test data, evaluate the following performance metrics for both.

    - Confusion matrix
    - Accuracy, Precision, Recall, and F1-score

Part G: Implementation of Naive Bayes classifier (Gaussian Distribution) on PCA transformed ($l = 2$) data

The Naive Bayes Classifier assumes that features follow a Gaussian distribution and all features are independent.

1. Load train, validation, and test data from ***WDBC_PCA2_Train.csv***, ***WDBC_PCA2-_Validation.csv***, and ***WDBC_PCA2_Test.csv***, respectively.

2. Prepare the input feature vectors and target/class vectors for train, validation, and test data. The input features vectors contain the records/tuples without the class label attribute and the target/class vectors contain the class label attribute.

3. Now, to implement the Naive Bayes classifier, use *GaussianNB()* from *Scikit-learn*. The parameter estimation or model training should be performed using the training features vectors and target/class vectors (use the *fit* function from *GaussianNB()*).

4. Now use the *predict* function from *GaussianNB()* to predict the class levels of validation and test data.

5. Now that you have the actual and predicted class labels for all the validation and test data, evaluate the following performance metrics for both.

    - Confusion matrix
    - Accuracy, Precision, Recall, and F1-score

Part H: K-Nearest Neighbors (KNN) classification on PCA transformed ($l = 10$) data

Perform the KNN classification and performance metric evaluation as listed below.

1. Load train, validation, and test data from ***WDBC_PCA10_Train.csv***, ***WDBC_PCA-10_Validation.csv***, and ***WDBC_PCA10_Test.csv***, respectively.

2. Implement KNN classification with $K = 7$.

3. Evaluate and compare the performance metrics (listed below) on validation and test data.

    - Confusion matrix
    - Accuracy, Precision, Recall, and F1-score

Part I: Implementation of Bayes classifier (Gaussian Distribution) on PCA transformed ($l = 10$)

The Bayes Classifier assumes that features follow a Gaussian distribution and applies Bayes' Theorem for classification.

1. Load train, validation, and test data from ***WDBC_PCA10_Train.csv***, ***WDBC_PCA--10_Validation.csv***, and ***WDBC_PCA10_Test.csv***, respectively.

2. Prepare the input feature vectors and target/class vectors for train, validation, and test data. The input features vectors contain the records/tuples without the class label attribute and the target/class vectors contain the class label attribute.

3. Separate the training features vectors into two classes (Benign = 0, Malignant = 1).

4. Compute the mean vector and covariance matrix for each class (exclude the class labeled attributes i.e., 'diagnosis'). Also, compute the prior probability for each class, $P(C_i) = \frac{\text{\# samples in class i}}{\text{Total training samples}}$ where, C_i represents the i^{th} class.

5. Now, for each tuple/record in the test data (exclude the class labeled attributes i.e., 'diagnosis'), compute the likelihood for each class. The likelihood can be found using the mean vector and covariance matrix computed on the training dataset for each class. (**Use *multivariate_normal* from *scipy* and get the probability density function (pdf)**)

6. Next, compute posterior probability: $P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{P(\mathbf{x})}$. Here, $P(C_i|\mathbf{x})$: Posterior Probability, $p(\mathbf{x}|C_i)$: Likelyhood, $P(C_i)$: Prior, and $P(\mathbf{x})$: Total Probability.

7. Finally, for each tuple/record in the validation and test data, assign a class labeled with respect to the highest posterior probability.

8. Now that you have the actual and predicted class labels for all the validation and test data, evaluate the following performance metrics for both.

   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part J: Implementation of Naive Bayes classifier (Gaussian Distribution) on PCA transformed ($l = 10$) data

The Naive Bayes Classifier assumes that features follow a Gaussian distribution and all features are independent.

1. Load train, validation, and test data from ***WDBC_PCA10_Train.csv***, ***WDBC_PCA--10_Validation.csv***, and ***WDBC_PCA10_Test.csv***, respectively.

2. Prepare the input feature vectors and target/class vectors for train, validation, and test data. The input features vectors contain the records/tuples without the class label attribute and the target/class vectors contain the class label attribute.

3. Now, to implement the Naive Bayes classifier, use *GaussianNB()* from *Scikit-learn*. The parameter estimation or model training should be performed using the training features vectors and target/class vectors (use the *fit* function from *GaussianNB()*).

4. Now use the *predict* function from *GaussianNB()* to predict the class levels of validation and test data.
5. Now that you have the actual and predicted class labels for all the validation and test data, evaluate the following performance metrics for both.
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part K: Comparision and Summarization of accuracy for all datasets and classification algorithms on validation and test data.

In the end, summarize the accuracy in the form of table. The sample code is also shown below.

```
# Define the table data
data = {
    "Dataset": ["Original", "Standardized", "PCA (l=2)", "PCA (l=10)"],
    "KNN": [100, 100, 100, 100],
    "Bayes Classifier": [100, -, 100, 100],
    "Naive Bayes Classifier": [100, -, 100, 100]
}

# Create DataFrame
df = pd.DataFrame(data)
df.set_index("Dataset", inplace=True)

# Print DataFrame
print(f"Test Accuracy Comparison\n{df}")
```

Note

1. Create a separate Jupyter Notebook for each part.
2. For likelihood computation, use the below function from *scipy* library.

   ```
   from scipy.stats import multivariate_normal
   likelihood = multivariate_normal(mean=mean_vectors, cov=covariance_matrix).pdf(X)
   ```

3. For Naive Bayes Classifier, use the below code from *scikit-learn*

   ```
   from sklearn.naive_bayes import GaussianNB
   naiveBayes = GaussianNB()
   naiveBayes.fit(X_train, y_train)
   y_test_pred = naiveBayes.predict(X_test)
   ```

4. Refer to the slide uploaded on Moodle about the performance evaluation (the confusion matrix, accuracy, etc.). You can import *confusion_matrix*, *accuracy_score*, *precision_score*, *recall_score*, *f1_score*, and *classification_report* from *sklearn.metrics* library.

Please upload the completed Jupyter Notebook. Copy all Jupyter Notebook and data files in a folder and rename the folder as ***your_rollnumber_Lab5***, then create the zip file with the name ***your_rollnumber_Lab5.zip***. Finally, upload the zip file on the Moodle for evaluation.

References

[1] K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992

[2] K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34