Indian Institute of Technology Dharwad

CS214: Artificial Intelligence Laboratory

Lab 6: Logistic Regression and Support Vector Machine

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. Standardized Dataset
   - train data: ***WDBC_Scaled_Train.csv***
   - validation data: ***WDBC_Scaled_Validation.csv***
   - test data: ***WDBC_Scaled_Test.csv***
3. PCA transformed dataset for ten 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: Logistic Regression on original data

Perform the logistic regression 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 Logistic Regression classification with `sklearn.linear_model.LogisticRegression`
3. Analyze the importance of features by observing the absolute values of the learned weight vector $\mathbf{w}$, Visualize the weight distribution by plotting a bar graph.
4. Evaluate and compare the performance metrics (listed below) on validation data and test data.
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part B: Support Vector Machine with a Linear Kernel on Original Data

1. Load train, validation, and test data from ***WDBC_Train.csv***, ***WDBC_Validation.csv***, and ***WDBC_Test.csv***, respectively.
2. Perform the classification using the Support Vector Machine (SVM) method with a linear kernel. Train the model on the training data using an in-built SVM function, `sklearn.svm.SVC(kernel='linear')`, and fine-tune the regularization parameter C using cross-validation. After selecting the optimal C, use the trained model to predict the class labels for the validation and test data. Finally,you have the actual and predicted class labels for both the validation and test data, generate a weight plot and evaluate the following performance metrics for both:
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part C: Support Vector Machine with a Gaussian (RBF) Kernel on Original Data

1. Load train, validation, and test data from ***WDBC_Train.csv***, ***WDBC_Validation.csv***, and ***WDBC_Test.csv***, respectively.
2. Perform the classification using the Support Vector Machine (SVM) method with a Gaussian (RBF) kernel. Train the model on the training data using an in-built SVM function, `sklearn.svm.SVC(kernel='rbf')`, and fine-tune both the regularization parameter C and the kernel parameter γ using cross-validation. After selecting the optimal values for C and γ, use the trained model to predict the class labels for the validation and test data. Finally, with the actual and predicted class labels for both the validation and test data, generate a weight plot (for a 2D dataset) and evaluate the following performance metrics for both:
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part D: Support Vector Machine with a Polynomial Kernel on Original Data

1. Load train, validation, and test data from ***WDBC_Train.csv***, ***WDBC_Validation.csv***, and ***WDBC_Test.csv***, respectively.
2. Perform the classification using the Support Vector Machine (SVM) method with a polynomial kernel. Train the model on the training data using an in-built SVM function, `sklearn.svm.SVC(kernel='poly')`, and fine-tune the regularization parameter C, the kernel parameter γ, and the polynomial degree degree using cross-validation. After selecting the optimal values for C, γ, and degree, use the trained model to predict the class labels for the validation and test data. Finally, with the actual and predicted class labels for both the validation and test data, generate a weight plot (for a 2D dataset) and evaluate the following performance metrics for both:
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part E: Logistic Regression on Scaled Data

Perform the logistic regression 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 Logistic Regression classification with `sklearn.linear_model.LogisticRegression`
3. Analyze the importance of features by observing the absolute values of the learned weight vector **w**, Visualize the weight distribution by plotting a bar graph.
4. Evaluate and compare the performance metrics (listed below) on validation data and test data.
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part F: Support Vector Machine with a Linear Kernel on Scaled Data

1. Load train, validation, and test data from ***WDBC_Scaled_Train.csv***, ***WDBC_Scaled_-Validation.csv***, and ***WDBC_Scaled_Test.csv***, respectively.
2. Perform the classification using the Support Vector Machine (SVM) method with a linear kernel. Train the model on the training data using an in-built SVM function, `sklearn.svm.SVC(kernel='linear')`, and fine-tune the regularization parameter C using cross-validation. After selecting the optimal C, use the trained model to predict the class labels for the validation and test data. Finally,you have the actual and predicted class labels for both the validation and test data, generate a weight plot and evaluate the following performance metrics for both:
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part G: Support Vector Machine with a Gaussian (RBF) Kernel on Scaled Data

1. Load train, validation, and test data from ***WDBC_Scaled_Train.csv***, ***WDBC_Scaled_-Validation.csv***, and ***WDBC_Scaled_Test.csv***, respectively.
2. Perform the classification using the Support Vector Machine (SVM) method with a Gaussian (RBF) kernel. Train the model on the training data using an in-built SVM function, `sklearn.svm.SVC(kernel='rbf')`, and fine-tune both the regularization parameter C and the kernel parameter γ using cross-validation. After selecting the optimal values for C and γ, use the trained model to predict the class labels for the validation and test data. Finally, with the actual and predicted class labels for both the validation and test data, generate a weight plot (for a 2D dataset) and evaluate the following performance metrics for both:
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part H: Support Vector Machine with a Polynomial Kernel on Scaled Data

1. Load train, validation, and test data from ***WDBC_Scaled_Train.csv***, ***WDBC_Scaled_-Validation.csv***, and ***WDBC_Scaled_Test.csv***, respectively.
2. Perform the classification using the Support Vector Machine (SVM) method with a polynomial kernel. Train the model on the training data using an in-built SVM function, `sklearn.svm.SVC(kernel='poly')`, and fine-tune the regularization parameter C, the kernel parameter γ, and the polynomial degree degree using cross-validation. After selecting the optimal values for C, γ, and degree, use the trained model to predict the class labels for the validation and test data. Finally, with the actual and predicted class labels for both the validation and test data, generate a weight plot (for a 2D dataset) and evaluate the following performance metrics for both:

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

Part I: Logistic Regression on on PCA transformed ($l = 10$) Data

Perform the logistic regression classification and performance metric evaluation as listed below.

1. Load train, validation, and test data from ***WDBC_PCA10_Train.csv***, ***WDBC_PCA10--_Validation.csv***, and ***WDBC_PCA110_Test.csv***, respectively.
2. Implement Logistic Regression classification with `sklearn.linear_model.LogisticRegression`
3. Analyze the importance of features by observing the absolute values of the learned weight vector $\mathbf{w}$, Visualize the weight distribution by plotting a bar graph.
4. Evaluate and compare the performance metrics (listed below) on validation data and test data.
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part J: Support Vector Machine with a Linear Kernel on PCA Transformed ($l = 10$) Data

Perform the logistic regression classification and performance metric evaluation as listed below.

1. Load train, validation, and test data from ***WDBC_PCA10_Train.csv***, ***WDBC_PCA10--_Validation.csv***, and ***WDBC_PCA110_Test.csv***, respectively.
2. Perform the classification using the Support Vector Machine (SVM) method with a linear kernel. Train the model on the training data using an in-built SVM function, `sklearn.svm.SVC(kernel='linear')`, and fine-tune the regularization parameter C using cross-validation. After selecting the optimal C, use the trained model to predict the class labels for the validation and test data. Finally,you have the actual and predicted class labels for both the validation and test data, generate a weight plot and evaluate the following performance metrics for both:
   - Confusion matrix
   - Accuracy, Precision, Recall, and F1-score

Part K: Support Vector Machine with a Gaussian Kernel on PCA Transformed ($l = 10$) Data

Perform the logistic regression classification and performance metric evaluation as listed below.

1. Load train, validation, and test data from ***WDBC_PCA10_Train.csv***, ***WDBC_PCA10--_Validation.csv***, and ***WDBC_PCA110_Test.csv***, respectively.
2. Perform the classification using the Support Vector Machine (SVM) method with a Gaussian (RBF) kernel. Train the model on the training data using an in-built SVM function, `sklearn.svm.SVC(kernel='rbf')`, and fine-tune both the regularization parameter C and the kernel parameter γ using cross-validation. After selecting the optimal values for C and γ, use the trained model to predict the class labels for the validation and test data. Finally, with the actual and predicted class labels for both the validation and test data, generate a weight plot (for a 2D dataset) and evaluate the following performance metrics for both:
    - Confusion matrix
    - Accuracy, Precision, Recall, and F1-score

Part L: Support Vector Machine with a Polynomial Kernel on PCA Transformed ($l = 10$) Data

Perform the logistic regression classification and performance metric evaluation as listed below.

1. Load train, validation, and test data from ***WDBC_PCA10_Train.csv***, ***WDBC_PCA10--_Validation.csv***, and ***WDBC_PCA110_Test.csv***, respectively.
2. Perform the classification using the Support Vector Machine (SVM) method with a polynomial kernel. Train the model on the training data using an in-built SVM function, `sklearn.svm.SVC(kernel='poly')`, and fine-tune the regularization parameter C, the kernel parameter γ, and the polynomial degree degree using cross-validation. After selecting the optimal values for C, γ, and degree, use the trained model to predict the class labels for the validation and test data. Finally, with the actual and predicted class labels for both the validation and test data, generate a weight plot (for a 2D dataset) and evaluate the following performance metrics for both:
    - Confusion matrix
    - Accuracy, Precision, Recall, and F1-score

Part M: 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=10)"],
    "Logisistic Regression": [100, 100, 100],
    "SVM Linear": [100, -, 100, 100],
    "SVM Gaussian": [100, -, 100, 100]
    "SVM Polynomal": [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 Logistic Regression, use the below code from *scikit-learn*:

```
    from sklearn.linear_model import LogisticRegression
    logistic_reg = LogisticRegression()
    logistic_reg.fit(X_train, y_train)
    y_test_pred = logistic_reg.predict(X_test)
```

3. For Linear SVM, use the below code from *scikit-learn*:

```
    from sklearn.svm import SVC
    linear_svm = SVC(kernel='linear', C=1.0)  # You can adjust the C parameter here
    linear_svm.fit(X_train, y_train)
    y_test_pred = linear_svm.predict(X_test)
```

4. For Gaussian Kernel SVM (RBF Kernel), use the below code from *scikit-learn*:

```
    from sklearn.svm import SVC
    #You can adjust C and gamma parameters here
    rbf_svm = SVC(kernel='rbf', C=1.0, gamma='scale')
    rbf_svm.fit(X_train, y_train)
    y_test_pred = rbf_svm.predict(X_test)
```

5. For Polynomial Kernel SVM, use the below code from *scikit-learn*:

```
    from sklearn.svm import SVC
    #Adjust C, gamma, and degree parameters here
    poly_svm = SVC(kernel='poly', degree=3, C=1.0, gamma='scale')
    poly_svm.fit(X_train, y_train)
    y_test_pred = poly_svm.predict(X_test)
```

6. 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