Introduction to Machine Learning and Data Science

The Academic Center Ruppin

Final Assignment

Authors:

- **Matan Shemesh**
- **Ariel Cohen**

Table of Contents:

Part 1: Full Training of QDA Classifier

Introduction

In this task, we trained a QDA (Quadratic Discriminant Analysis) classifier to recognize sign language from images. We utilized solutions from our course's homework notebooks and implemented additional functions as needed. Each image was represented as a square matrix, and we reshaped it to the appropriate size for the code. We aimed for efficiency, writing short and efficient code to save runtime and resources. We added comments and explanations for complex calculations, and where we used ChatGPT, we noted it as a comment and added it to the appendix.

Question 2

Question 3 - Section B

After receiving the training series, we arranged the data according to the required format and size. We then wrote the estimation function and calculated the model parameters, creating an additional function that accepts all model parameters and returns the labeling for each image.

Calculations

During the calculation, for each LABEL, we calculated the appropriate model values - sigma, inverse sigma, determinant, and the appropriate Gaussian for use in the estimation calculation (taking the maximum value which represents the correct classification). We chose to use the expression we proved in question 2 because it is more convenient to work with in Python, as many of the functions are already available in the numpy library. This allowed us to access each parameter in the calculation individually and input these values into the Gaussian calculation.

Important Note

We added a comparison in the code between the values we calculated (mu_ml, pi_ml, sigma_ml) and those calculated by the sklearn API. This is because the API performs additional operations beyond those we performed, such as data sorting and other optimizations not covered in this course.

Question 3 - Section C

Source: Average: Source: Average:

The image generated from the vector of averages we calculated is less clear than the original, as it is an average of many examples classified the same. The classifier learned the shape of the sign on average, and as we expected, the image quality varies based on the number and quality of appearances in that series. We can see that the original training series was of higher quality, but it is important to note that we did not print all examples from the training series and may have found images of lower than average quality.

Question 4

We achieved 100% accuracy in the training series, meaning all examples were correctly classified. However, it is important to note that in real-world examples, we would not want such perfect classification to avoid overfitting.

Confusion Matrix

When looking at the confusion matrix (24x24), we see values only on the main diagonal, meaning all examples were correctly classified without misclassifications. The overall size of each labeled group is almost equal.

Precision and Recall

Precision: 1.0 - No cases where the model incorrectly predicted a label. Recall: 1.0 - Successfully identified all positive examples, indicating classification success.

Question 5

We ran the same process on the test series, using our generic functions, and obtained the corresponding results. The test images are very similar to each other, meaning that a classification that is unclear and very similar to another label will likely affect subsequent classifications.

We achieved 92.4% accuracy in the test series, indicating the classifier works but not perfectly. The confusion matrix shows that not only the main diagonal has values, meaning some examples were misclassified. Some mistakes are repeated (e.g., label 2 was often confused with label 6), aligning with our earlier observations of unclear images.

Precision: 1.0 - There were misclassifications. Recall: 1.0 - Successfully identified most positive examples, indicating classification success. Some categories had lower recall, indicating fewer examples in the test series.

Question 6

We applied the same process to the message file, successfully reading and interpreting the message in the code.

Part 2: Training and Performance Evaluation of Classifier

Introduction

We read about various semantic analysis models on the Huggingface website to understand and choose the appropriate approach. Semantic analysis applies natural language processing (NLP) technologies to interpret text similarly to humans.

In the preprocessing stage, sentiment analysis identifies keywords to highlight the main message of the text. Tokenization breaks a sentence into elements or tokens, while lemmatization converts words to their root forms. Stop words filter out insignificant words (e.g., 'the', 'of'). Sentiment scores assign emotional value to keywords, providing a numerical representation of the sentiment.

Cross Validation

We used the cross_validate function from sklearn.model_selection to evaluate model performance on various metrics and return accuracy percentages for the test and training series. This function automatically splits the data into training and test sets based on specified parameters, enabling us to understand and generalize different models studied in the course.

Using the cv parameter, we could define the strategy to maximize results, remembering that too many training iterations could lead to overfitting. The shuffle parameter ensures the data is mixed before splitting, and random_state=42 ensures the same random actions occur in each run, providing consistent results. We chose scoring=accuracy for the model evaluation.

We created an array containing all classifiers, running each classifier separately with the best values found. Each test weighed the computation speed (runtime) against accuracy. For example, if classifier X had a slightly higher accuracy but much longer runtime than classifier Y, we would prefer Y in most cases.

Decision Tree Classifier

We tested the max_leaf_nodes parameter, saving the maximum and average values for each fold and finally setting the main function to maximize accuracy for the test series.

Random Forest Classifier

We tested three parameters (n_estimators, max_depth, min_samples_split) separately, saving the highest accuracy values for the test series.

GaussianNB

We tested the var_smoothing parameter to influence the minimal variance of each feature, saving the maximum average accuracy for each test series.

KNeighborsClassifier

We tested parameters (n_neighbors, leaf_size, n_jobs) separately, saving the maximum average accuracy for each fold. We found that n_neighbors had a significant impact due to many examples directly affecting accuracy.

LinearDiscriminantAnalysis

We tested the solver parameter (changed to 'lsqr') and shrinkage (changed to 'auto'), achieving improved performance.

QuadraticDiscriminantAnalysis

Compared to other models, QDA in sklearn has fewer adjustable parameters. Despite constructing and implementing it ourselves in class, the improvements were minor.

Graphical Display

We created a bar graph to visualize accuracy for each classifier in the training and test series.

Classifier	Train Accuracy Before (%)	Test Accuracy Before (%)	Train Accuracy After (%)	Test Accuracy After (%)
KNeighborsClassifier	80.903	70.302	75.470	73.434
QuadraticDiscriminantAnalysis	89.758	74.068	89.758	74.068
DecisionTreeClassifier	100.000	63.940	79.592	68.689
RandomForestClassifier	100.000	76.014	97.463	76.015
LinearDiscriminantAnalysis	81.274	80.316	81.479	80.487
GaussianNB	71.171	70.739	71.168	70.757

Conclusions

In the tests for each classifier, we detailed the changes made to maximize results, reflected in the tables and graphs. The improvements were significant in most classifiers, demonstrating the effectiveness of our approach.

If asked to choose the best classifier for our semantic analysis model with the given data, we would choose the LDA classifier, as its test series accuracy was maximized. Additionally, the differences between the test and training series accuracies decreased in almost all classifiers, indicating reduced overfitting.