Clara Kang
Rachel Yu
BE 175
Mar 21, 2024

BE 175 Final Report

Introduction/Motivation

Our final project based on the paper, “Sign-to-speech translation using machine learning assisted stretchable sensor arrays”, focuses on implementing a machine learning algorithm to aid in the translation of sign language gesture recognition. Sign language plays a crucial role in bridging the conversational gap between those with and without speech impediments, but faces a major limitation in its breadth to only those who can understand it. In this way, the paper looks into ways to use wearable technology to read finger electrode signals during sign language hand gestures and translate them into audible speech, ultimately addressing the approachability of sign language. With the use of machine learning to assist these stretchable sensor arrays, we are able to shrink the gap of communication for disabilities and open the world to a whole new audience of storytellers and listeners, an important step in improving inclusivity in our world.

In terms of the specifics of the problem, the process from reading the signals to actually predicting the gesture from the data points entailed a complicated machine learning algorithm that we worked to implement according to the paper. In larger bioengineering applications, this similar approach with machine learning could help identify important points in data sets, classify them by relationship, and predict future interpretations for different biological models and systems, whether that be for understanding specific biological pathways for pharmaceutical advancement or designing biomedical devices to improve societal issues, as we have done here.

Methods

Our approach to solving this data analytics problem constituted 5 parts – signal collection, data preprocessing, dimensionality reduction, classification, and evaluation of model through testing.

For signal collection, the paper used a wearable glove that consisted of yarn-based stretchable sensor arrays (YSSAs) that read each finger movement into electrical signals and a wireless printed circuit board (PCB) that would integrate clear signal conditioning and transmit the signals to an external device for recording.

The next step, data preprocessing, necessitated dimensionality reduction and classification as there were too many data points and trials to run it through our machine learning algorithm. The original data file comprised 11 files, each specific for a separate gesture, ‘A’, ‘B’, ‘C’, ‘I’, ‘L’, ‘Y’, ‘1’, ‘2’, ‘3’, ‘8’, and ‘I love you’. Within each file were two days of multiple gesture collections, amounting to 263,215 data points in total and far exceeding the ability of our system to run. After graphing our data points and recognizing that the amplitudes of the peaks were what mattered as it indicated a complete gesture, we chose to manually reduce our sample size to

around 7 peaks (or periods of gestures) for each trial, adding up to 3000 data points from each trial and 6000 total for each gesture class. We then labeled the corresponding class labels (1-11) to each dataset to keep track of the relationship between the signals and the class it should predict to be. When we got to the step of normalizing and graphing our dataset, we actually found that although it is useful to scale features to a similar range, our experiment needed the clearly differing amplitudes to classify differences between gestures, a factor that was disrupted upon normalization and its consequent intensified noise. Going forward, we chose to skip the normalization step and move on to dimensionality reduction.

For dimensionality reduction, we chose to study two different methods, PCA (Principal Component Analysis) and LDA (Linear Discriminant Analysis) and its efficiency in tailoring our dataset to extract the features of interest for our SVM. Before performing any of the methods, we first split our data into training and testing sets to prevent overfitting, specifying that 20% of the data would be used to test and the other 80% would be trained.

For the first approach, PCA, we imported the sklearn.decomposition PCA package and applied dimensionality reduction using PCA(n_components=0.95), which would retain 95% of the variance in the data with the number of principal components. Next, fit_transform was used to fit PCA on the training data (X_train) and then transform it. Finally, the transform was applied to the testing data (X_test) to project it onto the same principal component from the training data. Furthermore, we graphed the scores plot and a loadings plot to identify the efficiency of PCA in observing patterns or separations between classes in the 2-dimensional space.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

For the second approach, LDA, we imported the sklearn.discriminant_analysis LDA package and followed a similar logic to PCA as above but including the y_train data as well in the initial step using fit_transform to fit the LDA on both X and Y training data sets and transform both of them. We also graphed the scores plot for LDA and the coefficients to better understand how LDA was organizing our data utilizing discriminant axes rather than principal components. Our final choice to use LDA came from our understanding that it was a supervised process, requiring data labeling, and its focus on finding feature subspace that maximized class separability, rather than just variance in the X data set as with PCA. Considering that the objective of our classification was to separate the X data by gesture class in a data set that had significant overlap between gestures, a consideration of the classes, y_train, proved to have better separation for our objective, which was further proved visually by the better separation in the scores plot for LDA compared to that of PCA.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA()
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)
```

Our next step, classification, used SVM (Support Vector Machines) to perform a One-vs-All strategy that classified our classes as binary, positive for the class being evaluated, and negative for all others. In order to implement this in our code, we imported the sklearn.svm SVC and sklearn.metrics accuracy_score and classification_report packages to first create an SVM classifier object with a specified kernel and ‘c’ parameter to control the regularization strength in terms of margin. We then trained the SVM classifier using X_train_lda, the transformed feature space attained after LDA was applied to our training data, as well as y_train, our corresponding labels, to separate the data points based on patterns existing in the training data set. Afterwards, we used the trained SVM classifier to make predictions for the X_test_lda and store the labels in y_pred. Lastly, we evaluated the classifier by printing a classification report that compared y_test with y_pred. We ran this SVM for all four kernel types – linear, sigmoid, polynomial, and rbf – and further evaluated with a confusion matrix to choose the best type to further evaluate and optimize in our last step, as seen in the figures below.

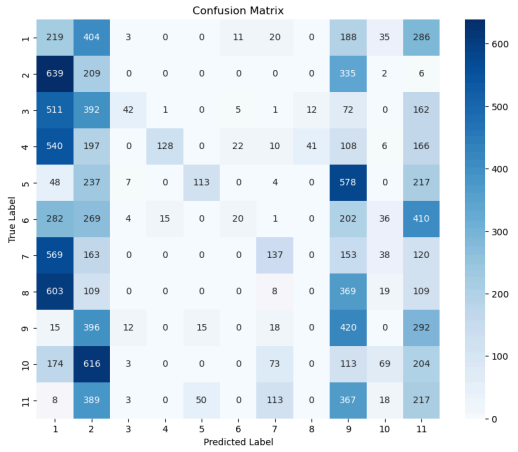

Figure A. Linear kernel confusion matrix

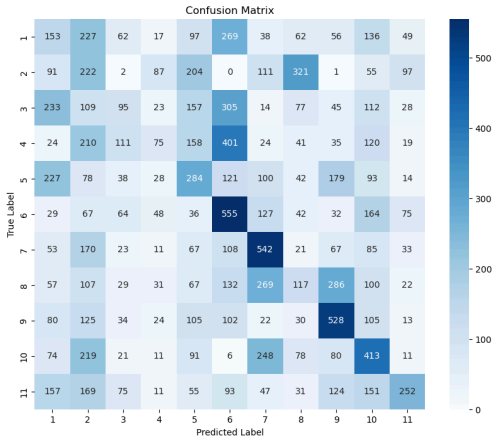

Figure B. Sigmoid kernel confusion matrix

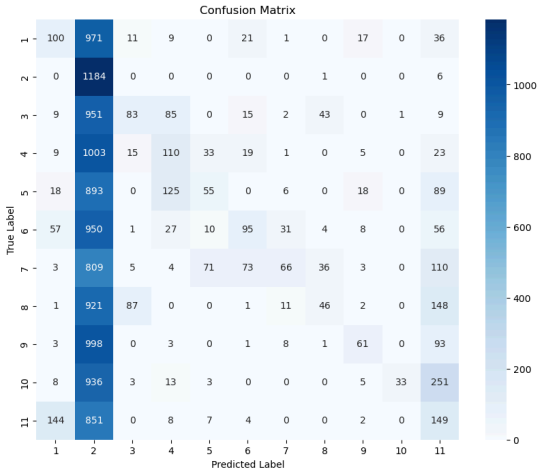

Figure C. Polynomial kernel confusion matrix

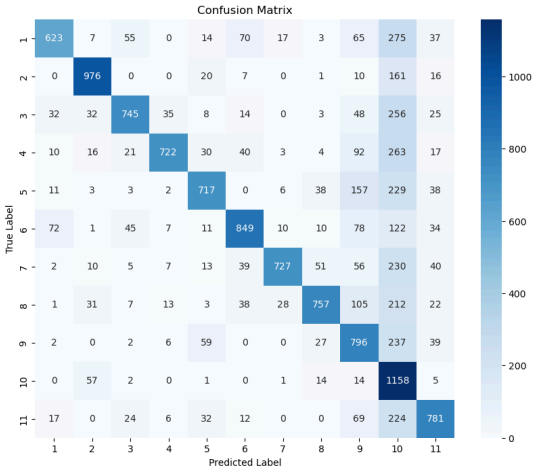

Figure D. RBF kernel confusion matrix

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import classification_report

svm_classifier = SVC(kernel='rbf', C=1)  svm_classifier.fit(X_train_lda,
y_train)
y_pred = svm_classifier.predict(X_test_lda)
print(classification_report(y_test, y_pred))
```

Lastly, we evaluated and optimized our model using K-fold cross validation as well as grid search cross validation using the sklearn.model_selection cross_val_score, GridSearchCV package. In order to use the GridSearchCV package, we first defined our parameter grid, including different ‘c’ and gamma (kernel coefficients for rbf) values. Using the package, we included the estimator parameter as the svm_classifier trained from above, the parameter grid defined before, the amount of folds for cv=5 to match the number of variables for each finger, and an accuracy metric to evaluate the optimal parameters. After fitting the Grid Search CV to the training data, the code ran until we found the highest accuracy of 87% to come with a ‘c’ value of 1000 and a gamma of 10. We also ran a k-fold cross validation for these parameters with the SVM and found the accuracy to be 86.3%, validating our approach in calculating the most accurate SVM model.

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10],}
svm_classifier = SVC(kernel='rbf')
grid_search = GridSearchCV(estimator=svm_classifier,
param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_lda, y_train)
```

Result

Finally, we graphed a final confusion matrix of our optimized SVM of 87% accuracy and found there to be a distinct dark blue diagonal line, indicating high success in our prediction versus the true labels of each gesture signal.

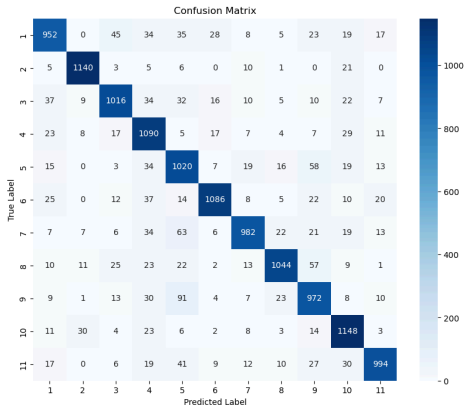

Figure D. Optimized RBF SVM confusion matrix

Compared to the results of the paper, we were able to achieve an accuracy similar to the paper's 98%. However, taking into account the fact that we were unable to use the whole data set considering the size and that we used a creative approach with LDA instead of PCA for dimensionality reduction, some next steps we can take from here are to find ways to include more data points while optimizing the efficiency of LDA, bringing us closer to the accuracy of the paper.