APPLIED MACHINE LEARNING

MULTI-CLASS TEXTURE CLASSIFICATION USING ERROR-CORRECTING OUTPUT CODES AND GAUSSIAN MIXTURE MODELS

Islam Ehab Anwar

Student ID: 201900337

Faculty of Science and Innovation

The Universities of Canada

December 2021

Contents

ABSTRACT

Multi-class texture classification is a common machine learning objective used to test supervised and unsupervised algorithms. This report covers the results of applying a supervised algorithm: Multi-class Error-Correcting Output Codes (ECOC), and an unsupervised algorithm: Gaussian Mixture Models (GMM). Data preparation and pre-processing methods are highlighted along with various feature engineering and dimensionality reduction to improve training performance. Furthermore, hyper-parameter optimization is conducted to identify optimal parameter values for each algorithm corresponding to the Describable Texture Data set, which is our data set of choice. Upon completing our testing; results, as well as the choice of hyper-parameters and dimensions, are discussed for both models.

Keywords Multi-class texture classification · Error-Correcting Output Codes · Gaussian Mixture Models · Describable Texture Dataset

1 Data Description & Analysis

In this section of the report, we discuss the motivation behind using the Describable Texture Dataset for our research. Moreover, extensive description is provided for the content of the data set, analysis of its structure as well as some visual examples of the various classes. We discuss the objective of this research paper and what machine learning techniques we will use to reach our objective.

1.1 Motivation to select the data set

The chosen data set is the Describable Texture Dataset. The motivation behind selecting this data set as opposed to other data sets is because it contains high variability in applying various classification algorithms. Furthermore, it created opportunities to prepare the data set as optimally as possible for the implementation of the classifiers.

1.2 Data Description

Regarding the structure of the data set, it consists of 5640 images – comprised of 47 key attributes or classes. The images were divided according to different pattern schemes such as bubbly, wrinkled, or zigzagged. The basic premise of our data is that each folder contains a class of images. Images consist of unique and identifiable texture patterns by which we can use to classify our data points. This can be enhanced further by using pixel values and other key features to generate more accurate classifications. It is worth mentioning that these predictions rely on the assumption that all images are set to gray-scale, therefore we are only operating through a single channel in place of 3 channels for RGB.

1.3 Data Visualization

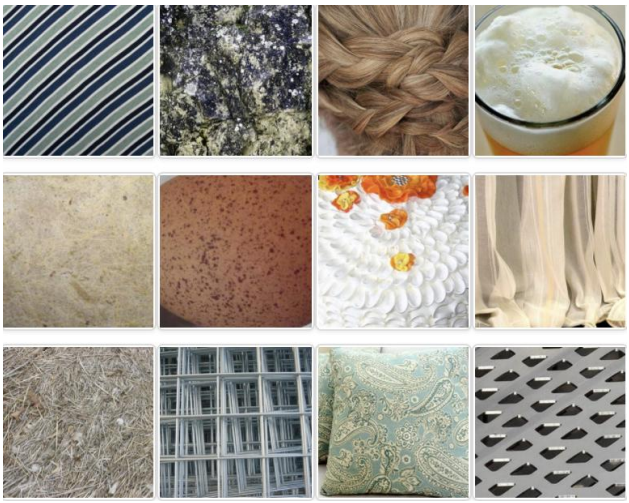

Figure 1: Sample Images - Describable Texture Dataset

1.4 Data Analysis

The data set is organized according to the type of texture it belongs to and the corresponding label is assigned to each image in a categorical form such as $banded/banded_0001.jpg$. Advisement was given to equally distribute the data set between the train, validation, and test set to produce consistent performance. However, we simply appended all the images in alphabetical order of their classes to an array of images and proceeded to manipulate the data from there.

1.5 Objective and Machine Learning Techniques

The primary objective of this paper is to achieve high accuracy for multi-class texture classification of each image using various machine learning techniques. This begins by importing the data set, then applying data preparation to configure the data as optimally as possible for training, and predictions. Furthermore, methods such as feature engineering, dimensionality reduction, as well as hyperparameter optimization are incorporated to fine-tune our parameters to achieve accurate models while accounting for potential overfitting. After applying the training data to our models, we will predict the unseen test data and review the results.

The two algorithms we will be testing on this data set are:
- Error-Correcting Output Codes
- Gaussian Mixture Models

2 Predictions

The contribution of this section of the report is the implementation of both of the above-mentioned algorithms: Error-Correcting Output Codes and Gaussian Mixture Models. Several techniques are mentioned and described throughout this section including data preparation, pre-processing as well as image processing techniques to attempt to differentiate between the labels, and lastly performance optimization via dimensionality reduction and hyper-parameter optimization methods. Lastly, analysis of results is provided by using various Python libraries such as matplotlib & scikit learn.

2.1 Data Preparation

The data set was first imported by using the python 'glob' library to search through the various folders and read them into NumPy arrays of images and their corresponding labels as categorical data types. width and height were specified during the data preparation phase at a value of 64x64 pixels to standardize image sizes across the dataset and compromise for both information loss & computation time. After importing, one hot encoding was used to fit and transform the categorical labels to integer labels classified from 0-46. Regarding data splitting, an 80-20 split was performed using the traditional train-test split. Moreover, stratified 4-fold cross-validation was used on the training set to train multiple validation folds.

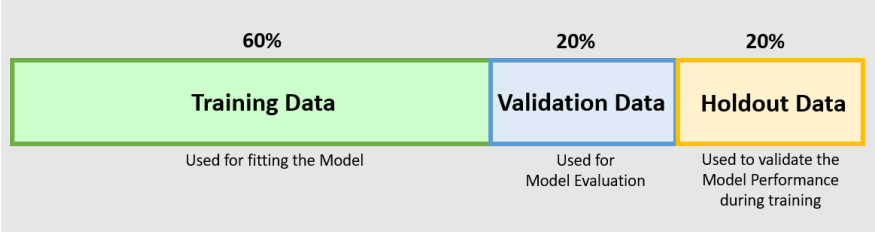

Figure 2: Data Split

2.2 Normalization & Feature Extraction

Normalization is an essential preprocessing step for image processing as it allows us to center the data and ensure all image pixels follow a similar data distribution.

$$X' = \frac{X}{255.0} \tag{1}$$

With regards to feature extraction, a data frame was created using the Pandas library to store the various features we generated from our images including:

- Pixel Values
- Mean
- Gabor Filters
- Gaussian Filters
- Median Filters
- Variance Filter

1. Pixel Values
Using our Pandas data frame, we iterated through every image in our dataset and extracted the pixel values to our data frame.

2. Mean
Computing the average local value of an images' pixel values using Python's NumPy library seemed like a sensible decision. This is because images in the same texture class may have similar pixel patterns and therefore, their means would not be too different value-wise.

$$Mean = \frac{\sum x_i}{x} \tag{2}$$

3. Gabor Filter
Gabor filters are linear filters used for texture analysis. It works by identifying specific frequencies or patterns in a certain direction, very similar to how the human visual system operates. It is defined as a sinusoidal wave multiplied by a Gaussian function. We defined a kernel size to insert a 2D portion of the image into the filter & created a filter bank of Gabor filters by using the Gabor kernel method in the OpenCV library and appended them to our data frame. Furthermore, the extensive number of adjustable hyperparameters make Gabor filters extremely versatile in determining different texture patterns for many types of images as shown by equation (3).

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) \cos(2\pi \frac{x'}{\lambda} + \psi) \tag{3}$$

Where:
$x' = x \cos\theta + y \sin\theta$
$y' = -x \sin\theta + y \cos\theta$
λ = Wavelength of sinusoidal wave
θ = Orientation of parallel stripes of filter
ψ = Phase offset
σ = Standard deviation of Gaussian function
γ = Spatial aspect ratio

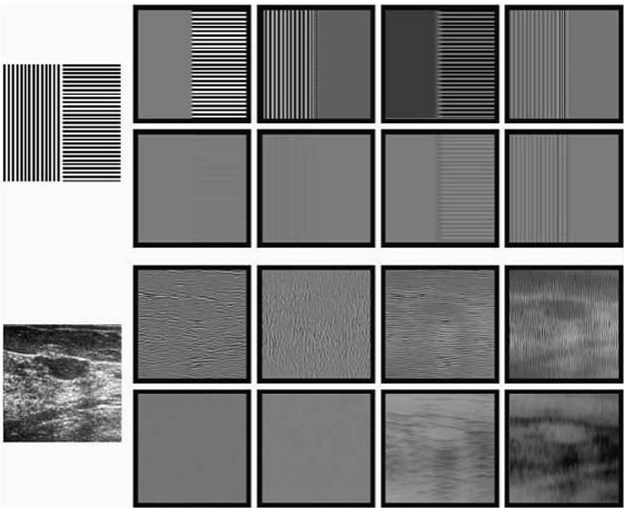

Figure 3: Gabor Filter Bank

4. Gaussian Filter
This is an operation that extracts a multidimensional Gaussian filter from an image and is implemented as a 1-dimensional convolution filter. We extracted two separate Gaussian filters of standard deviations of 3 and 7 to identify which of the two can capture more essence of the images. A linear filter such as this can be used to reduce noise in images and prepare them for better extraction of edge-detection features.

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{x^2}{2\sigma^2}) \tag{4}$$

Where:
σ = Standard Deviation

5. Median Filter
A median filter is a filter that receives an image and returns a multidimensional median filter of the same shape. This is especially useful to reduce random noise in an image.

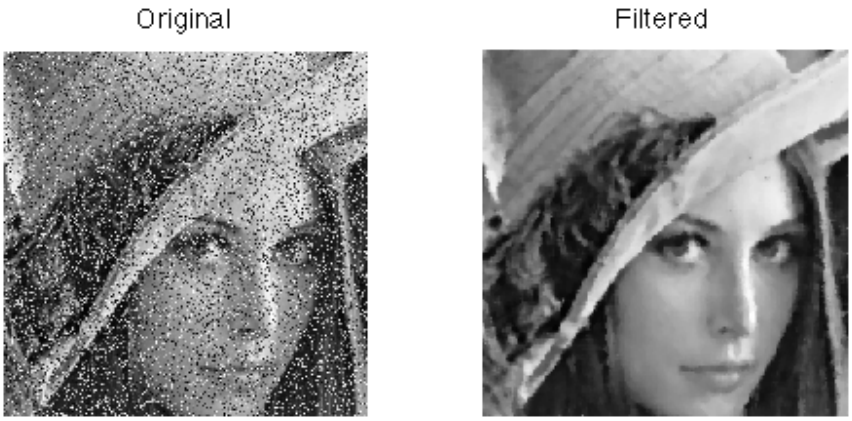

Figure 4: Median Filter

6. Variance Filter
Our final feature extractor is a generic filter located in the SciPy library that allows us to perform a mathematical filter of function onto an image. Our function of choice was the variance function. As was previously discussed with computing the mean, computing the variance helps us standardize the data, and extracting the images' variance can aid in identifying differences between various classifications according to this second-order statistic.

2.3 Dimensionality Reduction

Dimensionality Reduction involves the transformation of high-dimensional data to lower-dimensional data while preserving significant information or essence of the dataset. The chosen method of reduction is Linear Discriminant Analysis (LDA). LDA is pre-processing step that uses projections to maximize the component axes for class separation. This relies on minimizing the scatter in-between classes and maximizing the scatter between various classes. We will mention the impact of dimensionality reduction on both of our algorithms in the upcoming sections.

In-class scatter matrix:

$$S_W = \sum_{c=1} S_c \tag{5}$$

$$S_c = \sum_{i \in c} (x_i - \bar{x_c})(x_i - \bar{x_c})^T \tag{6}$$

Between-class scatter matrix:

$$S_B = \sum_{c} n_c(\bar{x_c} - \bar{x})(\bar{x_c} - \bar{x})^T \tag{7}$$

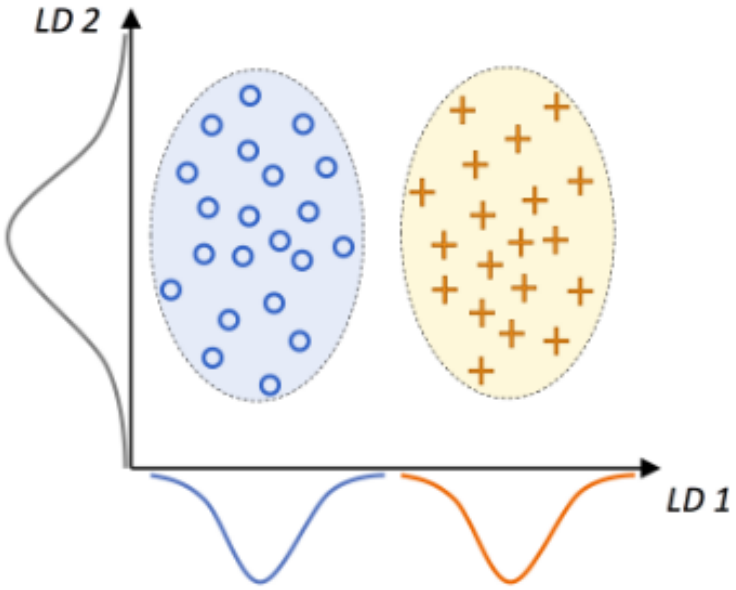

Figure 5: Dimensionality Reduction via LDA

2.4 Error-Correcting Output Codes

Error-correcting output codes (ECOC) are an ensemble learning framework to address multi-class classification problems Dietterich and Bakiri (1995). As opposed to One-vs-Rest and One-vs-One classifiers, the use of a code-based strategy separates class labels by assigning unique arrays of binary values (0s and 1s) in a large matrix of output codes (referred to as the codebook).

At a code size between 0-1, the algorithm requires fewer classifiers than the One-vs-Rest classifier. However, a value greater than 1 would use more classifiers and effectively correct codes that do not match their correct labels. By splitting the problem from a single multi-class task into multiple binary classification tasks, we can considerably improve model accuracy and any additional classifiers can contribute to correcting any erroneous labels.

Row	Column														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Figure 6: Error Correcting Output Codes

2.4.1 Model Optimization

In this subsection, we attempt to optimize our ECOC algorithm by discovering the best hyperparameter values. Error-correcting output codes use a binary classifier as a built-in estimator to fit and predict the results. We have opted for Random Forest as our binary classifier. The hyperparameters we are optimizing are:

- LDA - number of components, $[2, 5, 8, 11, 14, 16, 19, 22, 25, 28, 31, 35, 39, 44]$
- Random Forest - number of estimators in forest, $[50, 100, 125, 150, 200, 250]$
- Random Forest - loss criterion, $['gini', 'entropy']$
- Random Forest - max depth, $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
- ECOC - code size, $[2, 3, 4, 5, 6, 7, 8, 9, 10]$

1. LDA - Number of components:
Regarding dimensionality reduction, we needed to reduce the number of features after performing feature extraction from approximately 8100 features to a value where we could easily separate the classes. According to our testing and tuning, we concluded from Figure (7) that a value of 14 was an optimal value as that is where the effect of LDA plateaus performance-wise as well as to avoid overfitting the model.

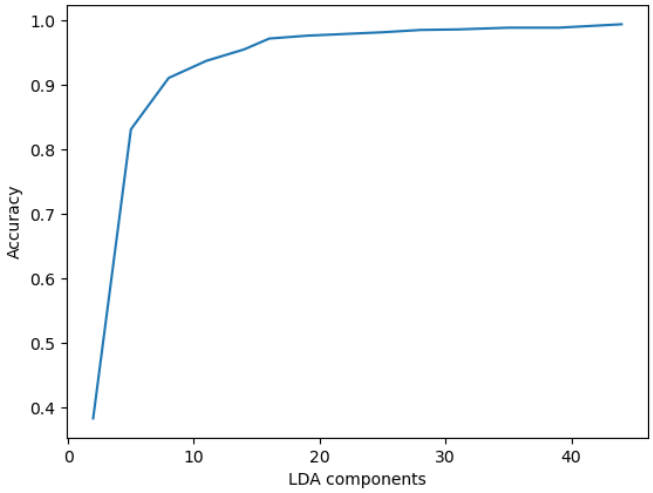

Figure 7: LDA Components - Accuracy

2. Random Forest - Number of estimators:
As for the number of trees in our random forest plotted in Figure (8), the results fluctuate repeatedly. Therefore, it is safe to assume a middle ground to avoid overfitting such as a value of 150.

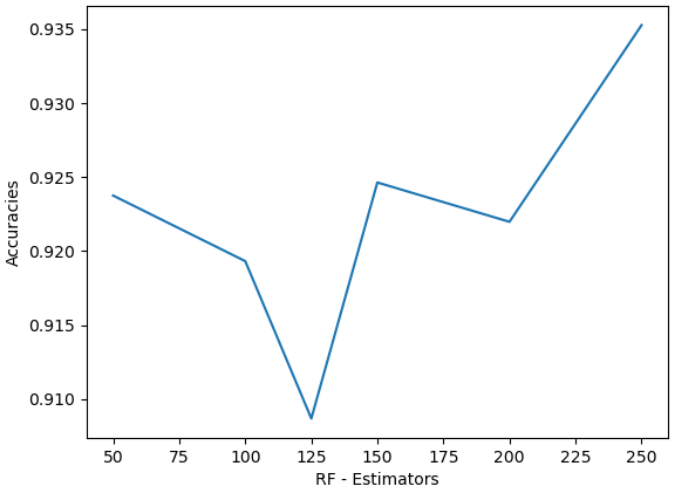

Figure 8: RF Estimators - Accuracy

3. Random Forest - Max depth & Loss Criterion:
Optimization was performed on the max depth parameter while using both criterion values: 'gini' and 'entropy'. The default criterion is set to 'gini'. However, observing Figures (9) & (10), we can conclude that the optimal max depth is 7 using the 'gini' criterion as it is a more gradual rise in performance as opposed to fluctuating results for 'entropy'.

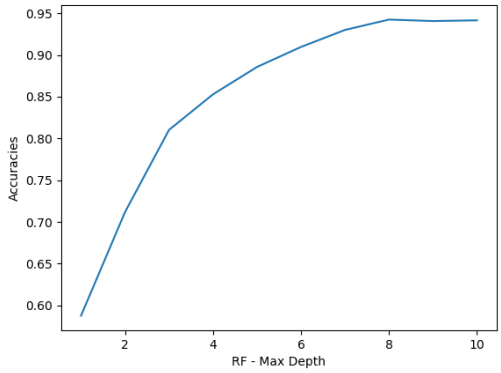

Figure 9: Depth - Accuracy 'gini'

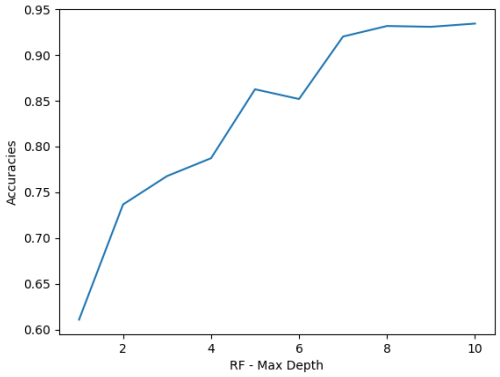

Figure 10: Depth - Accuracy 'entropy'

4. ECOC - Code size:
Our code size will be a percentage of our total number of classes and according to Figure (11), the results are very consistent as the algorithm is correcting the erroneous labels with the extra classifiers. Therefore, a value of 5 was chosen.

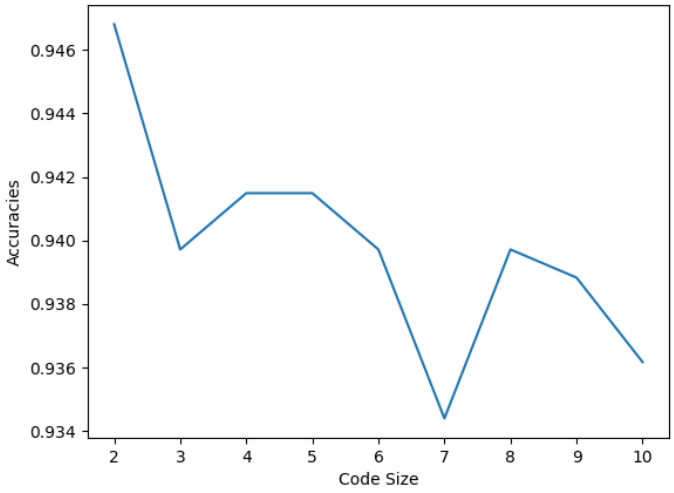

Figure 11: Code Size - Accuracy

2.4.2 Analysis of Results

Post hyperparameter optimization, we began fitting our training data into the ECOC model and obtained the following validation accuracies the final testing accuracy:

- Validation fold 1: 0.9867021276595744
- Validation fold 2: 0.9804964539007093
- Validation fold 3: 0.9867021276595744
- Validation fold 4: 0.9796099290780141
- Test accuracy: 0.016843971631205674

Based on the results above, it is clear that our model is overfitting for the training set. Even after applying dimensionality reduction to create meaningful decision boundaries for our classifiers, we still ended up with an overfit model.

2.5 Gaussian Mixture Models

A Gaussian Mixture is an unsupervised clustering algorithm that mitigates k-mean's hard clustering algorithm approach - it associates one point to only one cluster. It performs this by creating several Gaussian distributions all with the respective following properties:

- Each Gaussian has a unique μ.
- A covariance that defines it's width.

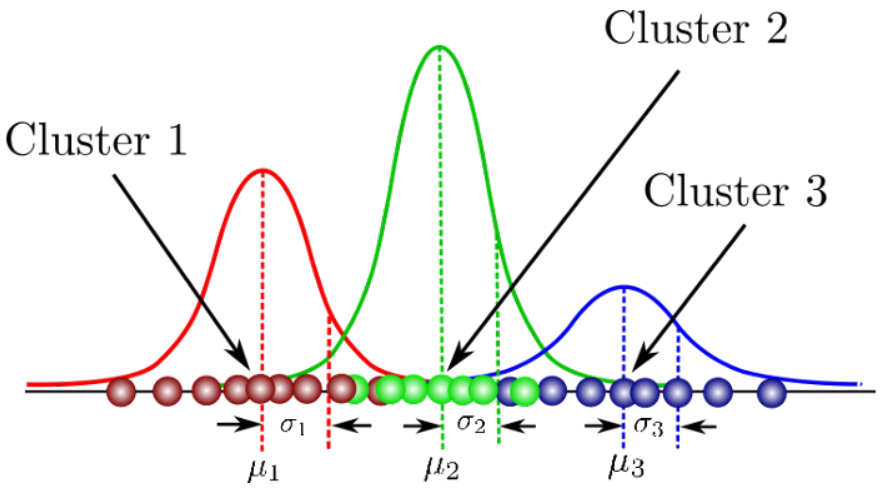

Figure 12: k=3 Gaussian Distributions

Accordingly, Gaussian Mixtures use the series of Gaussian distributions attempting to model the entire distribution to our dataset. Considering that our dataset is evenly distributed in terms of observations, we don't need to apply weights to each cluster based on the number of images per class. GMM implements the Expectation-Maximization (EM) algorithm for fitting the various distributions, as well as drawing ellipsoidal shapes for cases of multivariate models. Moreover, computing the Bayesian Information Criterion (BIC) can assess the number of clusters identified in the data.

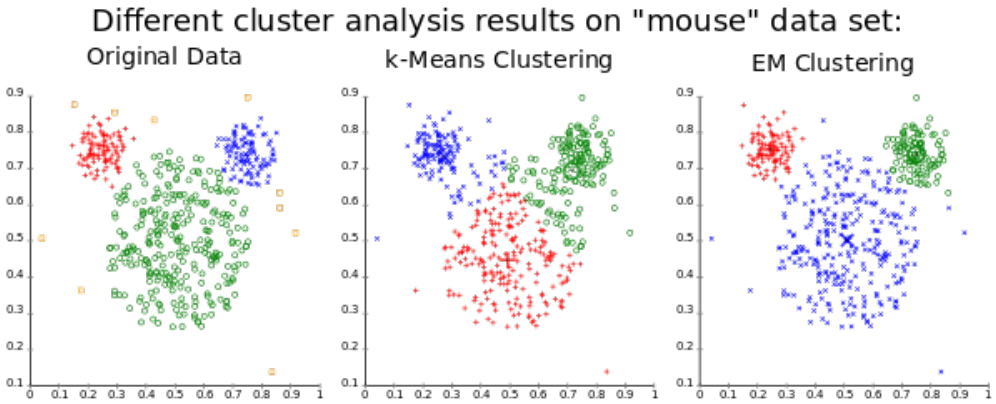

Figure 13: EM vs k-Means on sample data set

2.5.1 Model Optimization

This subsection covers the hyperparameter optimization for our Gaussian Mixture Model, where we address key hyperparameters and how their plots can aid us in deciding the best values to not overfit our model. The hyperparameters we are optimizing are:

- LDA - number of components, [2, 5, 8, 11, 14, 16, 19, 22, 25, 28, 31, 35, 39, 44]
- GMM - maximum iterations, [100, 125, 150, 175, 200]
- GMM - covariance type, ['full', 'tied', 'diag', 'spherical']
- GMM - initialization method, ['kmeans', 'random']

1. LDA - Number of Components:
According to Figure (14), it is evident that our optimal dimension for LDA is approximately 11 to avoid drastic overfitting.

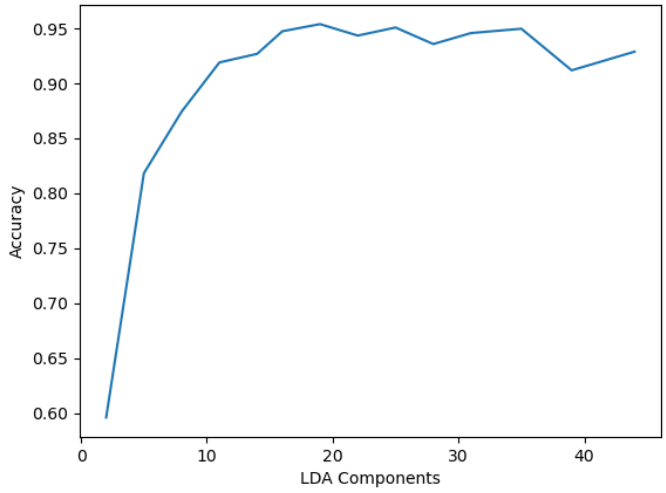

Figure 14: LDA Components - Accuracy (GMM)

2. GMM - Maximum Iterations:
The maximum iterations parameter is the number of times the EM algorithm is performed before converging at a cluster center. Given Figure (15), we can conclude that our optimal value is approximately 150.

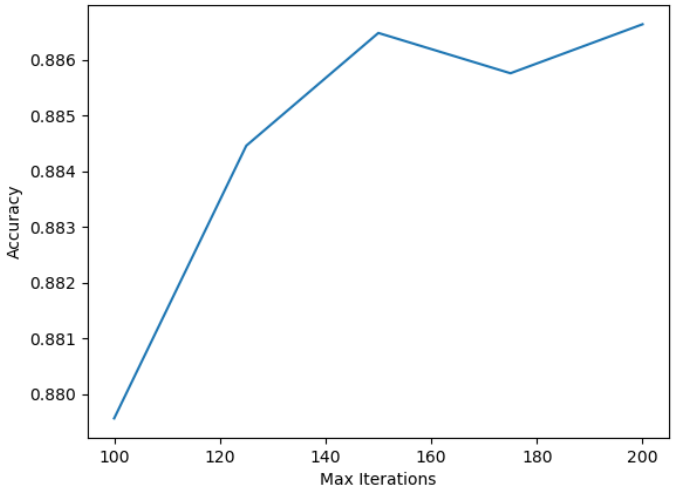

Figure 15: GMM Max Iterations - Accuracy

3. GMM - Covariance Type:
Our covariance type describes how each Gaussian distribution is fitted to our model by specifying if the components have their covariance matrix, or they collectively own a single covariance matrix. This can significantly impact the distribution of the clusters for our classification. Observing Figure (16), our best value is 'tied', even though it may decrease the accuracy but the percentage of accuracy of images to their respective clusters is higher than any other parameter value.

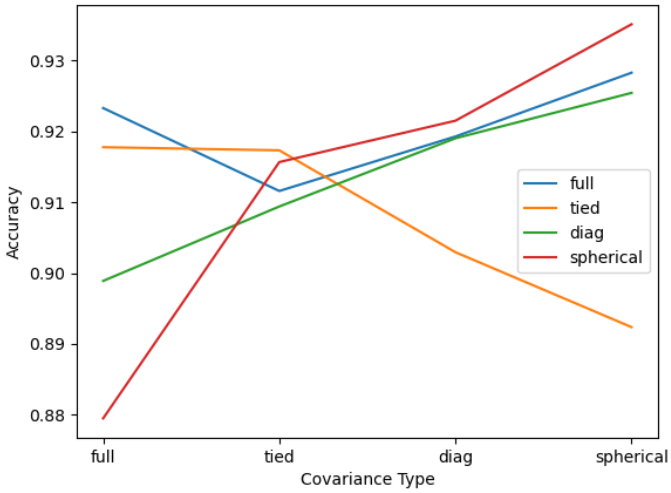

Figure 16: Covariance Type - Accuracy

4. GMM - Initialization Method:
The initial parameter is how the Gaussian Mixture chooses the cluster centers, either using k-means or random initialization. The results below show that k-means performs better, therefore, that is our optimal value for this hyperparameter.

K-means:

- Validation fold 1: 0.9418767978377576 , Rand accuracy: 0.8566308664977559
- Validation fold 2: 0.9363145966328633 , Rand accuracy: 0.852725671541216
- Validation fold 3: 0.9485918764936759 , Rand accuracy: 0.882784583498829
- Validation fold 4: 0.9485775154514876 , Rand accuracy: 0.8637294845181493

Random:

- Validation fold 1: 0.641650068096099 , Rand accuracy: 0.21345961694117024
- Validation fold 2: 0.6545149053462781 , Rand accuracy: 0.27564842658099625
- Validation fold 3: 0.6544901051401708, Rand accuracy: 0.20790051826986047
- Validation fold 4: 0.6857834870111611 , Rand accuracy: 0.253985891769832

2.5.2 Analysis of Results

After optimizing our Gaussian model, we ran the code and obtained the training and predicted accuracies:

- Validation fold 1: 0.9543129144555897, Rand accuracy: 0.8061548462088984
- Validation fold 2: 0.9599483891704208, Rand accuracy: 0.8503396899054784
- Validation fold 3: 0.9575824325578361, Rand accuracy: 0.83814097724073
- Validation fold 4: 0.9429254443857918, Rand accuracy: 0.7798564154340505
- Test accuracy: 0.6415510127218146, Rand accuracy: 0.1539267794819544

Interpreting these results, our training results show a positive pattern in identifying accurate image clusters as well as a strong rand score to identify which image belongs to a particular class. However, the unseen data accuracy is far lower, which is an expected result. This indicates that the model is overfitted to our training data.

3 Conclusion

The final contribution of this report is assessing the results of our Error-correcting output codes and Gaussian mixture model algorithms. Comparison of results is interpreted as well as a discussion of the tuned hyperparameters and how they contribute to the complexity of the problem.

3.1 Comparison of Results

The previous sections of this report gave an informative description of model optimization as well as an analysis of our final results. Using that obtained data, we can conclude that the Error-correcting output code model performed slightly better in the training data than the Gaussian mixture model, potentially due to deploying an unsupervised algorithm to a supervised machine learning problem. Furthermore, Gaussian mixtures have to not only determine the various clusters but also has to identify which clusters images belong to - increasing the margin of error for our accuracy and rand accuracy.

Moreover, from our results, we have discovered that our models are overfitting for our training data. However, Gaussian mixtures outperformed Error-correcting output codes in the unseen data. ECOC performed terribly on the test data, which could bring up many potentially valid points. Our feature extraction procedure on the test data might have adversely affected our accuracy, as well as the random state of our data splitting process as not all images of each class, are evenly distributed across the train, validation, and test sets.

3.2 Hyperparameters, Structural Choices & Complexity

Considering the hyperparameters we tuned for our Error-correcting output codes, there was much more room for improvement as we could have tested other binary classifiers other than random forests such as a single decision tree, support-vector machines, and logistic regression to name a few. In addition to that, our code size was an integral parameter for specifying the layout of our code book for classifying each image.

On the other hand, the hyperparameters applied and tuned for Gaussian mixture models contributed more to the spread and dimensional layout of the clusters. This could be a valid reason as to why our test data performed better using Gaussian mixture models.

In conclusion, both of our models performed as was expected and managed to transform a highly complex task into multiple binary classifications for Error-correcting output codes, as well as an accurate hybrid distribution of several Gaussian curves to model every single class in our Describable Texture Dataset.

References

Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *J. Artif. Int. Res.*, 2(1):263–286.

Brownlee, J. (2021, April 26). Error-correcting output codes (ECOC) for Machine Learning. Machine Learning Mastery. Retrieved December 4, 2021, from https://machinelearningmastery.com/error-correcting-output-codes-ecoc-for-machine-learning/.

Carrasco, O. C. (2020, February 21). Gaussian mixture models explained. Medium. Retrieved December 4, 2021, from https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95.

Learn. scikit. (n.d.). Retrieved December 4, 2021, from https://scikit-learn.org/stable/index.html.

News¶. scikit. (n.d.). Retrieved December 4, 2021, from https://scikit-image.org/.

Scipy. A blue circle with a snake in the shape of the letter 'S'. (n.d.). Retrieved December 4, 2021, from https://scipy.org/.