Continent Classification using Stacking and Voting Classifiers

SYSC 5405 Pattern Classification and Experiment Design

Group 11
Aishwarya Eswaramoorthy (101208111)
Mahitha Sangem (101212458)
Surya Alagathi Ekantharajan (101258802)
Venkatesan Mohan (101230705)

December 2022

CONTINENT CLASSIFICATION USING STACKING AND VOTING CLASSIFIERS

Introduction –

The presented challenge is a multi-class image classification problem in which we must predict the class of an image. In order to train the system, we have been provided with the train data which consists of the true labels of the images and once the algorithm is developed, we have to predict the class labels on the test data. There are 5 different class labels in the train data which are 0,1,2,3, and 4. Each of the images belongs to one of these classes. Apart from the images, we were also provided with image2vec embeddings of the images, and we chose to solve this problem using classical machine learning algorithms by training the models using image2vec embeddings.

Methodology Used -

Method – 1: Stacking using KNN, Decision Tree, Random Forest, and Support Vector Machine.

In this methodology, we have trained the four different base learners which are KNN, Decision Tree, Random Forest, and SVM. Once the base learners are trained, the stacking classifier is trained on top of these base learners to obtain the final predictions. Various machine learning models such as Decision Tree, Random Forest, Gradient-Boosted Decision Tree, XGBoost, and AdaBoost are used as stacking classifiers to compare the performance with each other.

Method – 2: Voting using KNN, Decision Tree, Random Forest, and Support Vector Machine.

As part of the other meta-learning strategy, we used a voting classifier on the predictions of the base learners that have been trained as described in Method 1.

Further sections of the report cover the specific approach to training utilizing these methodologies.

Source of Implementation –

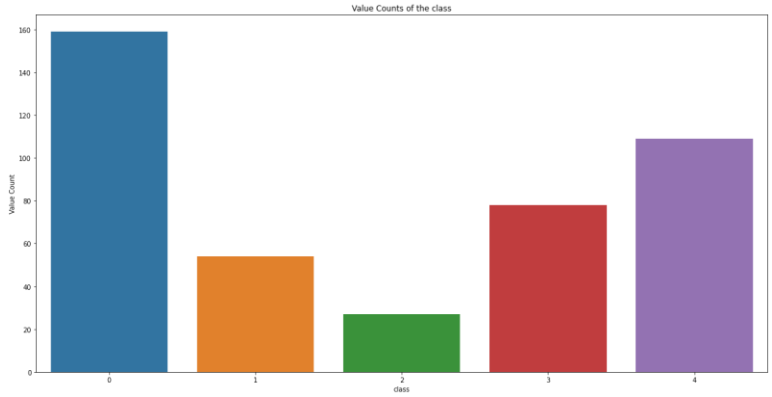
The summary from [1] is that stacking is an ensemble learning approach that combines multiple machine-learning algorithms via meta-learning. Predictions of base learners are used as features to train a meta-classifier, which is responsible for the final prediction. It can be thought of as an extension to the voting classifier illustrated in [3], which has also been implemented for this project. These ensembles are useful when we have varied base learners (with varied features), and are well known to improve the overall predictive performance.

For solving the given problem, we implemented the proposed methodologies using the scikit-learn package in Python.

We used pipelines for sequentially applying transforms (tuning, ADASYN, PCA), using the imblearn library in Python. The same library is used for implementing oversampling which is ADASYN.

Analysis, Pre-processing, and Feature Filtering using Train Data -

The training dataset consists of 427 images and their img2vec encodings and each of the encodings is of 1024 dimensions. The below figure shows the class distribution of the images in the training dataset. The data is imbalanced, there are more images that belong to class 0 and class 1 than the other classes as shown in figure 1.



class	counts	Percentage
0	159	37.23653
4	109	25.52693
3	78	18.26698
1	54	12.64637
2	27	6.32319

Figure 1: Class Distribution of the classes in the training data

As part of processing the data, we checked for the missing values in the data and found that there were no missing values. In the next step, we wanted to remove the outliers present in the data. Hence, we considered removing the values which are more than the 99.9th percentile in the given feature, however, we performed this operation, we ended up having only 1 sample out of 427 samples present in the dataset. Hence, we didn't remove any outliers from the train data. Now, the features have been normalized using Min-Max Scaler and the values of the features are transformed into the range of 0 to 1. Furthermore, feature filtering is performed by computing Pearson's correlation coefficient (PCC) between the features and removing one of the features from a pair of highly correlated features, for this purpose, the threshold value of 0.9 is chosen and the following logic is applied - if A and B are two features and have PCC(A, B) greater than 0.9, filter out feature A if PCC(A, Target label) is less than PCC(B, Target label), otherwise filter out feature B. Following the logic, the features - {'feature_107', 'feature_211', 'feature_233', 'feature_336'} have been filtered out.

Training the base learners –

The training dataset is further split into train and test in the ratio of 80:20 and hence we have 341 samples for training and 86 samples for evaluating the performance. In order to preserve the class distribution, we have used stratified split which would preserve the class distribution in these train and test sets.

1) Decision Tree –

   Step 1: For training the decision tree base learner, at first, the optimal parameters are obtained using the hyperparameter tuning with 5-fold cross-validation on the train data (341 samples). The parameters - max_depth, min_samples_leaf, and criterion are considered as the hyperparameters, and obtained values from grid search are criterion='entropy', max_depth=5,

min_samples_leaf=10. With these parameters, the decision tree is trained and tested on the test data (86 samples) to obtain the performance metrics

Step 2: In this step, the feature selection is performed, and the optimal number of features is obtained using the PCA method. For finding the optimal number of PCA components, the macro-f1 score is computed on the test data at different values of the components. Shown in figure 2.

Step 3: We have repeated the procedure followed in Step 1 but in each of the folds we did data balancing using ADASYN and found the hyperparameters from grid search as criterion='gini', max_depth=5, min_samples_leaf=50.

Step 4: We repeated the procedure followed in Step 2 but used decision tree parameters obtained from step 3 and also performed data balancing on the train set using the ADASYN algorithm.

The four different results obtained from each of the steps are shown in table 1. The confusion matrix for the selected best-performing implementation is shown in Figure 3.

Classifier	f1-score	Precision	Recall
Decision Tree	0.3225	0.3265	0.35
Decision Tree with PCA	**0.4468**	0.5363	0.42
Decision Tree with ADASYN	0.3682	0.3544	0.4
Decision Tree with ADASYN and PCA	0.3755	0.3707	0.38

Table 1: Decision Tree Results on the Test Data (86 samples)

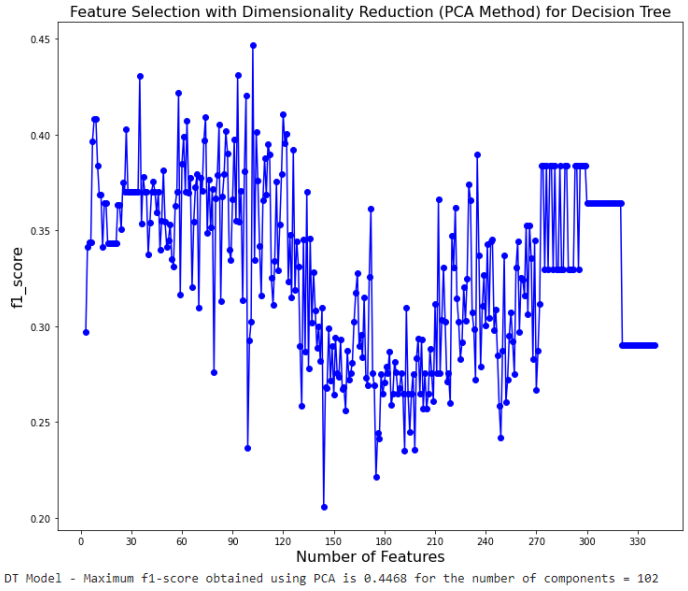


Figure 2: f1_score vs No of features with PCA and ADASYN for Decision Tree no of components = 102

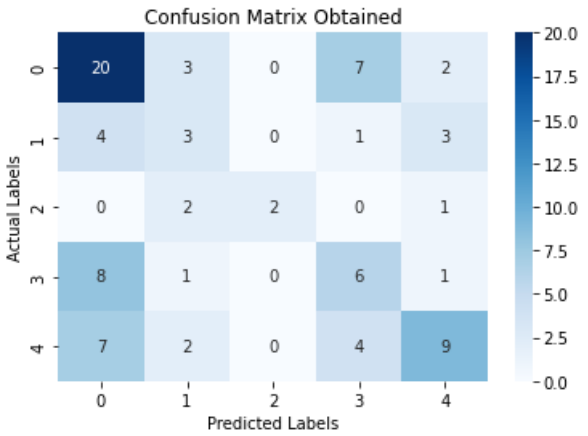


Figure 3: The confusion matrix obtained for Decision Tree - After Hyperparameter Tuning and PCA feature selection

2) K Nearest Neighbors –

The steps discussed in training the Decision Tree are followed for training the KNN. The hyperparameter used for KNN is n_neighbors and the obtained value in Step – 1 for n_neighbors is 15 whereas in Step 3 the value is 1.

The four different results obtained from each of the steps are shown in table 2. The confusion matrix for the selected best-performing implementation is shown in Figure 5.

Classifier	f1-score	Precision	Recall
KNN	0.3926	0.4721	0.41
KNN with PCA	0.4443	0.5055	0.45
KNN with ADASYN	0.4634	0.457	0.52
KNN with ADASYN and PCA	**0.464**	0.4562	0.52

Table 2: KNN Results on the Test Data (86 samples)

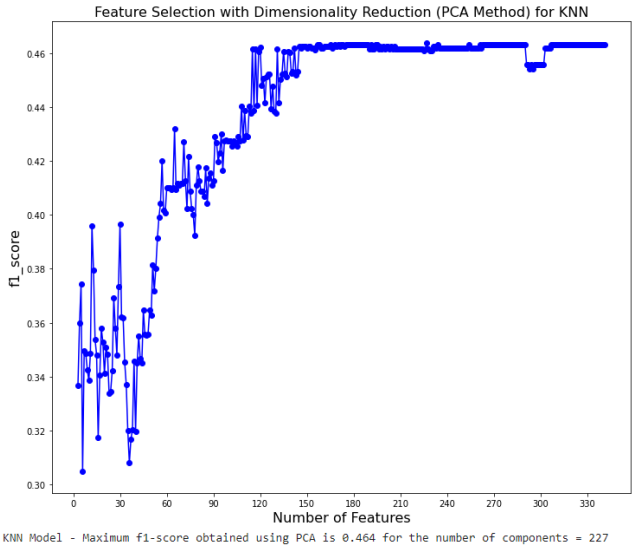


Figure 4: f1_score vs No of features with PCA and ADASYN for Decision Tree no of components = 227

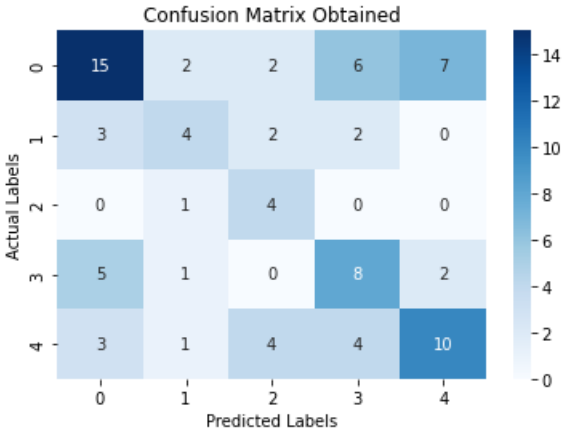


Figure 5: KNN- After Hyperparameter Tuning, ADASYN and Feature Selection based on PCA

3) Random Forest –

A similar procedure which is discussed in training the earlier models is also followed to train the Random Forest. The hyperparameters considered in Random Forest are criterion, max_depth, n_estimators, and the values obtained for these in Step – 1 are criterion='entropy', max_depth=15, n_estimators=40 whereas the values obtained for these in Step – 3 are criterion='gini', max_depth=15, n_estimators=100.

The four different results obtained from each of the steps are shown in table 3. The confusion matrix for the selected best-performing implementation is shown in Figure 7.

Classifier	f1-score	Precision	Recall
Random Forest	0.3915	0.478	0.3701
Random Forest with PCA	0.4926	0.5752	0.4681
Random Forest with ADASYN	0.4742	0.507	0.5145
Random Forest with ADASYN and PCA	**0.5716**	0.5825	0.5657

Table 3: Random Forest Results on the Test Data (86 samples)

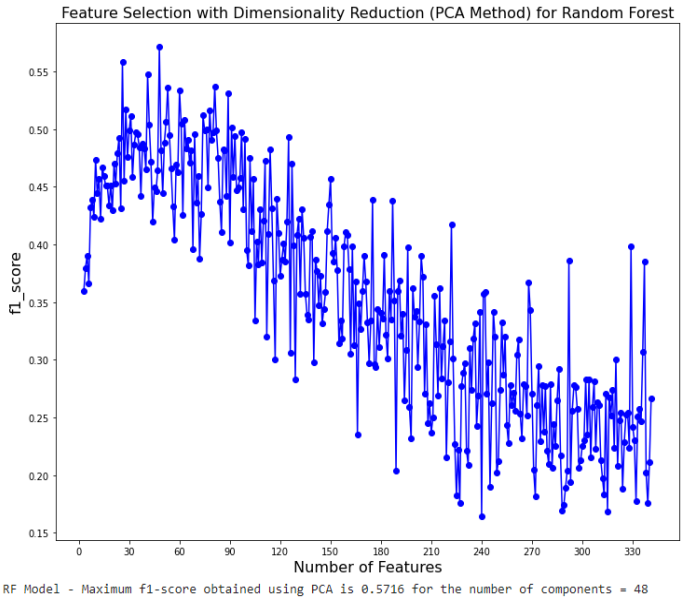


Figure 6: f1_score vs No of features with PCA and ADASYN for Decision Tree no of components = 48

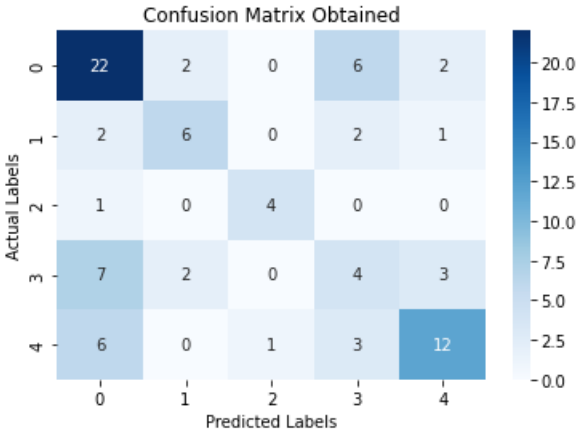


Figure 7: Random Forest ADASYN- After Hyperparameter Tuning, ADASYN and Feature Selection based on PCA

4) Support Vector Machine –

For SVM the hyperparameters used are the kernel, C, and gamma. In Step 1 the values obtained for these are kernel='rbf', C=29.763514416313132, and gamma='scale'. Whereas in Step 3 the values obtained are kernel='rbf', C=11.288378916846883, and gamma='scale'.

The four different results obtained from each of the steps are shown in table 4. The confusion matrix for the selected best-performing implementation is shown in Figure 8.

Classifier	f1-score	Precision	Recall
SVM	0.5208	0.5509	0.512
SVM with PCA	0.5255	0.5571	0.5183
SVM with ADASYN	**0.5282**	0.5751	0.5551
SVM with ADASYN and PCA	0.524	0.5371	0.5327

Table 4: SVM Results on the Test Data (86 samples)

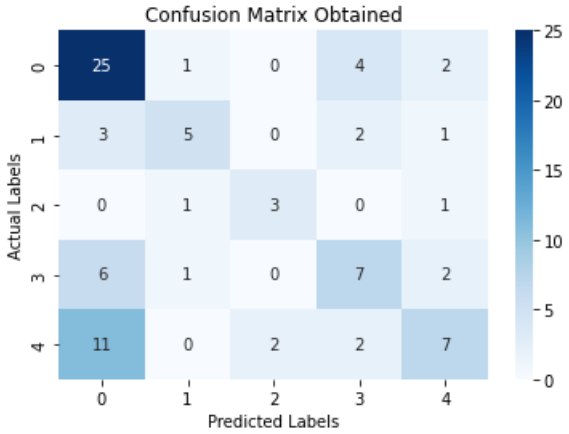


Figure 8: SVM - After Hyperparameter Tuning and ADASYN

Training the Stacking Classifiers –

We utilized the base learners that gave the best f1-score among the four outcomes obtained for each base learner to train the stacking classifiers. The best results on test data were obtained using the Decision Tree with PCA, KNN with ADASYN and PCA, SVM with ADASYN, and Random Forest with ADASYN and PCA, hence these base learners are used as input to the stacking classifiers. The same 80:20 ratio used for training the base learners is employed for training and evaluating the stacking classifiers. The different machine learning models such as Decision Tree, Random Forest, AdaBoost, XGBoost, and GBDT are used as Stacking Classifiers to obtain the predictions. The performance metrics obtained are shown in Table 5 and confusion matrices in figures 9, 10, 11, 12 and 13.

Stacking Classifier	f1-score	Precision	Recall
Random Forest	0.3667	0.4056	0.3573
AdaBoost	0.3341	0.3822	0.3201
XGBoost	0.3444	0.3665	0.3383
GBDT	0.269	0.2727	0.283
Decision Tree	0.3028	0.345	0.2834

Table 5: Stacking Classifier Results on the Test Data (86 samples)

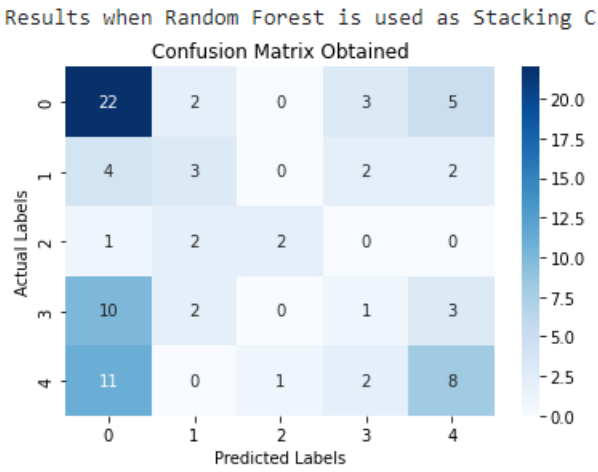


Figure 9: Random Forest is used as Stacking Classifier

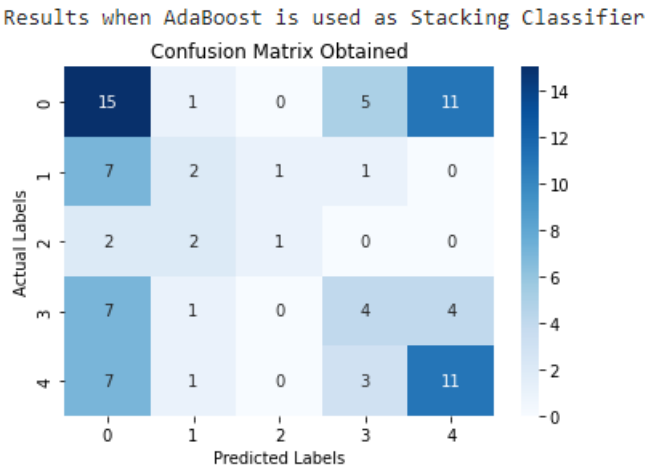


Figure 10: AdaBoost is used as Stacking Classifier

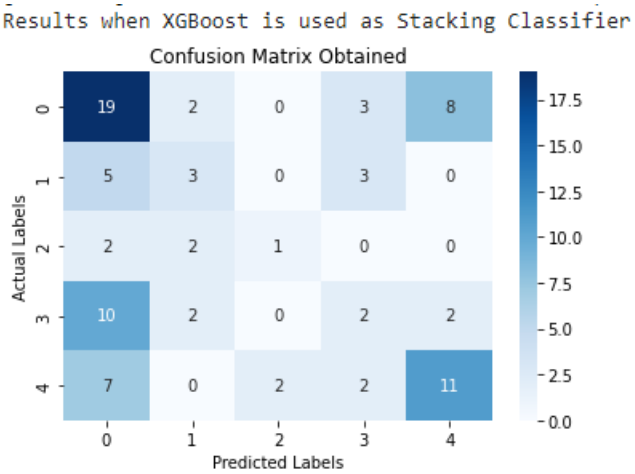


Figure 11: XGBoost is used as Stacking Classifier

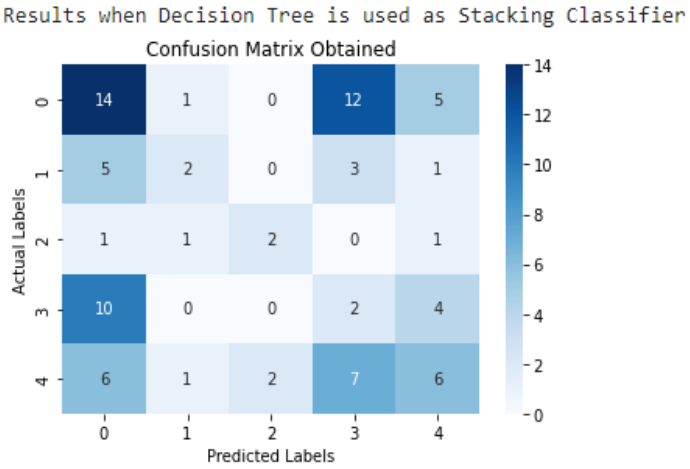


Figure 12: Decision Tree is used as Stacking Classifier

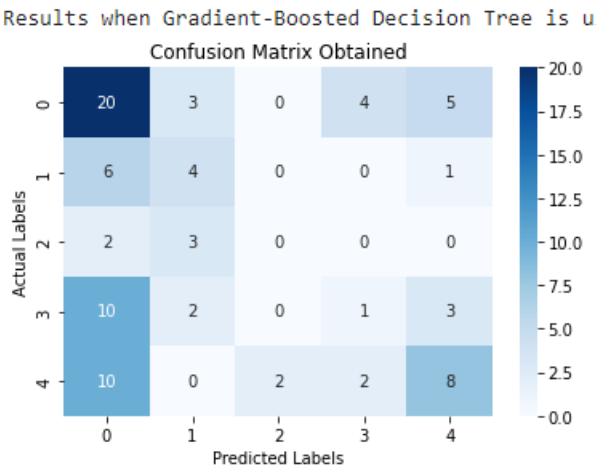


Figure 13: Gradient-Boosted Decision Tree is used as Stacking Classifier

Training the Voting Classifier –

For training the voting classifier, we have used a similar procedure followed for training the stacking classifier but instead used the Voting Classifier as the meta classifier to make the predictions on the test data (86 samples). The obtained results from Voting Classifier are tabulated in table 6, and the confusion matrix in figure 14.

Meta Classifier	f1-score	Precision	Recall
Voting Classifier	0.6163	0.6473	0.6219

Table 6: Voting Classifier Results on the Test Data (86 samples)

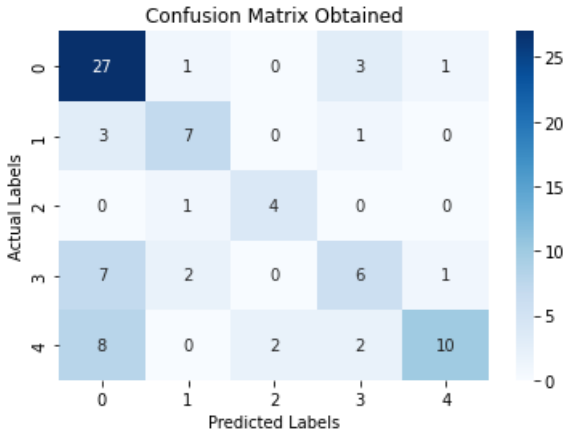


Figure 14: Voting Classifier

When the results of the Stacking and the voting classifiers are compared, the voting classifier produced a better macro f1-score than the stacking classifiers. However, the results might just be attributable to how the data was split. As a result, we ran 5-fold cross-validation on the complete training dataset of 427 samples to determine the average f1-score throughout the 5 folds. The values obtained are shown in Table 7.

Meta Classifier	Macro f1-score (Average across 5 folds)	Standard Deviation
Voting Classifier	0.478453084	0.0449
Random Forest -Stacking Classifier	0.458083039	0.0462
AdaBoost -Stacking Classifier	0.283307852	0.0494
XGBoost -Stacking Classifier	0.424438795	0.0564
GBDT -Stacking Classifier	0.401012923	0.0696
Decision Tree -Stacking Classifier	0.338919815	0.0383

Table 7: 5-fold average Macro f1-score obtained by classifiers on the Train Data (427 Samples)

As indicated in table 7, the f1-score achieved for the Voting Classifier is more than that acquired for the stacking classifiers; hence, we opted to train the Voting classifier on the complete 427 samples of data to predict the class labels of the test data that lacks label information.

Estimated and Actual macro f1-score using the Voting Classifier -

We estimated the overall f1-score on the test data to be **0.4785 ± 0.045**, which is the same as the Voting Classifier's average f1-score on the 5 distinct folds of train and test data derived from 427 samples of training data, with the corresponding standard deviation.

However, the actual f1-score obtained by the voting classifier on the test data is **0.4154**, which is within 2 standard deviations, and the confusion matrix obtained is shown in Figure 15.

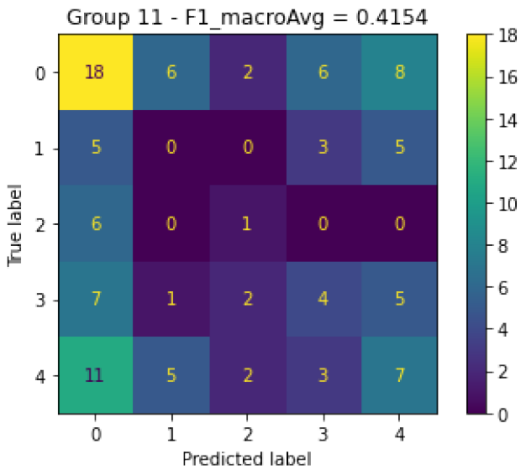


Figure 15: Confusion Matrix obtained when Voting Classifier is used on Test Data (107 Samples)

Analysis of test data results –

The testing data set had 107 samples/img2vec encodings with each having a unique id bundled into a CSV file. We removed the four features {'feature_107', 'feature_211', 'feature_233', 'feature_336'} from feature filtering performed on train data, and then performed scaling.

The voting classifier performs poorly in classifying the labels. It was unable to accurately identify class 1 data; instead, it predicted that they belonged to one of classes 0, 3, or 4. Furthermore, it was only able to accurately categorize one of the seven actual class 2 records. In the case of class 0 records, it was able to classify only eighteen records out of the forty actual class 0 records. Additionally, it was only able to accurately categorize seven samples from the class 4 records, accounting for just 25% of the total class 4 data (28 samples). Furthermore, the voting classifier properly predicted just four class 3 records, while the remainder were predicted as the other classes, with class 0 and class 4 being the most common. Overall, the voting classifier has struggled to classify the labels correctly and the low macro-average F1 score of 41.54% may be due to several factors, such as imbalanced class distribution in the training data, inadequate model capacity, or poor feature engineering.

Our recommendations for further implementation for improvement–

1) Instead of utilizing image2vec encodings, we would like to use the images and use Deep Learning methods that employ neural networks for the classification of the images.
2) If we had to rely only on the image2vec encodings, then we would try using the features selected using Wrapper-based methods instead of PCA.
3) Moreover, we want to try using different base learners and stacking classifiers instead of using the ones which we have used in our methodology.

4) Furthermore, we want to implement a voting classifier on the predictions of the stacking classifiers to derive the final prediction of the class.

References:

[1] https://towardsdatascience.com/stacking-classifier-approach-for-a-multi-classification-problem-56f3d5e120c8#:~:text=A%20stacking%20classifier%20is%20an,task%20of%20the%20final%20classification.&text=The%20stacking%20classifier%20approach%20can,implement%20a%20multi%2Dclassification%20problem.

[2] http://rasbt.github.io/mlxtend/user_guide/classifier/StackingCVClassifier/

[3] http://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/