Letter Recognition using XGBoost

The task was to design and implement a letter recognition solution. The code is written in python as it gives more flexibility to use numerous libraries. The data provided is in .csv format which contains letters, in the first column, and features, in the next 16 columns. The .csv file is imported as a Pandas (python library) data frame. To check if there are any missing values, null values count is run. There are no null values. The frequency of each letter is also generated as to ensure that each letter count similar. The names of the columns are not in the imported file which causes problems when running machine learning models later, therefore column names are added at this step. Before doing any further analysis, the data is standardized. It means that all features of each letter are transformed such that they have zero mean and unit variance. This step is necessary to make sure that each feature contributes equally to the machine learning algorithm.

Although using large number of features is beneficial but it can also worsen the performance that is curse of dimensionality and it can also cause model overfitting. Due to this reason, the Principal Component Analysis, also known as PCA, is used as a dimensionality reduction method. In this method, the number of features is reduced such that the features are projected into subspace with lower number of features while retaining the information of original number of features. The PCA, firstly, normalizes the data. Since it is already completed above, therefore the next step is to create a co-variance matrix of all the features. The dimension of the matrix is 16x16 as there are 16 features for each letter. Then the eigen vectors and eigen values are calculated from this matrix. The PCA projects the data where it varies the most and these directions are calculated from the eigen vectors corresponding to the largest eigen values. The new reduced features are uncorrelated and have most information of the original features. The letters are then encoded as the numbers using a separate function which uses ord() function to convert each letter into specific Unicode such as B is converted into 66. Once this is done, then these encoded letters array is combined with the reduced features data frame to make a new data frame and their column names are also added.

Before applying the machine learning algorithm, the data frame is split into train data and test data such that only 25% of the data is used as a test data and remaining for training. The model used for recognition is the XGBoost classifier. The XGBoost comes under the class of ensemble models. Ensemble models are those machine learning models which use number of different basic classifier models combined together working in isolation with one another. But unlike other ensemble models, XGBoost goes one step further using the results of the previous model to refine them further in a sequential manner. This way the next model is more focused on reducing only the mistakes made by the previous model. The learning rate is a hyperparameter in the algorithm and using a very low value would surely improve the accuracy but increase the run time, therefore it is kept as 0.01. The max_depth parameter is to limit the number of nodes in the tree and since using a high value will improve the results considerably it is taken as 10. N_estimators represent the number of trees in the classifier, and it is taken as 80. The subsample is the parameter to divide the fraction of parameter used for fitting the individual base learner which is taken as equal to 0.3. Colsample_bytree is the columns sabsample ratio constructed for every tree. The XGBoost model is used from the Sklearn library XGBoost API. Firstly, it builds the model and fits to the training data. Then predictions are made on this model.

The accuracy achieved by the model is 91.30% and precision is 91.37%. Where accuracy shows the number of correctly predicted results divided by total predictions and the precision is the true positive results divided by the predicted positive results. The cross validation is made on the test data with 3 folds giving results 82.96%, 80.74% and 81.63%. The recall score is 91.27% which tell that the number of true positives divided by the number of all the relevant samples identified as positive. The F1 score shows how precise the classifier works and gives a harmonic mean between precision and recall, which is equal to 91.28%. The squared average difference between the true positive and precited positive, described by Root Mean Squared Error (RMSE), is equal to 2.92. The confusion matrix, figure file "confusion_matrix_XGBoost", shows a clearer picture of predictions. The values in the cross are correctly identified whereas the values elsewhere shows the false positives. The key insights of the matrix are that mostly W and U letters are correctly predicted. Another key insight is that the greatest number of false positive for one letter are of P, equal to 24. Out of all its false positives, most of the false positives are predicted as F.

Conclusion

The letter recognition task proves that not only neural networks can achieve accuracies above 90% given right setting of parameters in the algorithm. The XGBoost was run several times with different parameters settings and two major things were considered while figuring out the optimum parameters. First thing was to achieve the highest accuracy without overfitting and secondly, to keep run time as small as possible. From the confusion matrix it is clearly evident that F and P are mistaken mostly, and it is logically correct since they have very similar shape except the only difference that P has two horizontal lines joined together unlike F. Therefore, they have similar features as well.