Affective and Persuasive Computing - Assignment 3

Submitted to Professor Hussein Al Osman

by The Team: Ali Vaseghnia - Hossein Zinaghaji

Faculty of Engineering

University of Ottawa

Fall 2020

This document provides a description for our work towards completing Assignment 3 of the Affective and Persuasive Computing course. In this assignment, we developed a machine learning solution to classify the images in the SMILE dataset. There are two types of images in the dataset which represent “happy” and “neutral” states.

Tools

After experimenting with Weka and TensorFlow, OpenCV and scikit-learn libraries, we decided to develop our model with Python and using the scikit-learn library. The neural network approach we used with TensorFlow had overfitting issue which was expected considering the small size of the dataset. After comparing the OpenCV and scikit-learn libraries, we realized that the scikit-learn has a stronger documentation for the purpose of our work. Also, having the professor’s tutorial in scikit-learn laid the groundwork for a better understanding of the classification process in scikit-learn library.

Features

Since this is a binary classification problem, we first gave numerical values to the two types of emotions we have predicted in this assignment. All the instances of ‘happy’ emotion have been given the value of 1 and the instances of ‘neutral’ are shown with 0.

We used Histogram of Oriented Gradients (HOG) for feature extraction. The following attributes, which have been identified as the optimal values, have been chosen for feature extraction:
Orientations: 10
Pixels per cell: (16, 16)
Cells per block: (2, 2)
To see whether we can get extract more features to help our model perform better, we also used Canny Edge Detection, however, this had a negative effect on the overall accuracy of our model. Therefore, we are only using HOG to extract features from the images.

Classification and Hyperparameters

In order to get the best accuracy from our model, we have used the following classifiers as part of our solution: Decision Tree, Random Forest, Logistic Regression, Support Vector Machines, AdaBoost, Extremely Randomized Trees, and K-Nearest Neighbors. We have also used Stacking as part of our ensemble-learning approach.

Also, we have performed a 10-fold cross-validation with random state for this model as per requirements of the assignment.

Decision Tree

Decision trees are one of the good candidates for classification. The parameters used in the Decision Tree model are:

```
DecisionTreeClassifier(criterion='gini', max_depth=10)
```

Both ‘gini’ and ‘entropy’ criterions were tested on this model and ‘gini’ ended up providing a better accuracy, thus we are using it in this model. The ‘max_depth’ hyperparameter defines the maximum depth of the tree, and is set to 10 for this classifier.

The results for this classifier are:

```
For the Decision Tree model:
       F1 Score: 0.8980985400172896
       Precision: 0.9243128319908506
       Recall: 0.8800000000000001
       ROC AUC: 0.9023684210526316
```

Random Forest

As one of the popular ensemble models discussed in the class, we used the Random Forest classifier with the following parameters in our solution:

```
RandomForestClassifier(criterion='entropy', n_estimators=50, max_depth=None)))
```

The ‘entropy’ criterion is selected over ‘gini’ because of its better accuracy. Number of estimators refers to the number of trees in the forest, and we have set it to 50. We haven’t defined a maximum depth for the tree, which is also the default behaviour for this classifier.

The results of this classifier are:

```
For the Random Forest model:
	F1 Score: 0.9417964627014692
	Precision: 0.9469047619047618
	Recall: 0.9400000000000001
	ROC AUC: 0.9875986842105263
```

Logistic Regression

Logistic regression is one of the popular classifiers for binary classification and we were expecting this classifier to have one of the best accuracies among the chosen classifiers. The parameters used for this classifier are:

```
LogisticRegression(max_iter=200, C=2)
```

Max_iter refers to the maximum number of iterations taken for the solvers to converge. The default value for this parameter is 100, however we noticed an increase in the accuracy of the model when the max_iter was increased, therefore it is set to 200. C hyperparameter is similar to what we have already learned in Support Vector Machines, and similar to SVM, smaller values of C specify stronger regularization.

The results of this classifier are:

```
For the Logistic Regression model:
	F1 Score: 0.9671445499301725
	Precision: 0.9761471861471861
	Recall: 0.96
	ROC AUC: 0.9945
```

The logistic regression classifier has the best F1 Score among the selected models in our solution.

Support Vector Machines

Another popular model for classification, SVM is the next classifier we have used. The parameters of this classifier are as follows:

```
SVC(C=2, kernel='poly', degree=3)
```

C hyperparameter is set to 2. As discussed earlier, smaller values of C specify stronger regularization in the model. Kernel refers to the kernel type to be used in the algorithm, and polynomial kernel has been used in this classifier. The degree parameter is linked to the 'poly' kernel and refers to the degree of the polynomial function.

The results of this classifier are:

```
For the SVM model:
       F1 Score: 0.9622691813962675
       Precision: 0.9714285714285713
       Recall: 0.9550000000000001
       ROC AUC: 0.9934999999999998
```

Adaptive Boost

AdaBoost is a popular boosting algorithm we explored in the class. AdaBoost works by assigning weights to instances in the dataset and doing iterations to find the curve of best fit for the model. The parameter used for this classifier is 'n_estimators' which refers to the number of estimators at which boosting is terminated.

```
AdaBoostClassifier(n_estimators=10)
```

The results of this classifier are:

```
For the AdaBoost model:
       F1 Score: 0.9427615003738238
       Precision: 0.9420528594212805
       Recall: 0.945
       ROC AUC: 0.9721973684210526
```

Extremely Randomized Trees

Another ensemble method, Extremely Randomized Trees goes one step further for randomness of the trees by using a different approach in computing the splits. This classifier is very similar to Random Forests, "but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule" (scikit-learn.org documentation).

The available parameters for this classifier are very similar to Random Forest, and the same parameters as Random Forest has been used for this model:

```
ExtraTreesClassifier(criterion='entropy',n_estimators=50, max_depth=None)
```

The results of this classifier are:

```
For the Extremely Randomized Trees model:
       F1 Score: 0.9522633741953384
       Precision: 0.9563602187286397
       Recall: 0.95
       ROC AUC: 0.9878750000000001
```

K-Nearest Neighbors

The last classifier used in our solution is the KNN model which is popular for solving both classification and regression problems. The first parameter used in this classifier or 'n_neighbors' which stands for the number of neighbors to be used by the algorithm. The second parameter is 'weights' which defines the weight function used in prediction. The parameter's default value is 'uniform', which means all points in each neighborhood are weighted equally. For this problem, we have used the 'distance' weight. Setting the weights parameter to 'distance' weights points by the inverse of their distance. This approach gives closer neighbors of a query point a higher weight than the neighbors that are further away.

```
KNeighborsClassifier(n_neighbors=6, weights='distance')
```

The results of this classifier are:

```
For the K-Nearest Neighbors model:
       F1 Score: 0.9402718315234362
       Precision: 0.9695938375350138
       Recall: 0.915
       ROC AUC: 0.98325
```

Ensemble – Stacking

To further modify our models, we implemented a Stacking Classifier. This ensemble method accepts all the classifiers discussed in the previous section and uses a classifier to compute the final prediction. This allows the solution to use the power of each of the estimators by using each of their outputs as input of a final estimator.

```
StackingClassifier(estimators=models, final_estimator=SVC()
```

The results of this ensemble did not provide us the best accuracy among the models we used. The results of this ensemble are:

```
Stacking Scores:
       F1 Score: 0.9402718315234362
       Precision: 0.9695938375350138
       Recall: 0.915
       ROC AUC: 0.98325
```

Conclusion

In this assignment we worked on solving a binary classification problem to classify pictures of people with either happy or neutral faces. To solve this problem, we used the scikit-learn library in Python.

We used 7 classifiers in our solution and compared the results of each classifier to identify the most accurate model. Also, we used a Stacking ensemble classifier on all 7 models and evaluated all the results to identify the best solution to this problem. To measure the performance of the models, we compared the F1 Score, Precision, Recall and the Area under the ROC (Receiver Operating Curve). As noted earlier in this report, the Logistic Regression classifier was the best performing model and had a F1 Score of 0.9671.

There are a number of approaches to improve our solution, which can be addressed in the future. For example, other feature extraction methods can be used, and the results can be appended to the features array, which can improve the model performance. Moreover, using other ensembles on models can improve the accuracy of the classifiers' predictions.