Facial Expression Recognition Using KNN

Zemin XU and Mohamed ABDUL GAFOOR

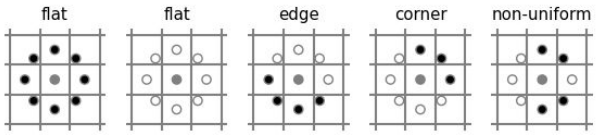
Assignment:

Classification of the emotions into different categories by using feature extraction and KNN technologies.

Theoretical research:

LBP:

Local Binary Pattern(LBP) is an algorithm for feature extraction, especially for texture classification. It looks at points surrounding a central point and tests whether the surrounding points are greater than or less than the central point as grayscale value. When surrounding pixels are all black(darker than center) or all white(brighter than center), then that image region is flat (i.e. featureless). Groups of continuous black or white pixels are considered “uniform” patterns that can be interpreted as corners or edges.



K-Nearest Neighbors

KNN which stands for k-nearest neighbors, is a way applied in Nearest Neighbors Classification. Here, k is an integer that defines that number of samples closest in distance to the chosen data. The optimal choice of the value k varies according to data. In general, a larger k suppresses the effects of noise, but makes the classification boundaries less distinct. Two important parameters are ‘weights’ and ‘algorithm’. The ‘weights’ defines the weight function used in prediction, with the default value as ‘uniform’, which attributes uniform weights to each neighbour. ‘Algorithm’ defines the algorithm to find the nearest neighbours. For the classification task, the label with the biggest k is the prediction. We have used the grid search CV to identify the optimal parameters. The below is the code snippet of the KNN grid search CV

```python
def Grid_Search_CV_KNN(imgs, lable_list):
    estimator = KNeighborsClassifier()

    parameters = {'n_neighbors': [10, 20, 30, 40, 50],
                  'weights': ['uniform', 'distance'],
                   'algorithm': ['auto', 'ball_tree'],
                  'p': [1, 2],
                   }

    grid = GridSearchCV(estimator, parameters, n_jobs=-1, cv=5)

    grid.fit(imgs, lable_list)

    return grid.best_score_ , grid.best_params_

def KNN(imgs_train, imgs_test, lable_list_train, lable_list_test, best_params):
    estimator = KNeighborsClassifier(n_jobs=-1).set_params(**best_params)
    estimator.fit(imgs_train,lable_list_train)
    y_predict = estimator.predict(imgs_test)
    return lable_list_test,y_predict
```

KNN Result: The below figure shows the Best Score, Root Mean Square Error (RMSE) and R2 values. The greedy search function may take up to 30 mins to run, so it is normal when it seems stuck. The accuracy of the KNN is about 32.5% and the RMSE is 2.06.

```
Best Score: 0.32571688086166517
Best params: {'algorithm': 'auto', 'n_neighbors': 50, 'p': 1, 'weights': 'distance'}
Average RMSE:  2.05572
Average R2:  -0.30888
Average BEST_SCORE:  0.32572
```

Random Forest:

Random Forest or random decision forests are an ensemble learning method for classification, regression problems on various sub-samples of the dataset. It uses averaging to improve the predictive accuracy and control over-fitting (Ref 9). The below code snippet is implementation of Random Forest.

```
#Test Random Forest
def Grid_Search_CV_RFR(imgs, lable_list):

    estimator = RandomForestClassifier()
    param_grid = {
            "n_estimators"      : [50,100],
            "criterion"         : ['gini', 'entropy'],
            "max_features"      : ["sqrt", 'auto'],
            "min_samples_split" : [8],
            "bootstrap"         : [False, True],
            }

    grid = GridSearchCV(estimator, param_grid, n_jobs=-1, cv=10)

    grid.fit(imgs, lable_list)

    return grid.best_score_ , grid.best_params_

def RFR(imgs_train, imgs_test, lable_list_train, lable_list_test, best_params):
    estimator = RandomForestClassifier(n_jobs=-1).set_params(**best_params)
    estimator.fit(imgs_train,lable_list_train)
    y_predict = estimator.predict(imgs_test)
    return lable_list_test,y_predict
```

The below figure is the result that we have obtained when we test the random forest. There is a slight improvement in random forest compared to the KNN model. The accuracy of the random forest was about 34% whereas the KNN is only 32%. RMSE value is 1.99 for the random forest.

Random Forest Result:

```
iteration:  0
--------------
Best Score: 0.34194857380727867
Best params: {'bootstrap': False, 'criterion': 'entropy', 'max_features': 'auto', 'min_samples_split': 8, 'n_estimators': 100}
iteration:  1
--------------
Best Score: 0.33968434798544134
Best params: {'bootstrap': False, 'criterion': 'gini', 'max_features': 'sqrt', 'min_samples_split': 8, 'n_estimators': 100}
Average RMSE:  1.9919850000000001
Average R2:  -0.228985
Average BEST_SCORE:  0.340815
```

Why the performance of KNN is bad:

1. The efficiency of KNN declines very fast as the dataset grows.
2. The accuracy of KNN deteriorates with high-dimension data as there is hardly any difference between the nearest and farthest neighbor, KNN suffers from the curse of dimensionality because it is usually implemented using an approximate nearest neighbor search algorithm such as KD-tree.[8]

Why not CNN:

In this experiment, we would like to compare the machine learning models, such as KNN, Random Forest, XGBoost etc. Not to include the deep learning methods in which the ***convolution operations*** are involved. We know CNN will perform better than the KNN, Random Forest, XGBoost for the image classification tasks.

Collaboration:

The way we collaborated is to create a common jupyter file and run it onto Google Colab.By applying the structure of the code provided by the professor and adding our adjustment, we ran successfully. Later, we started to change the parameters to improve the performance. In order to do so, we searched the theoretical part on documentation pages and did the experimental part in code by printing out results.

Reference:

1. Module: feature — skimage v0.18.0.dev0 docs (scikit-image.org)
2. sklearn.neighbors.KNeighborsClassifier — scikit-learn 0.23.2 documentation (scikit-learn.org)
3. 1.6. Nearest Neighbors — scikit-learn 0.23.2 documentation (scikit-learn.org)
4. sklearn.svm.SVR — scikit-learn 0.23.2 documentation (scikit-learn.org)
5. sklearn.svm.SVC — scikit-learn 0.23.2 documentation (scikit-learn.org)
6. https://scikit-learn.org/stable/modules/svm.html
7. Local Binary Pattern for texture classification — skimage v0.18.0.dev0 docs (scikit-image.org)
8. https://in.springboard.com/blog/knn-machine-learning-algorithm-explained/#:~:text=KNN%20algorithm%20is%20a%20good,hacks%2C%20is%20of%20no%20use.
9. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html