SID490182143_code

September 24, 2021

1 COMP5318 Assignment: 490182143

[1]:
```
import h5py
import numpy as np
import matplotlib.pyplot as plt
import os
print(os.listdir("./Input/train"))
```

```
['images_training.h5', 'labels_training.h5']
```

[2]:
```
with h5py.File('./Input/train/images_training.h5','r') as H:
    X_train = np.copy(H['datatrain'])
with h5py.File('./Input/train/labels_training.h5','r') as H:
    y_train = np.copy(H['labeltrain'])

with h5py.File('./Input/test/images_testing.h5','r') as H:
    X_test = np.copy(H['datatest'])
with h5py.File('./Input/test/labels_testing_2000.h5','r') as H:
    y_test = np.copy(H['labeltest'])

# using H['datatest'], H['labeltest'] for test dataset.
print(X_train.shape,y_train.shape)
```

```
(30000, 784) (30000,)
```

1.1 Pre-Processing Data

Normalising Image Data Our first step towards pre-processing the dataset can be achieved by normalising the image data.

[3]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

PCA (100 Components) Once we have used our min-max scaler to normalise the input data, we can run PCA for aiming towards a lower runtime for prediction whilst preserving a competent accuracy.

[4]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 100)
X_train = pca.fit_transform(X_train)
X_test = pca.fit_transform(X_test)
```

Now that we have transformed the image data with normalisation and PCA of 100 components, we can consider the input data to be pre-processed and ready for being classified.

1.2 Classification Algorithms

1.2.1 K-Nearest Neighbours (k = 5)

[7]:
```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
clf_knn = KNeighborsClassifier(n_neighbors = 5)
clf_knn.fit(X_train, y_train)
scores_knn = cross_val_score(clf_knn, X_train, y_train, cv = 10)
knn_score = scores_knn.mean()
```

```
0.85 accuracy for K-Nearest Neighbours Classifier
```

[38]:
```
print("%0.3f accuracy for K-Nearest Neighbours Classifier" % (knn_score))
```

```
0.855 accuracy for K-Nearest Neighbours Classifier
```

1.2.2 Naïve Bayes

[8]:
```
from sklearn.naive_bayes import GaussianNB
clf_gnb = GaussianNB()
clf_gnb.fit(X_train, y_train)
scores_gnb = cross_val_score(clf_gnb, X_train, y_train, cv = 10)
gnb_score = scores_gnb.mean()
```

```
0.77 accuracy for Naïve Bayes Classifier
```

[37]:
```
print("%0.3f accuracy for Naïve Bayes Classifier" % (gnb_score))
```

```
0.766 accuracy for Naïve Bayes Classifier
```

1.2.3 Decision Tree

[21]:
```
from sklearn.tree import DecisionTreeClassifier
clf_dt = DecisionTreeClassifier(criterion = "entropy", random_state = 42)
clf_dt.fit(X_train, y_train)
scores_dt = cross_val_score(clf_dt, X_train, y_train, cv = 10)
dt_score = scores_dt.mean()
```

```
0.76 accuracy for Decision Tree Classifier
```

[35]:
```
print("%0.3f accuracy for Decision Tree Classifier" % (dt_score))
```

```
0.761 accuracy for Decision Tree Classifier
```

1.3 Parameter Tuning Using Grid Search

[30]:
```
from sklearn.model_selection import GridSearchCV
KNeighborsClassifier().get_params().keys()
```

[30]:
```
dict_keys(['algorithm', 'leaf_size', 'metric', 'metric_params', 'n_jobs',
'n_neighbors', 'p', 'weights'])
```

1.3.1 kNN

[32]:
```
knn_grid_params = {"n_neighbors" : [3, 5, 7, 9, 11, 13, 15],
              "weights" : ["distance", "uniform"],
              "metric" : ["minkowski","euclidean","manhattan"]}
knn_gs = GridSearchCV(KNeighborsClassifier(), knn_grid_params, verbose = 1, cv␣
 ↪= 10, n_jobs = -1)
knn_gs.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 42 candidates, totalling 420 fits
```

[32]:
```
0.8588666666666667
```

[42]:
```
print(knn_gs.best_params_, knn_gs.best_score_)
```

```
{'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'uniform'}
0.8588666666666667
```

Output and save file for kNN using hyper-parameters as it performs the best of all 3 classifiers with and without parameter tuning.

[71]:
```
clf_knn_hp = KNeighborsClassifier(n_neighbors = 5, metric = "manhattan",␣
 ↪weights = "uniform")
clf_knn_hp.fit(X_train, y_train)
predict = clf_knn_hp.predict(X_test)
output = predict
```

[73]:
```
import numpy as np
with h5py.File('./Output/predicted_labels.h5','w') as H:
    H.create_dataset('Output',data=output)
H.close()
```

1.3.2 Naïve Bayes

[45]:
```
GaussianNB().get_params().keys()
gnb_grid_params = {'var_smoothing': np.logspace(0,-9, num=100)}
gnb_gs = GridSearchCV(GaussianNB(), gnb_grid_params, verbose = 1, cv = 10,␣
 ↪n_jobs = -1)
gnb_gs.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
```

[45]:
```
GridSearchCV(cv=10, estimator=GaussianNB(), n_jobs=-1,
             param_grid={'var_smoothing': array([1.00000000e+00, 8.11130831e-01,
6.57933225e-01, 5.33669923e-01,
       4.32876128e-01, 3.51119173e-01, 2.84803587e-01, 2.31012970e-01,
       1.87381742e-01, 1.51991108e-01, 1.23284674e-01, 1.00000000e-01,
       8.11130831e-02, 6.57933225e-02, 5.33669923e-02, 4.32876128e-02,
       3.51119173e-02, 2.848035…
       1.23284674e-07, 1.00000000e-07, 8.11130831e-08, 6.57933225e-08,
       5.33669923e-08, 4.32876128e-08, 3.51119173e-08, 2.84803587e-08,
       2.31012970e-08, 1.87381742e-08, 1.51991108e-08, 1.23284674e-08,
       1.00000000e-08, 8.11130831e-09, 6.57933225e-09, 5.33669923e-09,
       4.32876128e-09, 3.51119173e-09, 2.84803587e-09, 2.31012970e-09,
       1.87381742e-09, 1.51991108e-09, 1.23284674e-09, 1.00000000e-09])},
             verbose=1)
```

[46]:
```
print(gnb_gs.best_params_, gnb_gs.best_score_)
```

```
{'var_smoothing': 0.0003511191734215131} 0.7680333333333332
```

1.3.3 Decision Tree

[50]:
```
DecisionTreeClassifier().get_params().keys()
dt_grid_params = {"criterion" : ["gini", "entropy"]}
dt_gs = GridSearchCV(DecisionTreeClassifier(), dt_grid_params, verbose = 1, cv␣
↪= 10, n_jobs = -1)
dt_gs.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 2 candidates, totalling 20 fits
```

[50]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy']}, verbose=1)
```

[51]:
```
print(dt_gs.best_params_, dt_gs.best_score_)
```

```
{'criterion': 'entropy'} 0.7603666666666666
```