Comp 5318 Assignment 1

Group Member:
RUIXIAN LIU SID:510127572
FEIFAN GAO SID:510583158

Introduction

In this assignment, we first preprocessed the data, and then used four different models for training, each using the 10-fold Grid Search method to derive the highest accuracy for each model, as well as the training time. Finally, the results were analyzed and the most suitable model was selected.

Methods

Step 1: Preprocessing approaching

1.1 Normalization

We apply MinMaxScaler to normalize the data_train(X_train) and data_test(X_test)

1.2 Dimensionality Reduction

Principal Component Analysis (PCA)
After normalization, introducing the PCA Dimensionality Reduction technique, reduce the number of feathers from (30000, 784) to (30000, 188), which preserves 95 percent of the variance. 188 principal components are required to preserve 95% of the variance on the closing image dataset.

Step 2: Classification algorithms chosen

Algorithms

- K nearest neighbors
- Bagging ensemble of decision trees
- Gaussian Naïve Bayes
- SVM

Package

- sklearn.neighbors.KNeighborsClassifier
- sklearn.tree.DecisionTreeClassifier
- sklearn.ensemble.BaggingClassifier
- sklearn.naive_bayes.GaussianNB
- sklearn.svm.SVC

Step 3: Hyperparameter tuning and validation

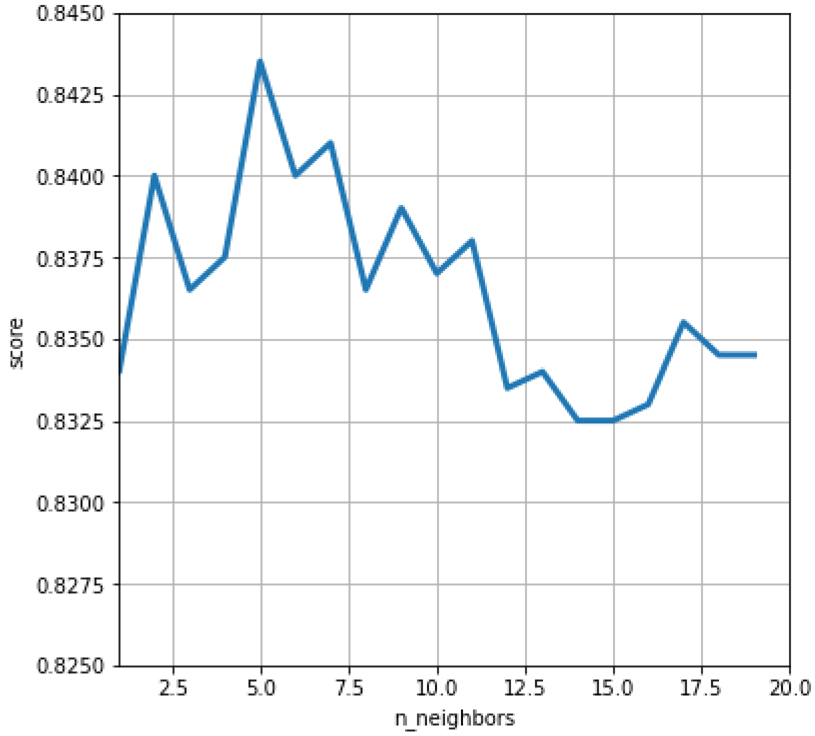
We implement grid search for parameter tuning for KNN and SVM Classifiers since Naïve Bayes has almost no hyperparameters to tune.

We use the 10-fold stratified cross-validation to validate all of the classification algorithms we chose in order to ensure the accuracy of each training result.

We combine them together to grid search with 10-fold stratified cross-validation(applying sklearn.model_selection.GridSearchCV).

Before the KNN grid search, we first compared the exact scores of 1-30 neighbors using a loop and found that the highest values were between 2 and 10. Therefore, in the grid.search, we randomly selected three numbers between 2 and 10 for the search.

The gamma we used in the previous step is taken automatically, i.e. 1/feature, and the accuracy score obtained is very high. Therefore, in the grid search, we took the gamma to two values around 1/188.



Experiments result and discussion

1.Time Comparison

Preprocessing

2.1 Normalization

```
from sklearn.preprocessing import MinMaxScaler

time_start_pre = time.time()

scaler = MinMaxScaler()
scaler.fit(data_train)
data_train = scaler.transform(data_train)
data_test = scaler.transform(data_test)
```

2.2 Dimensionality Reduction

apply PCA without reducing the dimensionality and compute the minimum number of dimensions (feature

```
from sklearn.decomposition import PCA

# pca=PCA(n_components=0.95)
# data_train_reduced = pca.fit_transform(data_train)

# print("Reduced shape of training data: {}".format(str(data_train_reduced.shape)))
pca = PCA()
pca.fit(data_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
d
```

```
188
```

```
# pca train
pca = PCA(n_components = 188)
data_train_reduced = pca.fit_transform(data_train)
data_test_reduced = pca.transform(data_test)

time_end_pre = time.time()
# data_train_recovered = pca.inverse_transform(data_train_reduced)
print("Reduced shape of training data: {}".format(str(data_train_reduced.shape)))
print("Reduced shape of testing data: {}".format(str(data_test_reduced.shape)))
print("time cost for preprocessing",time_end_knn - time_start_knn,'s')
```

```
Reduced shape of training data: (30000, 188)
Reduced shape of testing data: (5000, 188)
time cost for preprocessing 3.882610559463501 s
```

KNN

```
time_start_knn = time.time()
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(data_train_reduced,label_train)
print(knn.score(data_test_reduced[:2000],label_test))
y_pred = knn.predict(data_test_reduced)
time_end_knn = time.time()
print("time cost for knn",time_end_knn - time_start_knn,'s')
```

```
0.8435
time cost for knn 3.91951847076416 s
```

Decision Tree

```
from sklearn.ensemble import BaggingClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=700,
    max_samples=500, bootstrap=True, random_state=42)
time_start_dt = time.time()
bag_clf.fit(data_train_reduced, label_train)
y_bagging_pred = bag_clf.predict(data_test_reduced[:2000])

from sklearn.metrics import accuracy_score

print("Bagging ensemble of decision trees - accuracy on test set:")
print(accuracy_score(label_test, y_bagging_pred))
time_end_dt = time.time()
print("time cost for dt:",time_end_dt - time_start_dt," s ")
```

```
Bagging ensemble of decision trees - accuracy on test set:
0.77
time cost for dt: 53.335368156433105  s
```

Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
time_start_nb = time.time()
nb.fit(data_train_reduced, label_train)
y_pred_GaussianNB = nb.predict(data_test_reduced[:2000])
print("Accuracy on test set: {:.3f}".format(accuracy_score(y_pred_GaussianNB, label_test)))
time_end_nb = time.time()
print("time cost of naive bayes",time_end_nb - time_start_nb,'s')
```

```
Accuracy on test set: 0.729
time cost of naive bayes 0.08178186416625977 s
```

SVM

```
rbf_svm = SVC(kernel="rbf", gamma=0.006, C=1)
time_start_svm_rbf = time.time()
rbf_svm.fit(data_train_reduced, label_train)
print("SVM with ref kernel C=1, gama=0.006 - accuracy on test set: {:.3f}".format(accuracy_score(y_pred_SVM, label_test)))
time_end_svm_rbf = time.time()
print("time cost for rdf svm",time_end_svm_rbf-time_start_svm_rbf,'s')
```

```
SVM with ref kernel C=1, gama=0.006 - accuracy on test set: 0.855
time cost for rdf svm 22.87382960319519 s
```

Time Comparison Summary

By comparison, we found that Naive Bayes takes the shortest time of 0.08s. Decision Tree takes the longest time of 53.33s.

2. Accuracy Comparison

KNN

```
from sklearn.neighbors import KNeighborsClassifier
# alist = []
# for i in range(1,21):
knn = KNeighborsClassifier(n_neighbors=6)
time_start_knn = time.time()
knn.fit(data_train_reduced,label_train)
time_end_knn = time.time()
print("time cost for knn",time_end_knn - time_start_knn,'s')
print("score: {}".format(knn.score(data_test_reduced[:2000],label_test)))
# print(alist)
param_grid = {'n_neighbors': [3,8,10],
              'p': [2]}
print("Parameter grid:\n{}".format(param_grid))

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10,
                          return_train_score=True)


grid_search.fit(data_train_reduced, label_train)

print("Test set score: {:.2f}".format(grid_search.score(data_test_reduced[:2000], label_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

```
time cost for knn 0.007977962493896484 s
score: 0.8405
Parameter grid:
{'n_neighbors': [3, 8, 10], 'p': [2]}
Test set score: 0.84
Best parameters: {'n_neighbors': 8, 'p': 2}
Best cross-validation score: 0.86
Best estimator:
KNeighborsClassifier(n_neighbors=8)
```

Decision Tree

]:
```
from sklearn.ensemble import BaggingClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=700,
    max_samples=500, bootstrap=True, random_state=42)
time_start_dt = time.time()
bag_clf.fit(data_train_reduced, label_train)
time_end_dt = time.time()
y_bagging_pred = bag_clf.predict(data_test_reduced[:2000])

from sklearn.metrics import accuracy_score

print("Bagging ensemble of decision trees - accuracy on test set:")
print(accuracy_score(label_test, y_bagging_pred))
print("time cost fordt:",time_end_dt - time_start_dt," s ")
```

```
Bagging ensemble of decision trees - accuracy on test set:
0.7695
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

from sklearn.model_selection import cross_val_score
scores = cross_val_score(nb, data_train_reduced, label_train, cv=10)
print("Cross-validation scores: {}".format(scores)) #accuracy for each fold
print("Average cross-validation score: {:.2f}".format(scores.mean())) #average accuracy over all folds
```

```
Cross-validation scores: [0.738      0.74033333 0.73533333 0.73866667 0.752      0.73433333
 0.74066667 0.74733333 0.752      0.75066667]
Average cross-validation score: 0.74
```

SVM

```
param_grid = {'C': [1],
              'gamma': [0.004,1/188,0.006]}
print("Parameter grid:\n{}".format(param_grid))


# Use GridSearcCV on the training set
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
grid_search = GridSearchCV(SVC(), param_grid, cv=10,
                          return_train_score=True)

grid_search.fit(data_train_reduced, label_train)

# Accuracy on test set of the model with selected best parameters:
print("Test set score: {:.2f}".format(grid_search.score(data_test_reduced[:2000], label_test)))


print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

```
Parameter grid:
{'C': [1], 'gamma': [0.004, 0.005319148936170213, 0.006]}
Test set score: 0.86
Best parameters: {'C': 1, 'gamma': 0.006}
Best cross-validation score: 0.87
Best estimator:
SVC(C=1, gamma=0.006)
```

Conclusion

The model that takes the fastest time to train is Naive Bayes, but the accuracy is not high. The models with higher accuracy scores are KNN and SVM. We preferred higher accuracy scores compared to faster times, so we finally chose SVM for prediction.Overall, the **SVM model achieves the highest cross validation of 0.87**. The KNN method is faster and has a cross validation score of 0.86.

Appendix

1.Code expanlation

Since it took a long time to use grid search for the Knn and Svm models, we wrote the relevant code as comments to make the code run quickly. We also put the results of the 10 fold grid search and the generated csv screenshots in the appendix.

2.How to run the code

Open the file in jupyter notebook and click the run button to run the entire code

3.10-grid search result

KNN result

```
param_grid = {'n_neighbors': [3,8,10],
              'p': [2]}
print("Parameter grid:\n{}".format(param_grid))

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10,
                           return_train_score=True)


grid_search.fit(data_train_reduced, label_train)

print("Test set score: {:.2f}".format(grid_search.score(data_test_reduced[:2000], label_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

```
time cost for knn 0.007977962493896484 s
score: 0.8405
Parameter grid:
{'n_neighbors': [3, 8, 10], 'p': [2]}
Test set score: 0.84
Best parameters: {'n_neighbors': 8, 'p': 2}
Best cross-validation score: 0.86
Best estimator:
KNeighborsClassifier(n_neighbors=8)
```

KNN result csv

	mean_fit	std_fit_ti	mean_sc	std_score	param_n	param_p	params	split0_te	split1_te	split2_te	split3_te	split4_te	split5_te	split6_te	split7_te
0	0.01087	0.0007	1.32975	0.02079	3	2	{'n_neigh	0.84433	0.854	0.85533	0.84567	0.84867	0.846	0.86867	0.84567
1	0.01077	0.0006	1.48433	0.02236	8	2	{'n_neigh	0.85333	0.854	0.85333	0.85333	0.85933	0.85133	0.864	0.85067
2	0.01057	0.0008	1.495	0.01699	10	2	{'n_neigh	0.85167	0.85167	0.85033	0.853	0.85967	0.85433	0.86067	0.848

	split8_te	split9_te	mean_te	std_test_	rank_tes	split0_tra	split1_tra	split2_tra	split3_tra	split4_tra	split5_tra	split6_tra	split7_tra	split8_tra	split9_tra	mean_tra	std_train_score
0	0.86	0.85667	0.8525	0.0075	3	0.91881	0.91944	0.91711	0.91915	0.91804	0.91793	0.91881	0.91756	0.91826	0.91619	0.91813	0.00094
1	0.858	0.865	0.85623	0.00485	1	0.88478	0.88452	0.88407	0.88293	0.8833	0.88441	0.88385	0.88459	0.88426	0.88315	0.88399	0.00062
2	0.86433	0.862	0.85557	0.00533	2	0.87748	0.87804	0.87926	0.87781	0.877	0.87763	0.87767	0.87789	0.87719	0.87722	0.87772	0.0006

```python
from sklearn.neighbors import KNeighborsClassifier
# alist = []
# for i in range(1,21):
knn = KNeighborsClassifier(n_neighbors=6)
time_start_knn = time.time()
knn.fit(data_train_reduced,label_train)
time_end_knn = time.time()
print("time cost for knn",time_end_knn - time_start_knn,'s')
print("score: {}".format(knn.score(data_test_reduced[:2000],label_test)))
# print(alist)
param_grid = {'n_neighbors': [3,8,10],
              'p': [2]}
print("Parameter grid:\n{}".format(param_grid))

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10,
                          return_train_score=True)


grid_search.fit(data_train_reduced, label_train)

print("Test set score: {:.2f}".format(grid_search.score(data_test_reduced[:2000], label_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

```
time cost for knn 0.007976531982421875 s
```

SVM result csv

	mean_fit	std_fit_ti	mean_sc	std_score	param_C	param_g	params	split0_te	split1_te	split2_te	split3_te	split4_te	split5_te	split6_te	split7_te	split8_te	split9_te	mean_te	std_test_
0	188.083	4.6472	15.1598	0.16973	1	1	{'C': 1, 'ga	0.22067	0.22767	0.21967	0.23	0.23233	0.218	0.22833	0.227	0.228	0.23333	0.2265	0.00503
1	204.67	0.26091	17.7262	0.06291	1	10	{'C': 1, 'ga	0.10233	0.10133	0.10233	0.10267	0.10167	0.10167	0.10133	0.10233	0.10167	0.102	0.10193	0.00044
2	209.531	0.45009	17.622	0.10123	1	100	{'C': 1, 'ga	0.10133	0.10133	0.10167	0.10167	0.10167	0.10133	0.10133	0.10233	0.10133	0.10133	0.10153	0.00031
3	175.477	0.55194	14.9809	0.05562	10	1	{'C': 10, 'g	0.24767	0.254	0.249	0.26167	0.25933	0.24533	0.25333	0.25967	0.25667	0.257	0.25437	0.00524
4	203.536	0.33837	17.7534	0.05923	10	10	{'C': 10, 'g	0.10233	0.10133	0.10233	0.10267	0.10167	0.10167	0.10133	0.10233	0.10167	0.102	0.10193	0.00044
5	208.821	0.40513	17.6133	0.11296	10	100	{'C': 10, 'g	0.10133	0.10133	0.10167	0.10167	0.10167	0.10133	0.10133	0.10233	0.10133	0.10133	0.10153	0.00031
6	175.418	0.75789	15.0613	0.06176	100	1	{'C': 100, '	0.24767	0.254	0.249	0.26167	0.25933	0.24533	0.25333	0.25967	0.25667	0.257	0.25437	0.00524
7	205.725	1.65862	17.8901	0.13734	100	10	{'C': 100, '	0.10233	0.10133	0.10233	0.10267	0.10167	0.10167	0.10133	0.10233	0.10167	0.102	0.10193	0.00044
8	210.234	1.776	17.8167	0.12573	100	100	{'C': 100, '	0.10133	0.10133	0.10167	0.10167	0.10167	0.10133	0.10133	0.10233	0.10133	0.10133	0.10153	0.00031

	rank_tes	split0_tra	split1_tra	split2_tra	split3_tra	split4_tra	split5_tra	split6_tra	split7_tra	split8_tra	split9_tra	mean_tra	std_train_score
0	3	1	1	1	1	1	1	1	1	1	1	1	0
1	4	1	1	1	1	1	1	1	1	1	1	1	0
2	7	1	1	1	1	1	1	1	1	1	1	1	0
3	1	1	1	1	1	1	1	1	1	1	1	1	0
4	4	1	1	1	1	1	1	1	1	1	1	1	0
5	7	1	1	1	1	1	1	1	1	1	1	1	0
6	1	1	1	1	1	1	1	1	1	1	1	1	0
7	4	1	1	1	1	1	1	1	1	1	1	1	0
8	7	1	1	1	1	1	1	1	1	1	1	1	0