<Machine Learning HW#3>

21300008 강민수

I. Experimental Environment

1. Train data와 Test data의 수

실험 결과의 정확한 비교를 위해 MNIST Train data와 Test data는 shuffling 하지 않은 채로 실험을 진행했다. 매 실험 마다 Train data와 Test data의 수를 다르게 하여 data의 수에 따라 성능이 어떻게 변화하는지 관찰하기 위해 실험 마다 수를 다르게 조합하여 성능을 측정하였다. 실험에 사용된 Data의 수는 아래와 같다.

(1) Raw Image Train data

1000개, 2000, 5000개, 10000개, 25000개, 50000개

(2) Raw Image Test data

100개, 500개, 1000개, 5000개, 10000개

(3) Image Train data projected on Eigenspace

1000개, 10000개, 25000개

(4) Image Test data projected on Eigenspace

100개, 500개, 1000개, 5000개, 10000개

(5) Random Forest

Train: 50,000 Test: 10,000

2. 사용된 k와 eigenspace의 dimension

Raw image를 test할 때에는 k는 1, 5, 10으로 주었다. Eigenspace에 MNIST image를 projection할 때에는 k는 1, 5, 10을, eigenspace의 dimension은 2, 5, 10을 조합하여 실험하였다.

3. 사용된 코드

1) kNN 알고리즘 구현시 설계 고려 사항

kNN은 각 test sample을 모든 training sample에 대해 distance를 구해야 하기 때문에 실험에 걸리는 수행 시간은 train과 test의 sample 수와 data의 dimension에 정비례한다. 그러므로 실행속도와 메모리 소비를 최소화하기 위해 numpy를 사용하여 구현을 수행하였다. numpy 함수를 너무 많이 사용하는 것 또한 속도 저하의 원인이 될 수 있으므로 같은 결과를 내면서 필요 이상으로 많은 numpy 함수를 호출하지 않게 코드를 작성하였다.

2) 코드 분석

(1) kNN 알고리즘

```
for test_img_idx in np.ndindex(test_x.shape[:1]):

        distance = np.linalg.norm(test_x[test_img_idx] - train_x, axis=1)
        # get euclidean distance

        sorted_indices = np.argsort(distance)
        # sorted by order of min dist

        pred_class = stats.mode(self._train_y[sorted_indices[:self._k]])[0][0]
        # predicted class

        if self._test_y[test_img_idx] ==pred_class:
                accuracy += 1
        # count correct answer
```

매 iteration마다 한 개의 test sample이 뽑혀서 전체 train data에 각각의 train sample과의 Euclidean distance를 구한다. 그리고 낮은 순서대로 distance를 갖도록 정렬할 때 그 때의 index를 k개만큼 가져와서 (distance가 낮은 순서에 대응되는 index를 가져와서) 그 index에 대응되는 predicted label들을 뽑아낸다. 그리고 그 predicted label들 중 가장 많이 나온 값이 kNN model이 예측한 label이 되고, 이를 test sample의 원래 정답과 비교해 맞춘 개수를 count한 후 학습이 종료되면 test에 사용된 sample수로 나누어 정확도를 산출한다.

(2) Eigenspace projection

```
def eigenprojection(self, eigenspace_dim = 2):

        cov_matrix = np.cov((self._train_x-np.mean(self._train_x)).T)
        eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
        eigenvectors =eigenvectors.T[:eigenspace_dim].T

        dot_producted_unit_vector = np.dot(eigenvectors, eigenvectors.T)

        projectioned_train = np.dot(self._train_x, dot_producted_unit_vector)
        projectioned_test = np.dot(self._test_x, dot_producted_unit_vector)

        return projectioned_train, self._train_y, projectioned_test, self._test_y

        # project data on eigenspace
```

Eigenspace에 projection을 하여 projected된 data들을 반환하는 함수는 위와 같다. 먼저 train에 사용될 image들을 그 평균으로 빼고 그것을 대상으

로 각 feature 변수의 covariance를 구한다. 그리고 그것을 토대로 eigenvector를 구한 후 원하는 만큼의 eigenvector를 추출하여 eigenspace를 구성한다. 그리고 train과 test를 이 eigenspace에 projection하여 kNN 알고리즘을 적용하기 위한 준비를 마친다.

(3) Random Forest

```
clf = RandomForestClassifier()
clf.fit(images[0], images[1])

print(clf.feature_importances_)

acc = np.mean(np.equal(clf.predict(images[2]), images[3]))
print(acc)
```

Random Forest는 sklearn에 있는 example code를 참조하여 작성하였다.

II. Experimental Result and Analysis

<Raw MNIST Data>

Raw MNIST Data에 대해 kNN을 수행한 결과는 아래와 같다. Train number를 기준으로 정리하였고 각 Train sample의 수를 기준으로 Test sample의 수를 다르게 하여 정확도를 정리하였다. 빨간색으로 표기된 정확도는 각 test sample 수에 대해 가장 최상의 성능을 기록했음을 의미한다.

1) Train number 1000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.83	0.85	**0.828**	**0.828**	**0.8195**	**0.8244**	**0.869**
K=5	**0.84**	**0.86**	0.826	0.815	0.8075	0.8216	0.8582
K=10	**0.84**	0.855	0.8	0.799	0.791	0.7932	0.8448

2) Train number 2000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.86	0.875	0.862	**0.873**	**0.8675**	**0.869**	**0.9028**
K=5	0.9	0.895	0.864	0.866	0.859	0.8656	0.8984
K=10	**0.91**	**0.91**	**0.868**	0.864	0.8565	0.854	0.8895

3) Train number 5000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.9	0.92	0.906	0.902	**0.906**	**0.9108**	**0.9343**
K=5	**0.95**	**0.95**	**0.91**	**0.91**	0.9005	0.906	0.9325
K=10	**0.95**	0.94	0.89	0.898	0.8875	0.8954	0.9251

4) Train number 10000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.92	0.93	**0.922**	**0.92**	**0.9225**	**0.9266**	**0.9463**
K=5	0.95	0.94	0.918	0.916	0.9205	0.9222	0.9442
K=10	**0.96**	**0.945**	0.918	0.915	0.918	0.9174	0.9411

5) Train number 25000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.96	0.96	0.938	**0.942**	0.94	0.9406	**0.9593**
K=5	**0.97**	**0.965**	**0.946**	**0.942**	**0.943**	**0.9408**	**0.9593**
K=10	0.95	0.95	0.94	0.933	0.9385	0.938	0.9562

6) Train number 50000

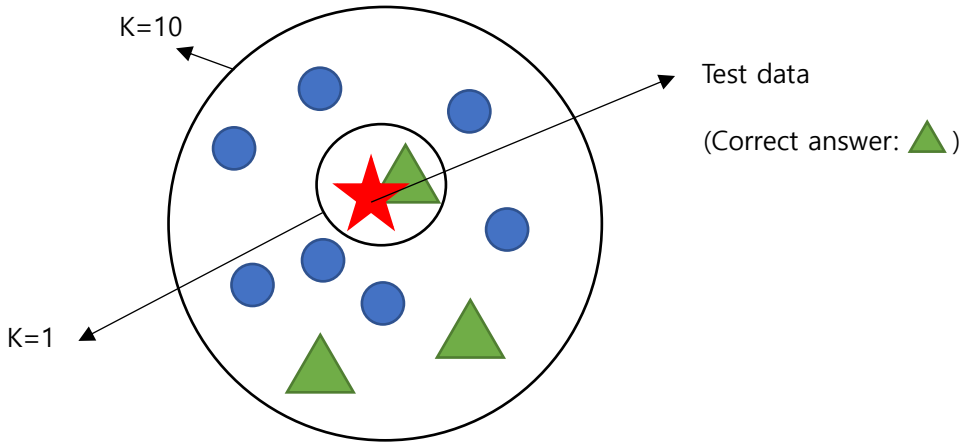
Test Sample	100	200	500	1000	2000	5000	10000
K=1	**1.0**	**0.985**	**0.962**	**0.957**	**0.9535**	**0.9514**	**0.9666**
K=5	0.98	0.975	**0.962**	**0.957**	0.9515	0.9512	0.9664
K=10	0.95	0.965	0.96	0.956	0.951	0.9504	0.9652

7) Experimental Discoveries and Result Analysis

같은 Test sample number와 같은 k값에 대해 Train sample number를 증가시킬수록 전체적인 정확도는 증가함을 알 수 있었다. 이는 모집단을 잘 나타내는 train sample data가 많을수록 모집단을 잘 추정하게 되어 모집단의 확률분포가 train sample data의 분포와 비슷해져 성능이 높아지는 것으로 추정된다.

실험에 앞서 k=1일 때 너무 decision boundary를 training sample에 과적합화시켜 testing data로 classification을 수행하였을 때 정확도가 낮게 나올 것이라고 생각하였다. 그러나 이와 다르게, 실험 결과는 적은 수의 train sample(sample의 수 25,000개 이하)에 대해 대부분의 경우 test sample의 수가 높아질수록, k값이 낮아질 때 정확도가 높아지고, test sample의 수가 적어질수록, k가 높아질 때 정확도가 최상이었다. Train sample의 수가 50,000인 경우에는 대부분의 경우 k=1일 때 성능이 가장 높았다.

위 문제를 분석하며 다음과 같은 결론을 내리게 되었다. 표 1), 2), 3), 4), 5), 6)을 참고하면, Train number의 수가 높아지고 k의 값이 변화함에 따라, k가 증가할수록 얻어지는 정확도의 차이가 줄어듦을 알 수 있다. 이는 train sample data가 모집단을 잘 대표하고 그것의 수가 높아질수록, test data 하나에 대해 거리가 가장 가까운 것을 자신의 neighbor로 두었을 때 최상의 성능을 얻을 수 있음을 의미한다. 하지만 이는 좋은 현상이 아니다. training data가 증가하면서 k와 증가할수록, underfitting이 발생해 nearest neighbor가 아닌 나머지 neighbor들은 noise처럼 인식되어 classification을 방해하게 되기 때문이다. 이를 그림으로 표현하면 아래와 같다.



training data의 분산은 training data의 수가 많을수록 감소하게 되고 모델이 이를 판별하기 쉽지 않게 된다. 이 경우를 그림으로 표현하면 위와 같이 표현할 수 있다. 빨간색 별은 정답이 녹색 세모인데, k=1일 때 가장 가까운 것이 녹색 세모라고 계산되었다면, 녹색 세모라고 예측되어 정답으로 인식이 될 것이다. 그러나, k=10일때를 고려하면, 파란색 원의 수가 녹색 세모의 수보다 훨씬 많으므로 빨간색 별을 파란색 원으로 판단할 것이고 이는 정답이 아니게 된다. 따라서 모델의 복잡도를 정교하게 표현해주지 못해 underfitting이 발생하게 된다. 그러므로 training data의 수가 많아질수록 k가 줄어들 때 정확도가 높게 나

올 수 밖에 없다. 결론적으로 kNN 알고리즘은 Raw MNIST image에 Euclidean distance를 사용하게 된다면, data가 너무 많지도 않고 적지도 않은, 충분한 수가 주어져야 한다.

<MNIST data Projected on Eigenspace>

Eigenspace에 projeted된 MNIST Data에 대해 kNN을 수행한 결과는 아래와 같다. 각각의 기준에 따라 결과를 다양하게 분석하였다.

1) Eigenspace의 dimension이 증가할 때에 따른 분류

아래의 표들은 eigenspace의 dimension이 증가함에 따라 정확도 차이가 어떻게 나는지 알아보기 위해 나열되었다.

(1) Eigenspace dimension 2, Train number 1000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.38	0.415	0.37	0.382	0.3775	0.3692	0.3852
K=5	**0.47**	**0.465**	**0.42**	**0.426**	0.417	0.4046	0.4205
K=10	0.45	0.4	0.394	0.42	**0.425**	**0.416**	**0.434**

(2) Eigenspace dimension 5, Train number 1000

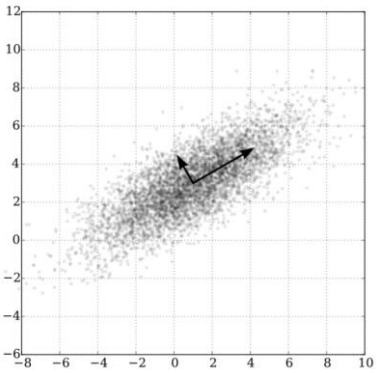
Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.58	0.6	0.592	0.597	0.5915	0.6038	0.638
K=5	0.64	0.675	0.642	0.647	0.6335	0.6404	0.6748
K=10	**0.67**	**0.69**	**0.652**	**0.659**	**0.6515**	**0.6502**	**0.686**

(3) Eigenspace dimension 10, Train number 1000

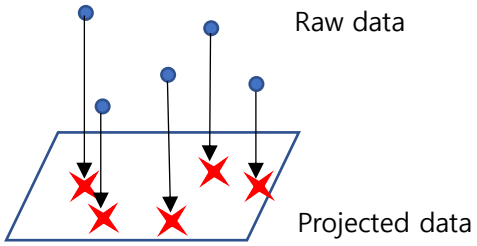
Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.76	0.79	0.78	0.781	0.772	0.7794	0.8115
K=5	0.82	0.845	0.814	0.811	**0.796**	**0.7982**	0.8333
K=10	**0.83**	**0.86**	**0.822**	**0.814**	**0.796**	0.7958	**0.8346**

(4) Analysis

Dataset 전체에 대해 공분산을 구하고, 이에 대해 eigenvector를 구해 eigenvalue가 큰 순서대로 정렬하여 그 중 가장 큰 eigenvalue에 해당하는 eigenvector를 뽑아낸 것은 결국 전체 data를 가장 잘 나타내는 방향(vector)를 뽑아준 것을 의미한다. 그래서 train과 test data들을 그 eigenvector에 projection 하여 kNN을 수행하게 되면, 그것이 data를 얼마나 설명해주는지 알 수 있다.



위 그림의 두 eigenvector를 data가 가장 잘 나타나는 eigenvector라고 가정하자. 그렇다면, dimension이 2일때의 eigenspace는 전체 eigenspace에 span하는 plane형태로 나타낼 수 있고 이는 곧 두 eigenvector가 이루는 subspace가 되고 이를 eigenspace에 projection하는 것은 아래 그림처럼 나타낼 수 있다.



그러므로, eigenvalue가 가장 큰 eigenvector는 나머지 eigenvector보다 전체 data들의 정보를 가장 많이 포함하고 있을 것이다. 하지만 그 eigenvector 만이 data 전체에서 핵심적인 부분만을 전부 나타내지 못하기 때문에, 전체 data를 충분히 나타낼 수 있도록 충분한 dimension을 갖는 eigenspace에 projection을 수행하여야 한다. 이러한 이유로 표 (1), (2), (3)에서 주어진 결과는 eigenvector의 dimension이 높아질수록 높은 정확도를 나타내고 있다. 하지만 dimension이 필요 이상으로 높아지게 되면, 전체 dataset에 있는 noise의 정보도 포함되게 될 것이므로, 적절한 eigenspace의 dimension을 선택하는 것이 중요하다.

(5) Eigenspace dimension 2, Train number 10,000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.41	0.35	0.356	0.374	0.371	0.3632	0.3834
K=5	0.45	0.405	0.392	0.408	0.397	0.3974	0.417
K=10	**0.46**	**0.425**	**0.424**	**0.435**	**0.4235**	**0.4172**	**0.4415**

(6) Eigenspace dimension 5, Train number 10,000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.66	0.69	0.666	0.67	0.646	0.6506	0.6792
K=5	0.71	**0.74**	0.71	0.713	0.695	0.6964	0.7281
K=10	**0.72**	0.735	**0.722**	**0.725**	**0.71**	**0.7088**	**0.7392**

(7) Eigenspace dimension 10, Train number 10,000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.87	0.895	0.866	0.87	0.867	0.8654	0.8891
K=5	**0.93**	**0.935**	**0.896**	**0.897**	0.883	0.8834	0.9062
K=10	**0.93**	0.925	0.89	0.893	**0.884**	**0.8862**	**0.9088**

(8) Eigenspace dimension 2, Train number 25,000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.37	0.35	0.354	0.365	0.3615	0.3636	0.3816
K=5	0.4	0.405	**0.426**	0.407	0.41	0.4002	0.4152
K=10	**0.45**	**0.41**	0.41	**0.425**	**0.4215**	**0.4182**	**0.4344**

(9) Eigenspace dimension 5, Train number 25,000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.64	0.69	0.668	0.671	0.6555	0.6554	0.6824
K=5	**0.71**	**0.75**	0.732	0.739	0.712	0.7026	0.7341
K=10	**0.71**	0.745	**0.738**	**0.744**	**0.723**	**0.719**	**0.754**

(10) Eigenspace dimension 10, Train number 25,000

Test Sample	100	200	500	1000	2000	5000	10000
K=1	0.9	0.905	0.896	0.89	0.8785	0.879	0.9018
K=5	0.94	0.935	0.912	0.906	0.897	0.8974	0.9187
K=10	0.95	0.95	0.918	0.912	0.8975	0.8982	0.9196

(11) Analysis

Raw image에 대해서 kNN알고리즘을 적용하였을 때, Training 수가 증가함에 따라 data의 분산이 적어지고, 그에 따라 underfitting이 심해져 k가 적은 값을 가질 때 높은 성능을 기록하게 됨을 알 수 있었다. 하지만 eigenspace에 projection을 한 MNIST data에 대해서 kNN알고리즘을 적용하여 classification을 적용하였을 때 표 (5), (6), (7), (8), (9), (10)의 결과는 충분한 training data가 주어지고 eigenspace의 dimension만 적절히 선택한다면 underfitting이 raw image보다 덜 발생하게 되고 높은 값의 k로 원하는 결과를 얻을 수 있음을 알게 되었다.

<Random Forest Experiment>

Random forest는 entropy를 통해 decision을 결정하는 여러 개의 decision tree model을 구성하여 다양한 파라미터 조합을 고려하고 각 tree에 대해 독립적으로 학습을 수행해 voting 또는 weighted sum의 형태로 model을 산출한다. 아래 결과는 Random forest classifier를 수행하였을 때, decision tree의 수(# of trees)와 tree의 깊이(max depth)를 조정하여 얻은 정확도 결과를 표로 구성한 것이다. 학습에 사용된 data의 수는 train data 50,000개, test data 10,000개이다. Random forest model은 시도에 따라 정확도가 다르게 산출되기에 같은 hyperparameter(max depth와 tree의 수)에 대해 5번씩 시도하여 그 중 가장 높게 산출된 정확도를 기록하였다.

Max Depth	None	10	50	100	250	500
# of trees = 10	0.95	0.927	0.9457	0.9453	0.9431	0.948
# of trees = 30	0.9631	0.9434	0.9639	0.9632	0.9636	0.9616
# of trees = 50	0.9664	0.9472	0.9652	0.9654	0.9668	0.9658
# of trees = 100	0.969	0.9474	0.9678	0.9683	0.9683	0.969
# of trees = 200	0.969	0.9485	0.9698	0.9699	0.9703	0.971

실험 결과, Tree의 depth(Max Depth)와 Tree의 수가 많아질수록 정확도는 증가함을 알 수 있고 1에 가까운 특정 값으로 수렴하려는 것을 알 수 있다.

Max depth가 None이라는 것의 의미는 각각의 decision tree가 classification을 수행할 때 node의 leaf들이 entropy에 의해 pure하거나 leaf가 포함할 수 있는 최저의 sample수보다 낮다고 판명되면, 더 이상 branch를 나누지 않는 것을 의미한다. decision boundary에 속한 node가 전부 pure할 때 decision boundary를 더 나누는 것은 필요 이상으로 decision boundary를 만들게 되어 overfitting이 발생할 가능성이 더 높아진다. 따라서 data에 대한 정확한 이해가 없고 모델이 복잡해진다면, max depth를 None으로 두어 결과를 산출하는 것이 효율적일 수 있을 것 같다.