exam 개인학습

Lec 1.1

AI summary Try on this page

Type something...

01) K-Nearest Neighbor classifier

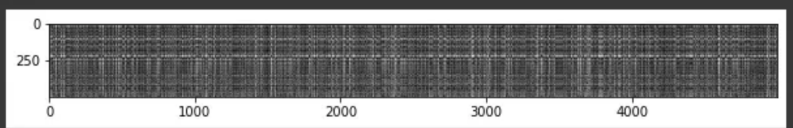
01_Compute_distances_two_loops(self, X):

- 처음에 구현해야 하는 함수는 compute_distances_two_loops(self, X)이다. L2 distance의 정의를 그대로 따라간 함수로 구현을 매우 간단하게 할 수 있었다. 주의해야할 점은 제곱들을 모두 더해준 후에 루트를 씌워야 한다. 순서를 반대로 해서는 안된다. 코드는 다음과 같다.

```
def compute_distances_two_loops(self, X):

    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))
    for i in range(num_test):
        for j in range(num_train):

            # *****START OF YOUR CODE

            dists[i][j] = np.sqrt(np.sum(np.square(X[i] - self.X_train[j])))

            # *****END OF YOUR CODE
    return dists
```

1. Inline Question 1

```
# We can visualize the distance matrix: each row is a single test example and
# its distances to training examples
plt.imshow(dists, interpolation='none')
plt.show()
```



새 섹션

Inline Question 1

Notice the structured patterns in the distance matrix, where some rows or columns are visibly brighter. (Note that with the default color scheme black indicates low distances while white indicates high distances.)

- What in the data is the cause behind the distinctly bright rows?
- What causes the columns?

Answer)

row : 대부분의 train image와 다른 test image가 존재하기 때문.

column : 대부분의 test image와 다른 train image가 존재하기 때문.

2. Inline Question 2

Inline Question 2

We can also use other distance metrics such as L1 distance. For pixel values $p_{ij}^{(k)}$ at location (i,j) of some image I_k,

the mean μ across all pixels over all images is

$$\mu = \frac{1}{nhw}\sum_{k=1}^{n}\sum_{i=1}^{h}\sum_{j=1}^{w}p_{ij}^{(k)}$$

And the pixel-wise mean μ_{ij} across all images is

$$\mu_{ij} = \frac{1}{n}\sum_{k=1}^{n}p_{ij}^{(k)}.$$

The general standard deviation σ and pixel-wise standard deviation σ_{ij} is defined similarly.

Which of the following preprocessing steps will not change the performance of a Nearest Neighbor classifier that uses L1 distance? Select all that apply.

1. Subtracting the mean μ ($\tilde{p}_{ij}^{(k)} = p_{ij}^{(k)} - \mu$.)
2. Subtracting the per pixel mean μ_{ij} ($\tilde{p}_{ij}^{(k)} = p_{ij}^{(k)} - \mu_{ij}$.)
3. Subtracting the mean μ and dividing by the standard deviation σ.
4. Subtracting the pixel-wise mean μ_{ij} and dividing by the pixel-wise standard deviation σ_{ij}.
5. Rotating the coordinate axes of the data.

- L1 거리에서 평균과 pixel 단위에서의 평균을 주고 달라지지 않는 것을 물어보고 있다.

1. 전체 평균은 값이 일정하므로 빼도 거리는 변하지 않는다. (O)
2. pixel 각각의 평균 값은 다르기 때문에 거리가 변한다. (X)
3. 전체 평균과 표준편차는 일정하므로 표준화해도 거리는 변하지 않는다.(O)
4. pixel 각각의 평균과 표준편차는 다르므로 거리는 변한다. (X)
5. 회전시키면 L1 거리가 변한다. (X)

02_predict_labels(self, dists, k = 1):

- `argsort()` 를 이용해서 0, 1, .. k-1번째에 오는 값의 index를 반환해줘서 `y_train` 에서 label를 가져온다. 그리고 `closest_y` 에 append해준다.
- 그 후 `bincount` 을 이용해서 숫자가 나온 빈도수를 구해준 후 `y_pred` 에 저장해준다.

```
def predict_labels(self, dists, k=1):
    num_test = dists.shape[0]
    y_pred = np.zeros(num_test)
    for i in range(num_test):
        # A list of length k storing the labels of the k nearest neighbors to
        # the ith test point.
        closest_y = []

        # START OF YOUR CODE

        closest_y = self.y_train[np.argsort(dists[i])[:k]]

        # END OF YOUR CODE
        # START OF YOUR CODE
        y_pred[i] = np.argmax(np.bincount(closest_y))

        # END OF YOUR CODE

    return y_pred
```

03_compute_distances_one_loop(self, X):

- 행 단위로 한번에 계산해서 저장해준다. 행의 합을 구하기 위해서 `sum(axis = 1)` 을 사용한다.

```python
def compute_distances_one_loop(self, X):
    """
    Compute the distance between each test point in X and each training point
    in self.X_train using a single loop over the test data.

    Input / Output: Same as compute_distances_two_loops
    """
    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))
    for i in range(num_test):
        dists[i] = np.sqrt(np.sum(np.square(X[i] - self.X_train), axis = 1))
    return dists
```

04_compute_distances_no_loop(self, X):

- 힌트: 행렬 곱셈과 2개의 broadcast 합을 이용한다.
- 사이즈를 맞춰줘야 하기 때문에 `reshape` 를 이용해줬다.

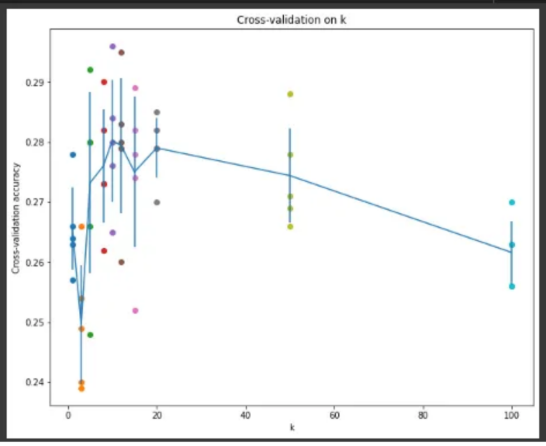
```python
def compute_distances_no_loops(self, X):
    """
    Compute the distance between each test point in X and each training point
    in self.X_train using no explicit loops.

    Input / Output: Same as compute_distances_two_loops
    """
    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))
    # X * X : 500 x 3072
    # X_train * X_train = 5000 x 3072
    # X * X_train = 500 x 5000
    testsum = np.sum(np.square(X),axis = 1)
    trainsum = np.sum(np.square(self.X_train), axis = 1)
    test_train_mul = np.dot(X,self.X_train.T)
    dists = np.sqrt(np.reshape(testsum,[num_test,1]) + np.reshape(trainsum,[1,num_train]) - 2 * test_train_mul)

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return dists
```

05_Cross_validation

- 5차 교차검증을 하기 위해서 데이터셋을 5개로 나눈 후 하나를 검증용으로 사용하였다.
- 각각의 정확도를 구해준 후 딕셔너리에 저장하였다.

```python
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

X_train_folds = []
y_train_folds = []

X_train_folds = np.split(X_train, num_folds)
y_train_folds = np.split(y_train, num_folds)

k_to_accuracies = {}

for k in k_choices:
  k_to_accuracies[k] = []
  for i in range(num_folds):
    X_train_fold = np.concatenate([x for num, x in enumerate(X_train_folds) if num!=i])
    y_train_fold = np.concatenate([y for num, y in enumerate(y_train_folds) if num!=i])

    classifier.train(X_train_fold, y_train_fold)
    y_fold_pred = classifier.predict(X_train_folds[i], k=k, num_loops=0)
    num_correct = np.sum(y_fold_pred == y_train_folds[i])

    accuracy = float(num_correct) / X_train_folds[i].shape[0]
    k_to_accuracies[k].append(accuracy)

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

# Print out the computed accuracies
for k in sorted(k_to_accuracies):
    for accuracy in k_to_accuracies[k]:
        print('k = %d, accuracy = %f' % (k, accuracy))
```

```python
[72] # plot the raw observations
     for k in k_choices:
         accuracies = k_to_accuracies[k]
         plt.scatter([k] * len(accuracies), accuracies)

     # plot the trend line with error bars that correspond to standard deviation
     accuracies_mean = np.array([np.mean(v) for k,v in sorted(k_to_accuracies.items())])
     accuracies_std = np.array([np.std(v) for k,v in sorted(k_to_accuracies.items())])
     plt.errorbar(k_choices, accuracies_mean, yerr=accuracies_std)
     plt.title('Cross-validation on k')
     plt.xlabel('k')
     plt.ylabel('Cross-validation accuracy')
     plt.show()
```



- 그 후 그래프를 그려보면 k = 10 주변에서 정확도가 최대값을 가지는 것을 파악할 수 있었다.

3. Inline Question 3

Inline Question 3

Which of the following statements about k-Nearest Neighbor (k-NN) are true in a classification setting, and for all k? Select all that apply.

1. The decision boundary of the k-NN classifier is linear.
2. The training error of a 1-NN will always be lower than or equal to that of 5-NN.
3. The test error of a 1-NN will always be lower than that of a 5-NN.
4. The time needed to classify a test example with the k-NN classifier grows with the size of the training set.
5. None of the above.

1. k-nn classifier은 거리를 이용해서 구분하기 때문에 보통 linear하지 않는다.(x)
2. 1-NN에서 training error는 0이므로 항상 5-NN보다 에러가 작거나 같다. (O)
3. test error는 문제마다 다르므로 알 수 없다. (X)
4. training size가 커질 수록 비교해야 하는 거리가 늘어나므로 시간이 늘어난다. (O)