<KNN>

```
num_train = 500
num_test = 250
X_train, y_train, X_test, y_test = coutils.data.cifar(num_train, num_test)
[500, 3, 32, 32] → [500, 3072]
```

① two loops

> dists[i][j] : i번째 train과 j번째 test의 거리

```
def compute_distance(X_train, X_test):
    for i in range(num_train):
        for j in range(num_test):
            val = X_train[i] - X_test[j]
            dists[i][j] = torch.sum(val*val)
```

② no loops

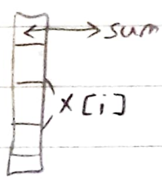
```
def compute_distance(X_train, X_test):
    X_train = X_train.view(num_test, -1)
    X_test = X_test.view(num_test, -1)
```

dists[i][j] = Σ(X_train[i]의 원소)^2 + Σ(X_test[j]의 원소)^2 − 2 Σ X_train[i]·X_test[j]

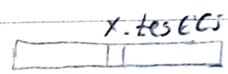
(· ← 내적)

행렬곱 X_train ⊗ X_test

dists =




```
torch.sum(X_train**2, dim=1).view(num_train, 1)
+
```




```
torch.sum(X_test**2, dim=1).view(1, -1)
+
-2 * torch.mm(X_train, X_test.t())
```

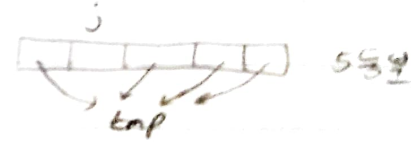
} broadcasting

③ predict label

```
for j in range(num_test):
    val, idx = torch.topk(dists[:, j], k, largest=false)  # 가장 작은 k개의 (값, 인덱스)
    k_near = y_train[idx]
    y_pred[j] = torch.mode(k_near)  # 가장 개수가 많은 값
```

④ 교차 검증

```
def knn_cross_validate(x_train, y_train, num_folds=5, k_choices):

    if k_choices is None:
        k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

    x_train_folds = torch.chunk(x_train, num_folds, dim=0)
    y_train_folds = torch.chunk(y_train, num_folds, dim=0)
                    # num folds 개수로 텐서를 쪼갠다

    k_to_accuracies = {}
    for i in k_choices:
        k_to_accuracies[i] = []
        for j in range(num_folds):
            tmp = [x for x in range(num_folds) if x != j]

            new_x_train = torch.cat([x_train_folds[k] for k in tmp])
            new_y_train = torch.cat([y_train_folds[k] for k in tmp])
            new_x_test = x_train_folds[j]
            new_y_test = y_train_folds[j]

            acc 계산 → k_to_accuracies[i]에 추가
```



그 중 정확도값이 가장 큰 k값을 선택