University of Nottingham
UK | CHINA | MALAYSIA

COMP3055 Machine Learning

Explain the Solution to Lab 3

Ying Weng

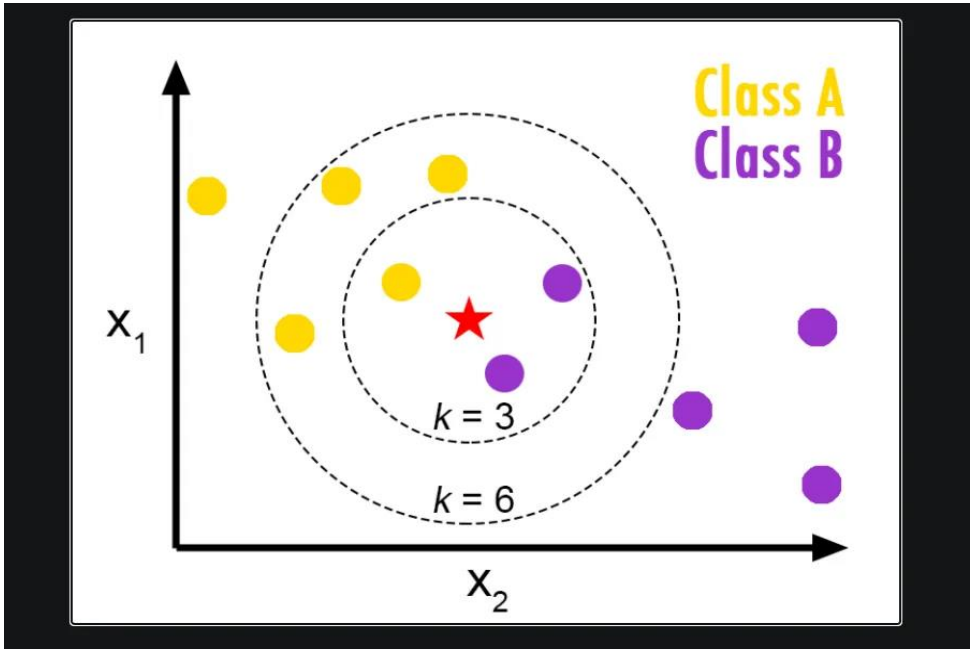
2024 Autumn

All imports in this lab

```
import operator
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.metrics import accuracy_score, confusion_matrix,
f1_score
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
```

kNN pseudo code

1. Calculate distances between the test sample and all samples in training data
2. Sort the distances and get the first 'k' samples from the sorted distances
3. Use a majority vote to get the most frequent class label



Define myknn

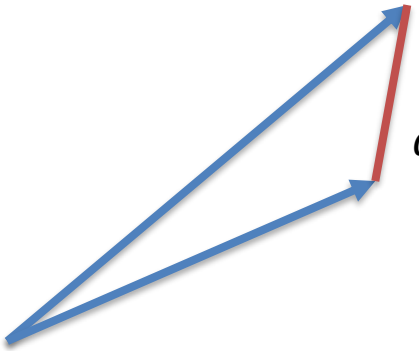
```
def myknn(sample, tr_feats, tr_label, k)

## sample: the test sample we want to classify

## tr_feats: training dataset

## tr_label: labels of the training dataset

## k: hyper-parameter k, the number of of neighbors to consider in the classification

# sample = X[60000]

# using the first 1000 sample for training
tr_feats = X[:1000]
tr_label = y[:1000]

# using the last 10000 sample for tesing
te_feats = X[-10000:]
te_label = y[-10000:]
```

Calculate Euclidean Distance

```
# Repeat sample the number of rowSize times first
rowSize = tr_feats.shape[0]

# Difference calculation
diff = np.tile(sample, (rowSize, 1)) - tr_feats

# Square the differences
sqrDiff = diff ** 2

# Sum the squared differences
sqrDiffSum = sqrDiff.sum(axis=1)

# Compute the square root
distances = sqrDiffSum ** 0.5
```

Define myknn

```
def myknn(sample, tr_feats, tr_label, k):
    rowSize = tr_feats.shape[0]
    # repeat sample the number of rowSize times first
    diff = np.tile(sample, (rowSize, 1)) - tr_feats
    sqrDiff = diff ** 2
    sqrDiffSum = sqrDiff.sum(axis=1)
    # calculate Euclidean distance
    distances = sqrDiffSum ** 0.5

    # sort distance  (from lowest to highest)
    sortDistance = distances.argsort()

    count = {}
    # calculate each class's frequency
    for i in range(k):
        vote = tr_label[sortDistance[i]]
        count[vote] = count.get(vote, 0) + 1

    # sort the frequency of each class  (from highest to lowest)
    sortCount = sorted(count.items(), key=operator.itemgetter(1), reverse=True)
    # return predicted class
    return sortCount[0][0]
```



$$distance = |X_t - X_i|$$

5-fold cross validation

```
for i in range(5):
    tv_feats = X_folds[i] # features of the validation set for i-th fold
    tv_label = y_folds[i] # labels of the validation set  for i-th fold

    tr_feats = np.vstack(X_folds[:i] + X_folds[i + 1:])
# features of the training set: folds before and after the i-th fold
    tr_label = np.hstack(y_folds[:i] + y_folds[i + 1:])
# labels of the training set: folds before and after the i-th fold
```

Evaluation metrics

```
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
```

- Accuracy

  ```
  accuracy_score(tv_label, val_pred)
  ```

- Error rate

  ```
  error_rates = 1 - accuracy_score(tv_label, val_pred)
  ```

- F1 score

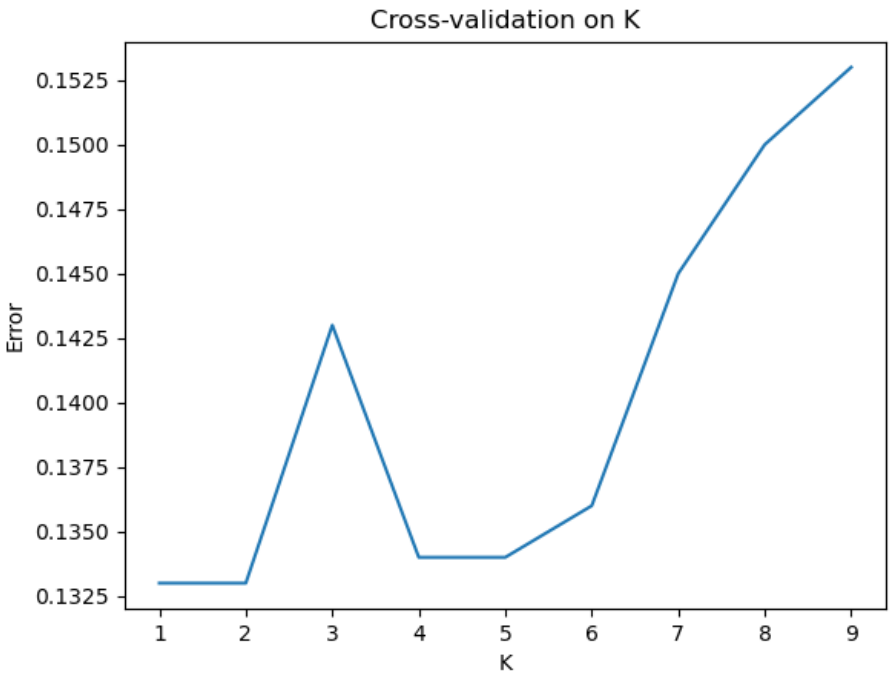
  ```
  f1_score(te_label, test_pred, average='macro')
  ```

- Confusion matrix

  ```
  confusion_matrix(te_label, test_pred)
  ```

5-fold cross validation

```
def select_best_k(trainval_feats, trainval_label, max_k):
    folds = 5
    X_folds = []
    y_folds = []

    errors_of_k = {}
    for k in range(1, max_k + 1):
        errors_of_k[k] = []

    X_folds = np.vsplit(trainval_feats, folds)
    y_folds = np.hsplit(trainval_label, folds)

    for i in range(folds):
        #  prepare train sets and validation sets
        tv_feats = X_folds[i]
        tv_label = y_folds[i]
        tr_feats = np.vstack(X_folds[:i] + X_folds[i + 1:])
        tr_label = np.hstack(y_folds[:i] + y_folds[i + 1:])

        # iterate all possible ks
        for k in range(1, max_k + 1):
            val_pred = []  # record each sample's prediction
            for i in range(tv_feats.shape[0]):
                pred = myknn(tv_feats[i], tr_feats, tr_label, k)
                val_pred.append(pred)
            error_rates = 1 - accuracy_score(tv_label, val_pred)
            errors_of_k[k].append(error_rates)

    '''
        Plot error rates-k figure
    '''
    k_range = np.arange(1, max_k + 1)
    errors_mean = np.array([np.mean(v) for k, v in sorted(errors_of_k.items())])
    plt.plot(k_range, errors_mean)
    plt.title('Cross-validation on K')
    plt.xlabel('K')
    plt.ylabel('Error')
    plt.show()
```



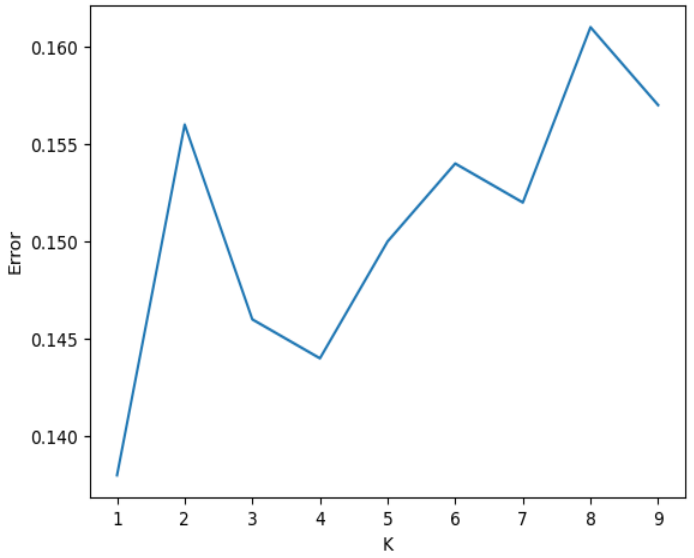
Evaluate KNN

```
def eval_myknn(te_feats, te_label, tr_feats, tr_label, best_k):
    test_pred = []  #
    for i in range(te_feats.shape[0]):
        pred = myknn(te_feats[i], tr_feats, tr_label, best_k)
        test_pred.append(pred)
    f1 = f1_score(te_label, test_pred, average='macro')
    print(f1)
    print(confusion_matrix(te_label, test_pred))
```

```
0.8668193324106703
[[ 938    1    5    1    1    8   12    2    1   11]
 [   0 1126    0    2    0    2    2    1    2    0]
 [  18   39  872   10    5    8   13   49   15    3]
 [   1    7   23  835    0   81    7   17   23   16]
 [   0   20    1    0  791    1   13   17    1  138]
 [  15   13    2   48    9  721   25    9   22   28]
 [  21    5    3    1   13   11  897    0    3    4]
 [   1   36    5    2   14    3    0  913    1   53]
 [  24    8   32   28   20   65   11   22  713   51]
 [   6    3    2   11   43    5    4   48    3  884]]
```

Use functions from sklearn library

```
def sklearn_knn_cv(tr_feats, tr_label, max_k):
    '''
       cross validation on K
    '''
    k_error = []
    folds = 5
    k_range = np.arange(1, max_k + 1)
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, tr_feats, tr_label, cv=folds, scoring='accuracy')
        k_error.append(1 - scores.mean())

    plt.plot(k_range, k_error)
    plt.xlabel('K')
    plt.ylabel('Error')
    plt.show()
```




```
def sklearn_knn_test(te_feats, te_label, tr_feats, tr_label, best_k):
    knn = KNeighborsClassifier(n_neighbors=best_k)
    knn.fit(tr_feats, tr_label)

    prediction = knn.predict(te_feats)
    acc_mean = knn.score(te_feats, te_label)
    print('Test set score:{:2f}'.format(acc_mean))
```

```
Test set score:0.849600
```

Any Questions?