D20 KNN 實作

重要知識點

- 了解如何實踐 KNN 算法，並使用 K-fold 選擇適合 K 值。

補充資料：K-Nearest Neighbors and Bias-Variance Tradeoff

本日實作內容

A. 資料讀取
B. 資料清理
C. 文字轉向量
D. KNN模型導入與參數設置
E. 模型驗證
F. 混淆矩陣 (Confusion matrix)
G. K-fold尋找適合K值

資料讀取

這裡我們將 Spam(垃圾郵件標註為 1)，反之 (Ham) 為 0。

Importing the dataset

- 從三個資料夾中讀取資料

```
In [2]: all_data=[]
paths =[r'spam_data/spam', r'spam_data/easy_ham', r'spam_data/hard_ham']
for path in paths:
    for fn in glob.glob(path+"/*"):
        if "ham" not in fn:
            is_spam = 1
        else:
            is_spam = 0
        #codecs.open可以避開錯誤，用errors='ignore'
        with codecs.open(fn,encoding='utf-8', errors='ignore') as file:
            for line in file:
                #這個line的開頭為Subject:
                if line.startswith("Subject:"):
                    subject=re.sub(r"^Subject:","",line).strip()
                    all_data.append([subject,is_spam])
all_data = np.array(all_data)
```

```
all_data=[]
paths =[r'spam_data/spam', r'spam_data/easy_ham', r'spam_data/hard_ham']
for path in paths:
  for fn in glob.glob(path+"/*"):
    if "ham" not in fn:
      is_spam = 1
    else:
      is_spam = 0
    #codecs.open可以避開錯誤，用errors='ignore'
    with codecs.open(fn,encoding='utf-8', errors='ignore') as file:
      for line in file:
        #這個line的開頭為Subject:
```

資料清理

細節可參考先前文字預處理章節，或是歸納出自己的清理方式也可以。

```
In [6]: from sklearn.metrics import confusion_matrix
from nltk.corpus import stopwords

import nltk

nltk.download('stopwords')

# Lemmatize with POS Tag
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

## 創建Lemmatizer
lemmatizer = WordNetLemmatizer()
def get_wordnet_pos(word):
    """將pos_tag結果mapping到lemmatizer中pos的格式"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)


def clean_content(X):
    # remove non-alphabet characters
    X_clean = [re.sub('[^a-zA-Z]',' ', x).lower() for x in X]
    # tokenize
    X_word_tokenize = [nltk.word_tokenize(x) for x in X_clean]
    # stopwords_lemmatizer
    X_stopwords_lemmatizer = []
    stop_words = set(stopwords.words('english'))
    for content in X_word_tokenize:
        content_clean = []
        for word in content:
            if word not in stop_words:
                word = lemmatizer.lemmatize(word, get_wordnet_pos(word))
                content_clean.append(word)
        X_stopwords_lemmatizer.append(content_clean)

    X_output = [' '.join(x) for x in X_stopwords_lemmatizer]

    return X_output

[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/jeff.yang/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

In [7]: X = clean_content(X)
```

```
from sklearn.metrics import confusion_matrix
from nltk.corpus import stopwords

import nltk

nltk.download ('stopwords')

# Lemmatize with POS Tag
from nltk.corpus import wordnet
from nltk.stem import WordNet Lemmatizer

## 創建Lemmatizer
```

```
def clean_content(X):
  # remove non-alphabet characters
  X_clean = [re.sub ('[a-zA-Z]',' ',x).lower() for x in X]
  # tokenize
  X_word_tokenize = [nltk.word_tokenize(x) for x in X_clean]
  # stopwords_lemmatizer
  X_stopwords_lemmatizer = []
  stop_words = set (stopwords.words ('english'))
  for content in X_word_tokenize :
    content_clean = []
    for word in content:
      if word not in stop_words:
```

```
X = clean_content(X)
```

文字轉向量

這邊使用 BOW 方式，學員們也可以嘗試 TFIDF。

Bag of words

```
In [8]: from sklearn.feature_extraction.text import CountVectorizer
#max_features是要建造幾個column，會按造字出現的頻率高低去篩選，1500並沒有特別含義，大家可以自己嘗試不同數值或不加入限制
cv=CountVectorizer(max_features = 1500)
X=cv.fit_transform(X).toarray()

In [9]: # 有 3423個樣本，每個樣本用1500維表示
X.shape

Out[9]: (3423, 1500)
```

```
from sklearn.feature_extraction.text import CountVectorizer
#max_features是要建造幾個column，會按造字出現的頻率高低去篩選，1500並沒有特別含義，大家可以自己嘗試不同數值
cv=CountVectorizer(max_features = 1500)
x=cv.fit_transform(X).toarray()
```

```
# 有 3423個樣本，每個樣本用1500維表示
X.shape
```

KNN 模型導入與參數設置

- 首先將資料拆解成 trainset/testset (詳細可見參考資料)
- 從 sklearn 中呼叫 KNN classifier API

將資料拆成 train/test set

```
In [10]: from sklearn.model_selection import train_test_split
# random_state 是為了讓各為學員得到相同的結果，平時可以移除
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
```

Training the K-NN model on the Training set

```
In [18]: from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)

Out[18]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
from sklearn.model_selection import train_test_split
# random_state 是為了讓各位學員得到相同的結果，平時可以移除
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

- n_neighbors : K 值，為 hyperparameters(超參數)，可透過 K-fold 協助選擇最適合 K 值。
- metric : 計算樣本距離的方式，默認設置為 Minkowski，其為歐式距離及曼哈頓距離兩種計算距離的延伸。
- P : 為 Minkowski metric 的超參數，p 為 2 時所使用歐式距離。

模型驗證

測試模型在訓練/測試集上的 Accuracy。

測試 train/testset的 Accuracy

```
In [19]: print('Trainset Accuracy: {}'.format(classifier.score(X_train, y_train)))
Trainset Accuracy: 0.91672753834916

In [20]: print('Testset Accuracy: {}'.format(classifier.score(X_test, y_test)))
Testset Accuracy: 0.8875912408759125
```

```
print('Trainset Accuracy: {}" .format(classifier.score(X_train, y_train)))
```

```
print('Testset Accuracy: {}" .format(classifier.score(X_test, y_test)))
```

混淆矩陣 (Confusion matrix)

矩陣對應 TP、FN、FP、TN

Making the Confusion Matrix

```
In [23]: from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

[[581   6]
 [ 71  27]]

Out[23]: 0.8875912408759125
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

		predicted condition	
	total population	prediction positive	prediction negative
true condition	condition positive	True Positive (TP)	False Negative (FN) (type II error)
	condition negative	False Positive (FP) (Type I error)	True Negative (TN)

協助我們更深入的了解正確與錯誤數據的分佈，主要可以分為以下四種

TP(True Positive): 正確預測成功的正樣本，例如在一個預測是不是貓的圖像分類器，成功的把一張貓的照片標示成貓，即為TP

TN(True Negative): 正確預測成功的負樣本，以上述例子，成功的把一張狗的照片標示成不是貓，即為TN

FP(False Positive): 錯誤預測成正樣本，實際上為負樣本，例如：錯誤的把一張狗的照片標示成貓

FN(False Negative): 錯誤預測成負樣本(或者說沒能預測出來的正樣本)，例如：錯誤的把一張貓的照片標示成不是貓

資料來源：混淆矩陣(confusion matrix)介紹

- 我們常用的 Accuracy(正確數量/樣品總數)能傳達的訊息並不夠豐富，常用的 metrics 還包含 Recall/Precision/F1，這些資訊可藉由混淆矩陣進一步推算獲得。
- 這裡我們不詳談其意義，有興趣的學員們可參考參考資料中所附上的連結。

	實際 Positive	實際 Negative
預測 Positive	TP (True Positive)	FP (False Positive)
預測 Negative	FN (False Negative)	TN (True Negative)

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$F_1\text{-score} = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

資料來源：Precision、Recall、F1 三種評估模型的指標

K-fold 尋找適合 K 值

scikit-learn 提供相當方便的 API 直接合併 K-fold 與 Classifier。

運用K-fold尋找適合K值

```
In [13]: # Applying k-Fold Cross Validation
#n-jobs=-1，是指cpu全開
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
n_neighbors = [5, 10, 25, 50, 100] ## 可自行嘗試不同K值
for k in n_neighbors:
    classifier = KNeighborsClassifier(n_neighbors = k, metric = 'minkowski', p = 2)
    # cv = 10 代表切成10等分
    accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10,n_jobs=-1)

    print('設置K值:{}'.format(k))
    print('Average Accuracy: {}'.format(accuracies.mean()))
    print('Accuracy STD: {}'.format(accuracies.std()))

設置K值:5
Average Accuracy: 0.880947378655408
Accuracy STD: 0.011561254412983857
設置K值:10
Average Accuracy: 0.8623193886405567
Accuracy STD: 0.006510165116834898
設置K值:25
Average Accuracy: 0.8528182474459847
Accuracy STD: 0.002483737223283705S
設置K值:50
Average Accuracy: 0.8520856467133839
Accuracy STD: 0.0014068907077920568
設置K值:100
Average Accuracy: 0.8520856467133839
Accuracy STD: 0.0014068907077920568
```

```
# Applying k-Fold Cross Validation
#n-jobs=-1，是指cpu全開
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
n_neighbors = [5, 10, 25, 50, 100] ## 可自行嘗試不同K值
for k in n_neighbors:
  classifier = KNeighborsClassifier(n_neighbors = k, metric = 'minkowski', p = 2)
  # cv = 10 代表切成10等分
  accuracies = cross_val_score(estimator = classifier, X_train, y = y_train, cv = 10,n_jobs=-1)

  pint('設置K值:{}'.format(k))
  print('Average Accuracy: {}'.format(accuracies.mean()))
```

KNN 優缺點整理

優 點	缺 點
1.原理簡單 2.在相對簡單的資料集上都能獲得不錯的精度 3.資料並不需要特別的前處理(沒有統計假設)	1.需要大量運算資源 2.需要較大的記憶體空間 3.不適合用在大型資料集 4.不適合處理高維度資料(Curse of Dimensionality) 5.不同超參數的設置可能會導致結果差異很大

參考資料

trainset/testset/validation set 用意：
網站：訓練集、驗證集、測試集


一文看懂 AI 訓練集、驗證集、測試集（附：分割方法+交叉驗證）

區塊鏈遊戲研究院 · 2019年12月20日20:01

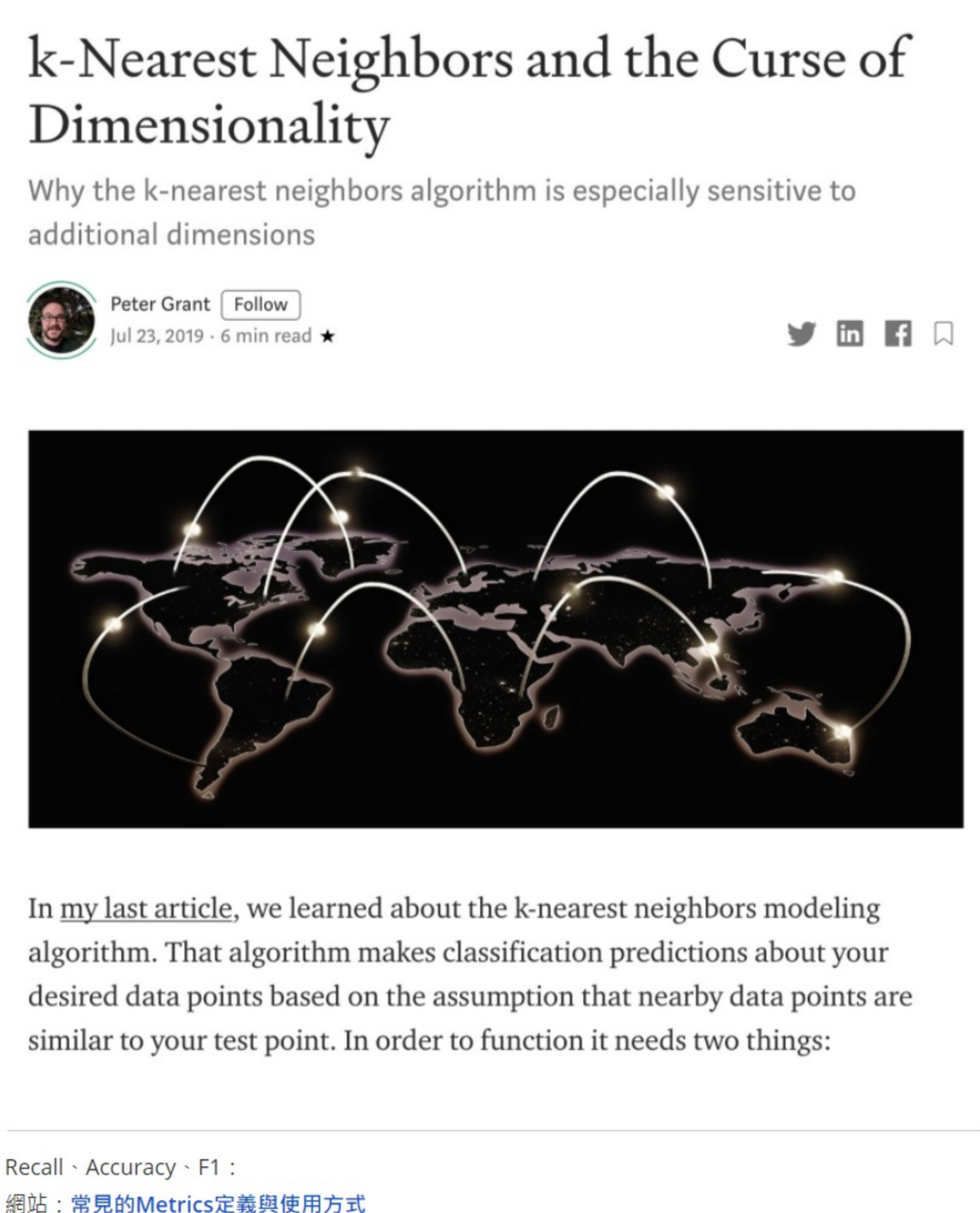
數據在人工智能技術裏是非常重要的！本篇文章將詳細給大家介紹 3 種數據集：訓練集、驗證集、測試集。

同時還會介紹如何更合理的講數據劃分爲 3 種數據集。最後給大家介紹一種充分利用有限數據的方式：交叉驗證法。

先用一個不恰當的比喻來說明 3 種數據集之間的關係：

- 訓練集相當於上課學知識
- 驗證集相當於課後的的練習題，用來糾正和強化學到的知識
- 測試集相當於期末考試，用來最終評估學習效果

高維度資料對模型的影響：
網站：Curse of Dimensionality

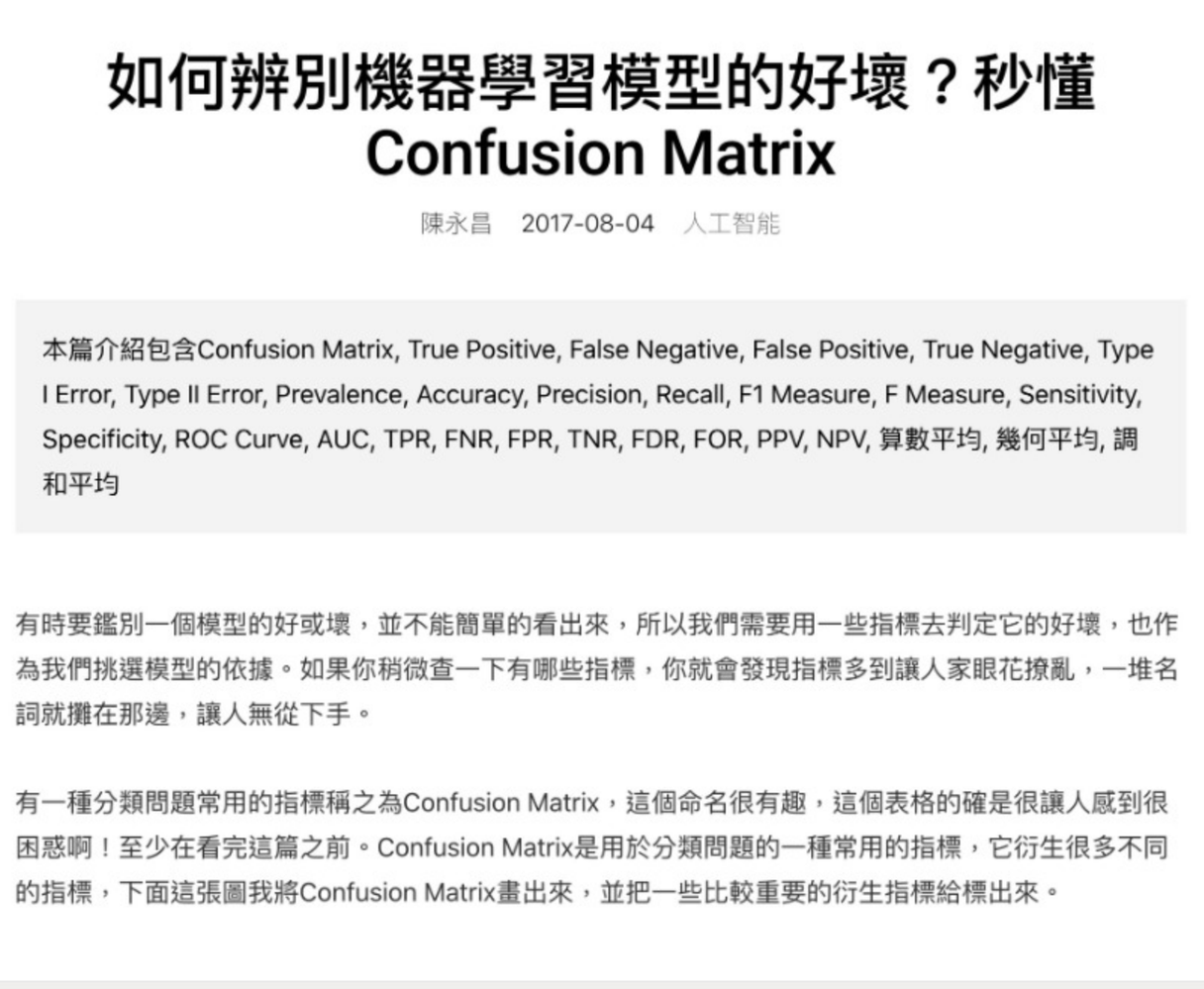

k-Nearest Neighbors and the Curse of Dimensionality

Why the k-nearest neighbors algorithm is especially sensitive to additional dimensions

Peter Grant Follow
Jul 23, 2019 · 6 min read ★

In my last article, we learned about the k-nearest neighbors modeling algorithm. That algorithm makes classification predictions about your desired data points based on the assumption that nearby data points are similar to your test point. In order to function it needs two things:

Recall、Accuracy、F1：
網站：常見的Metrics定義與使用方式


如何辨別機器學習模型的好壞？秒懂 Confusion Matrix

陳永昌 2017-08-04 人工智能

本篇介紹包含Confusion Matrix, True Positive, False Negative, False Positive, True Negative, Type I Error, Type II Error, Prevalence, Accuracy, Precision, Recall, F1 Measure, F Measure, Sensitivity, Specificity, ROC Curve, AUC, TPR, FNR, FPR, TNR, FDR, FOR, PPV, NPV, 算數平均, 幾何平均, 調和平均

有時要鑑別一個模型的好或壞，並不能簡單的看出來，所以我們需要用一些指標去判定它的好壞，也作為我們挑選模型的依據。如果你稍微查一下有哪些指標，你就會發現指標多到讓人家眼花撩亂，一堆名詞就攤在那邊，讓人無從下手。

有一種分類問題常用的指標稱之為Confusion Matrix，這個命名很有趣，這個表格的確是很讓人感到很困惑啊！至少在看完這篇之前。Confusion Matrix是用於分類問題的一種常用的指標，它衍生很多不同的指標，下面這張圖我將Confusion Matrix畫出來，並把一些比較重要的衍生指標給標出來。

下一步：閱讀範例與完成作業