**فصل 15- K نزدیکترین همسایگی**

**15.0 مقدمه**

دسته‌بندی کننده‌ی K-Nearest Neighbors (KNN) یکی از ساده‌ترین و متداول‌ترین دسته‌بندی کننده‌های مورد استفاده در یادگیری ماشین نظارت شده است.

KNN اغلب به عنوان یک یادگیرنده‌ی کند در نظر گرفته می‌شود. این روش از نظر فنی مدلی برای پیش بینی آموزش نمی‌دهد. درعوض، یک مشاهده از کلاس k پیش‌بینی می‌شود به طوری که همان کلاس با بیشترین نسبت k از نزدیک‌ترین مشاهدات باشد. برای مثال، اگر یک مشاهده با یک کلاس ناشناخته توسط یک مشاهده کلاس 1 احاطه شده باشد، سپس مشاهده دسته‌بندی شده به عنوان کلاس 1 است. در این فصل نحوه استفاده از scikit-learn برای ایجاد و استفاده از یک دسته‌بندی کننده‌ی KNN را بررسی خواهیم کرد.

**15.1 یافتن نزدیکترین مشاهده‌ی همسایه‌ها**

**مسئله**

شما باید k نزدیکترین مشاهدات (همسایگان) برای یک مشاهده را پیدا کنید.

**راه‌حل**

از تابع NearestNeighbors در scikit-learn استفاده کنید:

# Load librariesfrom sklearn import datasets  
from sklearn.neighbors import NearestNeighbors  
from sklearn.preprocessing import StandardScaler

# Load datairis = datasets.load\_iris()  
features = iris.data

# Create standardizerstandardizer = StandardScaler()

# Standardize featuresfeatures\_standardized = standardizer.fit\_transform(features)

# Two nearest neighborsnearest\_neighbors =  
NearestNeighbors(n\_neighbors=2).fit(features\_standardized)

# Create an observationnew\_observation = [ 1, 1, 1, 1]

# Find distances and indices of the observation's nearestneighborsdistances, indices =  
nearest\_neighbors.kneighbors([new\_observation])

# View the nearest neighborsfeatures\_standardized[indices]

array([[[ 1.03800476, 0.56925129, 1.10395287, 1.1850097  
],

[ 0.79566902, 0.33784833, 0.76275864,  
1.05353673]]])

**بحث**

در راه حل خود ما از مجموعه داده گل زنبق استفاده کردیم. ما از مشاهدات یک new\_observation، با مقادیری ایجاد کردیم و سپس دو مشاهده‌ای که به مشاهده‌ی ما نزدیکترند پیدا شدند. شاخص‌ها شامل مکان مشاهداتی در مجموعه داده ما هستند که نزدیک‌ترین موارد باشند، بنابراین X[بردار شاخص‌ها] مقادیر آن مشاهدات را نشان می‌دهد. به طور شهودی، فاصله را می‌توان به عنوان معیاری برای تشابه در نظر گرفت، بنابراین دو نزدیک‌ترین مشاهدات دو گل هستند که بیشتر شباهت را به گلی که ما خلق کردیم دارند.

چگونه فاصله را اندازه گیری کنیم؟ scikit-learn طیف گسترده‌ای از معیارهای فاصله، d، را پیشنهاد می‌کند از جمله اقلیدسی:

و فاصله منهتن:

به طور پیش فرض، نزدیکترین همسایگی از فاصله Minkowski استفاده می‌کند:

که در آن و دو مشاهده‌ای هستند که ما فاصله بین آنها را محاسبه می‌کنیم. Minkowski شامل یک هایپرپارامتر p است که در آن p=1 فاصله منهتن و p=2 فاصله اقلیدسی و غیره است. به طور پیش فرض در scikit learn مقدار p=2 استفاده می‌شود.

ما می‌توانیم متریک فاصله را با استفاده از پارامتر metric تنظیم کنیم:

*# Find two nearest neighbors based on euclidean distance*nearestneighbors\_euclidean = NearestNeighbors(  
n\_neighbors=2,  
metric='euclidean').fit(features\_standardized)

ما یک متغیر فاصله‌ ایجاد کردیم که حاوی اندازه گیری فاصله واقعی برای هر یک از دو همسایه نزدیک است:

*# View distances*distances

array([[ 0.48168828, 0.73440155]])

علاوه بر این، می‌توانیم از kneighbors\_graph برای ایجاد یک ماتریس استفاده کنیم.

نشان دادن نزدیکترین همسایگان هر مشاهده:

*# Find each observation's three nearest neighbors  
# based on euclidean distance (including itself)*nearestneighbors\_euclidean = NearestNeighbors(  
n\_neighbors=3,  
metric="euclidean").fit(features\_standardized)

*# List of lists indicating each observation's 3 nearest  
neighbors  
# (including itself)*nearest\_neighbors\_with\_self =  
nearestneighbors\_euclidean.kneighbors\_graph(  
features\_standardized).toarray()

*# Remove 1's marking an observation is a nearest neighbor to  
itself***for** i, x **in** enumerate(nearest\_neighbors\_with\_self):

x[i] = 0

# View first observation's two nearest neighborsnearest\_neighbors\_with\_self[0]

array([ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0.])

زمانی که ما به دنبال نزدیکترین همسایگی هستیم یا از هر نوع الگوریتم یادگیری استفاده می‌کنیم که براساس فاصله هستند، مهم است که ویژگی‌ها را به نحوی تبدیل کنیم که در یک مقیاس باشند. دلیل این موضوع نیز فاصله است. معیارها با همه ویژگی‌ها به گونه‌ای برخورد می‌کنند که گویی در یک مقیاس هستند، اما اگر ویژگی اول به میلیون‌ها دلار و ویژگی دوم به درصد است، فاصله‌ی محاسبه شده نسبت به اولی تمایل پیدا می‌کند و تاثیر اول بیشتر خواهد بود. در راه حل خود، با استانداردسازی ویژگی‌ها با استفاده از مقیاس استاندارد به این موضوع بالقوه پرداختیم.

**15.2 ایجاد یک K-Nearest Neighbor دسته‌بندی**

**مسئله**

با توجه به مشاهده کلاس ناشناخته، باید کلاس آن را بر اساس کلاس همسایگانش پیش بینی کنید.

**راه‌حل**

اگر مجموعه داده خیلی بزرگ نیست، از KNeighborsClassifier استفاده کنید:

# Load librariesfrom sklearn.neighbors import KNeighborsClassifier  
from sklearn.preprocessing import StandardScaler  
from sklearn import datasets

# Load datairis = datasets.load\_iris()  
X = iris.data  
y = iris.target

# Create standardizerstandardizer = StandardScaler()  
# Standardize featuresX\_std = standardizer.fit\_transform(X)

# Train a KNN classifier with 5 neighborsknn = KNeighborsClassifier(n\_neighbors=5,  
n\_jobs=-1).fit(X\_std, y)

*# Create two observations*

new\_observations = [[0.75 , 0.75 , 0.75 , 0.75 ,] ,

[1, 1, 1, 1,]]

*# Predict the class of two observations*knn.predict(new\_observations)

array([1, 2])

**بحث**

در KNN، با توجه به مشاهده‌ی ، با کلاس هدف ناشناخته، الگوریتم، ابتدا تعداد k نزدیکترین مشاهده را بر اساس برخی متریک‌های فاصله (به عنوان مثال، فاصله اقلیدسی) شناسایی می‌کند (که گاهی اوقات همسایگی نامیده می‌شود). سپس این k مشاهده بر اساس کلاس خود رای می‌دهند، و کلاسی که برنده رای می‌شود کلاس پیش بینی شده‌ی است. به طور رسمی تر، احتمالا مقدارکلاس j است:

جایی که v مشاهده‌ی kام در همسایگی است، کلاس iام مشاهده است، و I یک تابع نشانگر است (یعنی 1 درست است و 0 نادرست). در sickit-learn می‌توانیم استفاده از این احتمالات با متغیر predict\_proba ببینیم:

*# View probability each observation is one of three classes*knn.predict\_proba(new\_observations)

array([[ 0. , 0.6, 0.4],  
 [ 0. , 0. , 1. ]])

کلاسی که بیشترین احتمال را دارد به کلاس پیش بینی شده تبدیل می‌شود. به عنوان مثال، در خروجی قبلی، اولین مشاهده باید کلاس 1 باشد(pr=0.6)در حالی که مشاهده‌ی دوم باید کلاس 2 باشد (pr=1)، و این همان چیزی است که می‌بینیم:

knn.predict(new\_observations)

array([1, 2])

دسته‌بندی کننده‌ی KNeighborsClassifier شامل تعدادی از موارد مهم است که باید در نظر گرفته شوند. ابتدا metric، که فاصله‌ی استفاده شده را تنظیم می‌کند (در دستور العمل 14.1 بحث شده است). دوم، n\_jobs تعیین می‌کند که چگونه بسیاری از هسته‌های کامپیوتر استفاده میشوند. چون محاسبه‌ی فاصله از یک نقطه تا هر نقطه در داده‌ها را پیش بینی می‌کند، استفاده از چندین هسته بسیار توصیه می‌شود. سوم، algorithm، روش مورد استفاده برای محاسبه‌ی نزدیکترین همسایگی را تنظیم می‌کند. در حالیکه تفاوت‌های واقعی در الگوریتم‌ها وجود دارد. به طور پیش فرض دسته‌بندی‌کننده‌ی KNeighborsClassifier تلاش می‌کند تا بهترین‌ها را به صورت خودکار انتخاب کند. بنابراین شما اغلب نیازی به نگرانی در مورد این پارامتر ندارید. چهارم، KNeighborsClassifier به طور پیش فرض با مشاهده‌ی هر همسایگی، کسب یک رای می‌کند؛ با این حال، اگر پارامتر وزن‌ها را روی آن قرار دهیم فاصله‌ی مشاهدات نزدیکتر به طور شهودی وزن بیشتری نسبت به مشاهدات دورتر دارندکه این یک امر منطقی است، زیرا همسایگان مشابه بیشتر ممکن است در اطراف کلاس مشاهده به ما نزدیک باشند تا دیگران. در نهایت، از آنجایی که محاسبات فاصله با تمام ویژگی‌ها به گونه‌ای برخورد می کند که گویی در یک مقیاس هستند، استانداردسازی ویژگی‌ها قبل از استفاده از طبقه بندی کننده KNN مهم است.

**15.3 شناسایی بهترین اندازه‌ی همسایگی**

**مسئله**

شما می‌خواهید بهترین مقدار را برای k در دسته‌بندی‌کننده‌ی نزدیکترین همسایه‌ی k(k-nearest neighbors) انتخاب کنید.

**راه حل**

از تکنیک‌های انتخاب مدل مانند GridSearchCV استفاده کنید:

*# Load libraries***from sklearn.neighbors import** KNeighborsClassifier  
**from sklearn import** datasets  
**from sklearn.preprocessing import** StandardScaler  
**from sklearn.pipeline import** Pipeline, FeatureUnion  
**from sklearn.model\_selection import** GridSearchCV

*# Load data*iris = datasets.load\_iris()  
features = iris.data  
target = iris.target

*# Create standardizer*standardizer = StandardScaler()  
*# Standardize features*features\_standardized = standardizer.fit\_transform(features)

*# Create a KNN classifier*knn = KNeighborsClassifier(n\_neighbors=5, n\_jobs=-1)  
*# Create a pipeline*pipe = Pipeline([("standardizer", standardizer), ("knn",  
knn)])

*# Create space of candidate values*search\_space = [{"knn\_\_n\_neighbors": [1, 2, 3, 4, 5, 6, 7,  
8, 9, 10]}]

*# Create grid search*classifier = GridSearchCV(  
pipe, search\_space, cv=5,  
verbose=0).fit(features\_standardized, target)

**بحث**

اندازه‌ی k تاثیراتی واقعی در دسته‌بندی کننده‌های KNN دارد. در یادگیری ماشین ما بدنبال یافتن تعادل بین بایاس(سوگیری) و واریانس هستیم، و در چند جا اندازه‌ی مقدار k واضح است. اگر k = n باشد که در آن n تعداد مشاهدات است، پس سوگیری زیاد داریم اما واریانس کم. اگر k=1 باشد، بایاس کم ولی واریانس زیادی خواهیم داشت. بهترین مدل از یافتن مقداری برای k که این مبادله‌ی بایاس واریانس را متعادل می‌کند به دست می‌آید. در راه حل خود، ما از GridSearchCV برای انجام اعتبارسنجی(Cross-Validation) متقابل پنج برابری بر روی طبقه‌بندی‌کننده‌های KNN با مقادیر مختلف k استفاده کردیم. وقتی این کار تکمیل شد، می توانیم k را ببینیم که بهترین مدل را تولید می کند:

*#Bestneighborhoodsize(k)*classifier.best\_estimator\_.get\_params()["knn\_\_n\_neighbors"]

6

**15.4 ایجاد نزدیکترین دسته‌بندی کننده همسایه بر اساس شعاع**

**مسئله**

با توجه به مشاهده‌ی کلاس ناشناخته، باید کلاس آن را بر اساس کلاس همه‌ی مشاهدات در فاصله معین پیش بینی کنید.

**راه حل**

از RadiusNeighborsClassifier استفاده کنید:

*# Load libraries***from sklearn.neighbors import** RadiusNeighborsClassifier  
**from sklearn.preprocessing import** StandardScaler  
**from sklearn import** datasets

*# Load data*iris = datasets.load\_iris()  
features = iris.data  
target = iris.target  
*# Create standardizer*standardizer = StandardScaler()

*# Standardize features*features\_standardized = standardizer.fit\_transform(features)

*# Train a radius neighbors classifier*rnn = RadiusNeighborsClassifier(  
radius=.5, n\_jobs=-1).fit(features\_standardized, target)

*# Create two observations*new\_observations = [[ 1, 1, 1, 1]]

*# Predict the class of two observations*rnn.predict(new\_observations)

array([2])

**بحث**

در طبقه‌بند‌ی‌کننده‌ی KNN، کلاس مشاهده از کلاس‌های همسایه‌ی k پیش‌بینی می‌شود. یک تکنیک کمتر رایج، طبقه‌بندی در طبقه‌بندی‌کننده مبتنی بر شعاع (RNN) است، که در آن کلاس مشاهده از کلاس‌های همه‌ی مشاهدات در یک شعاع مشخص r پیش‌بینی می‌شود. طبقه‌بندی‌کننده‌ی RadiusNeighborsClassifier در scikit-learn بسیار شبیه به KNeighborsClassifier است، به استثنای دو پارامتر. ابتدا، در RadiusNeighborsClassifier باید شعاع ناحیه ثابت مورد استفاده برای تعیین اینکه آیا یک مشاهده همسایه ای است که از شعاع استفاده می کند را مشخص کنیم. غیر از اینکه دلیل اساسی برای تنظیم شعاع روی مقداری وجود داشته باشد، بهتر است آن را مانند هر هایپرپارامتر دیگری در نظر بگیریم و در طول انتخاب مدل تنظیم کنیم. دومین پارامتر مفید outlier\_label است که نشان می‌دهد چه برچسبی باید به مشاهده‌ای داد که هیچ مشاهداتی در شعاع مورد نظر ندارد - که خود اغلب می‌تواند ابزار مفیدی برای شناسایی نقاط پرت باشد.

**15.5 پیدا کردن نزدیکترین همسایه‌های تقریبی (ANN)**

**مسئله**

می‌خواهید نزدیک‌ترین همسایگان را برای داده‌های بزرگ با تأخیر کم را واکِشی[[1]](#footnote-1) کنید:

**راه‌حل**

از جستجوی تقریبی نزدیکترین همسایگان (ANN) با کتابخانه faiss فیس بوک استفاده کنید:

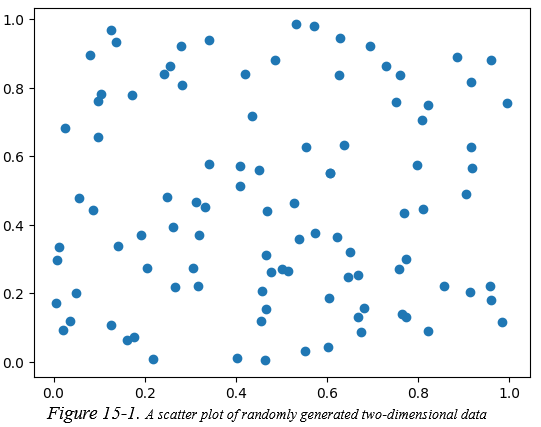


**بحث**

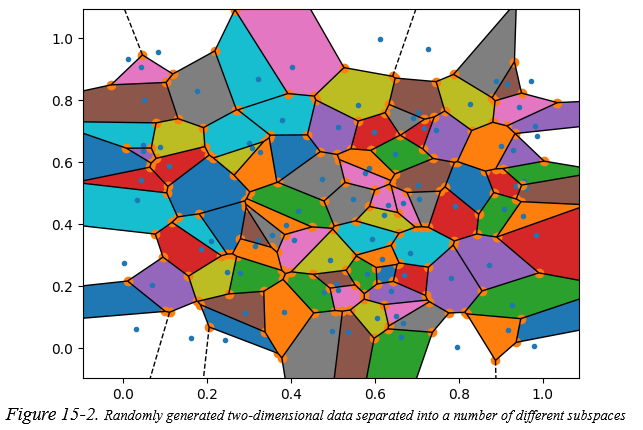
KNN یک رویکرد عالی برای یافتن مشابه ترین مشاهدات در مجموعه ای از داده‌های کوچک است. با این حال، با افزایش اندازه داده‌های ما، زمان محاسبه فاصله بین هر مشاهده و سایر نقاط مجموعه داده‌ی ما نیز افزایش می‌یابد. سیستم‌های ML در مقیاس بزرگ مانند موتورهای جستجو یا توصیه، اغلب از نوعی معیار تشابه برداری برای بازیابی مشاهدات مشابه استفاده می‌کنند. اما در مقیاس بزرگ در زمان واقعی، جایی که ما به نتایج کمتر از 100 میلی‌ثانیه نیاز داریم، اجرای KNN غیرممکن می‌شود.

ANN به ما کمک می‌کند تا با قربانی کردن مقداری از کیفیتِ جستجوی دقیقِ نزدیکترین همسایگان به نفع سرعت، بر این مشکل غلبه کنیم. این بدان معناست که اگرچه ترتیب و موارد در 10 همسایه اول یک جستجوی ANN ممکن است با 10 نتیجه اول از یک جستجوی دقیق KNN مطابقت نداشته باشد، ما آن 10 همسایه اول را بسیار سریعتر دریافت می‌کنیم.

در این مثال، ما از یک رویکرد ANN به نام شاخص فایل معکوس (IVF) استفاده می‌کنیم. این رویکرد با استفاده از خوشه بندی برای محدود کردن دامنه فضای جستجو برای جستجوی نزدیکترین همسایگان ما کار می‌کند. IVF از تسسلات[[2]](#footnote-2) Voronoi برای تقسیم فضای جستجوی ما به تعدادی منطقه (یا خوشه) مجزا استفاده می‌کند. و وقتی برای یافتن نزدیک‌ترین همسایگان می‌رویم، از تعداد محدودی از خوشه‌ها بازدید می‌کنیم تا مشاهدات مشابهی را پیدا کنیم، برخلاف مقایسه بین هر نقطه از مجموعه داده‌هایمان با همدیگر.

نحوه ایجاد مجموعه‌های Voronoi از داده‌ها به بهترین وجه با استفاده از داده‌های ساده، قابل مشاهده است. همانطور که در شکل 15-1 نشان داده شده است، یک نمودار پراکنده از داده‌های تصادفی که در دو بعد مشاهده می‌شود، در نظر بگیرید.

با استفاده از Tessellations Voronoi، می‌توانیم تعدادی زیرفضا ایجاد کنیم که هر کدام از آنها فقط شامل یک زیرمجموعه کوچک از کل مشاهداتی است که می‌خواهیم در آن جستجو کنیم، همانطور که در شکل 15-2 نشان داده شده است.



پارامتر nlist در کتابخانه Faiss به ما امکان می‌دهد تعداد خوشه‌هایی را که می‌خواهیم ایجاد کنیم تعریف کنیم. یک پارامتر اضافی، nprobe، می‌تواند در زمان پرس و جو برای تعریف تعداد خوشه‌هایی که می‌خواهیم جستجو کنیم تا نزدیک‌ترین همسایه‌ها را برای یک مشاهده مشخص بازیابی کنیم، استفاده شود. افزایش هر دو nlist و nprobe می‌تواند منجر به همسایگان با کیفیت بالاتر به قیمت تلاش محاسباتی بزرگتر و در نتیجه زمان اجرای طولانی تر برای شاخص‌های IVF شود. کاهش هر یک از این پارامترها، اثر معکوس خواهد داشت و کد شما سریعتر اجرا می‌شود اما همچنین خطر بازگشت نتایج با کیفیت پایین تر را خواهید داشت.

توجه داشته باشید که این مثال دقیقا همان خروجی دستور اول در این فصل را برمی‌گرداند. این به این دلیل است که ما با داده‌های بسیار کوچک کار می‌کنیم و فقط از سه خوشه استفاده می‌کنیم، که باعث می‌شود نتایج ANN ما به طور قابل‌توجهی با نتایج KNN ما متفاوت نباشد.

**همچنین ببینید:**

* [نزدیکترین نمایه‌های همسایه برای جستجوی شباهت (انواع شاخص ANN مختلف)](https://www.pinecone.io/learn/series/faiss/vector-indexes/)

**15.6 ارزیابی تقریبی نزدیکترین همسایگان**

**مسئله**

می‌خواهید ببینید ANN شما چگونه با نزدیک‌ترین همسایگان (KNN) مقایسه می‌شود:

**راه‌حل**

فراخوان[[3]](#footnote-3) @k نزدیکترین همسایه ANN را در مقایسه با KNN محاسبه کنید:



**بحث**

Recall @k به سادگی به عنوان تعداد آیتم‌هایی است که توسط ANN در برخی از k نزدیک‌ترین همسایه‌هایی که در نزدیک‌ترین همسایه‌ها در همان k، تقسیم بر k ظاهر می‌شوند؛ بازگردانده می‌شود. در این مثال، در 10 نزدیکترین همسایه، ما 100% فراخوانی داریم، به این معنی که ANN ما، همان شاخص های KNN ما را در k=10 برمی گرداند (البته نه لزوماً به همان ترتیب).

یادآوری یا فراخوان[[4]](#footnote-4) یک معیار رایج برای استفاده در هنگام ارزیابی شبکه های عصبی مصنوعی در برابر نزدیکترین همسایگان است.

**همچنین ببینید:**

* [یادداشت Google در ANN برای سرویس موتور تطبیق Vertex](https://cloud.google.com/vertex-ai/docs/vector-search/overview#why_does_ann_perform_approximate_matches_instead_of_exact_matches)

1. - fetch [↑](#footnote-ref-1)
2. - tessellations [↑](#footnote-ref-2)
3. - recall [↑](#footnote-ref-3)
4. - recall [↑](#footnote-ref-4)