Klassifikation ved brug K-Nærmeste-Naboer algoritmen

Af Henrik Sterner (henrik.sterner@gmail.com)

Nærværende dokument beskriver et projekt, der vedrører en af de mest simple maskinelæringsalgoritmer kaldet K-Nærmeste-Naboer eller bare KNN. Selvom den er relativt simpel, giver den et super godt indblik i hvad maskinelæring er og hvad det kan bruges til.

Har du probler med at læse formlerne i dette dokument kan du også hente dokumentet i pdf.

Del I: Generering af data og 1-NN i 2D

I det følgende er målet at implementere en visuel 1-NN, der altså klassificerer et nyt punkt efter sin nærmeste nabo.

Start med at lave en funktion, der givet n som input genererer n rækker af tre kolonner. De to første kolonner skal være x og y koordinater og den tredje kolonne skal være en label også i form af et tal. Til at starte med kan I nøjes med to labels (0 for rød 1 for blå). Herunder formen for listen med koordinater i tabelformat:

x	y	label
4	2	0
3	3	1
2	4	0
..	..	..

Prøv at lave en funktion, der genererer en liste med koordinater og labels og returnerer et numpy array. Herunder et skelet til en sådan funktion:

```
def generateData(n):
    # skriv din kode her - n er antal rækker
    return(np.array(data))
```

1. Lav en liste af disse punkter og visualiser dem på skærmen med forskellige farver
2. Generer nu et nyt punkt p, som du kender placering på men ikke kender label/farve på.
3. Implementer en funktion 1NN, der finder det nærmest punkt og farvelæg p udfra samme farve som nærmeste nabo
4. Udvid programmet, så det ikke kun består af to farver men et vilkårligt antal.
5. Udvid programmet, så det kan håndtere max og min værdier for x og y. Dvs. at punkterne ikke kan ligge udenfor et bestemt område.

Del 2: k-NN i 2D

Lav nu en ny funktion kNN som implementerer kNN-algoritmen, som tager et ulige K (!!) og et punkt p uden label. Antag til at starte med, at et punkt kun kan have to forskellige labels. Følgende skal gøres:
1. Beregn den euklidiske distance fra alle andre punkter til p 2. Sorter listen ud fra distancen til p 3. Udvælg de K første punkter fra denne liste 4. Lav en afstemning dvs. find ud af hvilken label der er flest af blandt de K udtagne punkter. 5. Farvelæg p ud fra dette 6. Udvid koden så den nu også kan inddrage punkter med flere forskellige labels. 7. Udvælg et virkeligt datasæt bestående af to kolonne med feauteres og 1 kolonne med mulighed for flere labels. Indlæs og afprøv din KNN algoritme på casen. 8. Konstruer en visualisering af valg af K. Hvor du har K ud af x-aksen og antal korrekte klassificeringer op ad y-aksen. Prøv den af i praksis.

Del 3: Forskellige distance metrikker

Indtil nu har vi kun gjort brug af den euklidiske distance, som er defineret ved

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Hvor x og y er to vektorer med n dimensioner.

Der findes mange andre distancemetrikker. En af dem er Manhattan distancen, som er defineret ved

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Et tredje eksempel er Minkowski distancen, som er en generalisering af de to ovenstående. Den er defineret ved:

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

Hvor p er en parameter. Når $p = 1$ får vi Manhattan distancen og når $p = 2$ får vi euklidisk distancen.

En anden metrik er Cosinus afstanden, som er defineret ved:

$$d(x, y) = 1 - \frac{x \cdot y}{||x|| \cdot ||y||}$$

1. Implementer de forskellige distancemetrikker i Python som funktioner. Herunder er den euklidiske distance givet som et eksempel.

```
def euclidDistance(x1,x2):
    return(np.sqrt(np.sum((x1-x2)**2)))
```

2. Afprøv de forskellige distancemetrikker på et datasæt og sammenlign resultaterne.

Del 4: Implementer kNN i numpy

I denne øvelse skal du prøve at implementere kNN i numpy, hvis du har mod på det (dvs. helt frivillig men hvis du har tid til overs så kig på den). Hvis du synes det er for svært, så prøv at forstå følgende kode og prøv den af i praksis:

```
# Function computing Euclidean distance
def euclidDistance(x1,x2):
    return(np.sqrt(np.sum((x1-x2)**2)))
    # Function to calculate KNN
def prediction(xTrain,y,xInput,K):
    predLabels=[]
    for x in xInput:
        distances=[] # to store distances
        for i in range(len(xTrain)):
            distances.append(euclidDistance(np.array(xTrain[i,:]),x))
        distances=np.array(distances)
        d = np.argsort(distances)[:K] # k first points
        labels = y[d]
        predLabels.append(stats.mode(labels)[0])
    return predLabels
```

Den kræver at du har datasættet liggende. Tjek mappen på github kaldet data.

Del 5: kNN i praksis

Anvend kNN på mindst tre forskellige cases. For hver case skal der 1. Udvælge relevante træningsdata og testdata, som I indlæser. 2. Separere features og labels for træningsdata og tilsvarende for testdata. 3. Afprøves KNN med mindst to forskellige distancemetrikker
4. Plottes i matplotlib en graf med k fra 2 til 50 ud af x-aksen og den respektive accuracy score opad y-aksen for den respektive k-værdi. 5. Argumenter for valg af den optimale K. 6. Overvej hvorledes man kunne lave en vægtet KNN. Dvs. hvor man tager højde for at datapunkterne kan have forskellige vægte (i 2D kunne det illustreres med radier). Prøv at implementer det i praksis. (Frivillig opgave)

Eksempler på cases kan være mange. Man kan hente dem eksempelvis på kaggle.com. Hvis man ikke selv gider støve data op, så er det muligt at bruge datasæt fra scikit learn. Herunder en liste over nogle af de mest brugte i sci-kit:

- Iris. Iris er et klassisk datasæt, som bruges til at illustrere maskinelæring. Det består af 150 rækker og 4 kolonner. De fire kolonner er længden og bredden af kronblad og sepal. Der er tre forskellige labels, som er tre forskellige typer af iris blomster.
- Digits. Digits er et datasæt med 1797 rækker og 64 kolonner. De 64 kolonner er pixelværdierne for et 8x8 billede af en håndskrevet tal. Der er 10 forskellige labels, som er de 10 tal fra 0 til 9.

- Wine. Wine er et datasæt med 178 rækker og 13 kolonner. De 13 kolonner er forskellige kemiske egenskaber vedrørende vin. Der er tre forskellige labels, som er tre forskellige typer af vin.
- Breast cancer. Breast cancer er et datasæt med 569 rækker og 30 kolonner. De 30 kolonner er forskellige mål vedrørende brystkræft. Der er to forskellige labels, som er henholdsvis godartet og ondartet brystkræft.
- Boston. Boston er et datasæt med 506 rækker og 13 kolonner. De 13 kolonner er forskellige mål vedrørende boligpriser i Boston. Der er kun én label, som er boligprisen.
- Diabetes. Diabetes er et datasæt med 442 rækker og 10 kolonner. De 10 kolonner er forskellige mål vedrørende diabetes. Der er kun én label, som er en kvantitativ måling af sygdommens progression et år efter baseline.

Herunder vises hvorledes iris kan indlæses og bruges:

```
from sklearn.datasets import load_iris
iris = load_iris()
# Storing the data and labels into "X" and "y" varaibles
X = iris.data
y = iris.target
```

I må gerne bruge den indbyggede knn - se nedenfor. I vælger selv cases men I kan lade jer inspirere af de datasæt i mine slides.

Herunder et eksempel på hvorledes man kan indlæse data og afvikle knn først med scikits knn og bagefter med knn ovenfor.

```
Endelig også et eksempel hvor I bruger indbyggede datasæt (iris):
```python
from sklearn.datasets import load_iris
iris = load_iris()
# Storing the data and labels into "X" and "y" varaibles
X = iris.data
y = iris.target

from sklearn.model_selection import train_test_split#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
from sklearn.neighbors import KNeighborsClassifier

# We "assumed" k(the number of neighbors i.e. n_neighbors)  = 3. It can also be 5, 7 … 10
model = KNeighborsClassifier(n_neighbors=3)
# Training or fitting the model with the train data
model.fit(X_train,y_train)

model.predict(X_test)
model.score(X_test,y_test)
```

Herunder lidt om features og labels for iris-sættet.
```

Features: * sepal length in cm * sepal width in cm * petal length in cm * petal width in cm

Labels: * “0”: setosa * “1”: versicolor * “2”: virginica

Del 6: Finde den optimale K-værdi

En vigtig del af KNN er at finde den optimale K-værdi. Dette kan gøres ved at plotte K-værdierne mod deres respektive scores. Dette kan gøres ved at bruge matplotlib. Herunder ses et eksempel på hvordan dette kan gøres.

```
import matplotlib.pyplot as plt
# Making a list of K for KNN
k_list = list(range(1,50,2))
# Creating empty list for scores
scores = []
# Looping over different values of k for KNN
for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    #Appending scores to empty list
    scores.append(knn.score(X_test,y_test))
# Plotting the results
plt.plot(k_list,scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
plt.title('Accuracy scores for values of K of k-Nearest-Neighbors')
plt.show()
```

Prøv at afprøve dette på jeres data. Gerne alle jeres cases.

Del 7: Andre metoder til at finde den optimale K-værdi

Der findes også andre metoder til at finde den optimale K-værdi. En af dem er GridSearchCV, som er en del af scikit-learn. Ideen med GridSearchCV er at lave et grid af forskellige hyperparametre og så finde den kombination af hyperparametre, som giver den bedste score.

Herunder ses et eksempel på hvordan dette kan gøres.

```
from sklearn.model_selection import GridSearchCV
# Making a list of K for KNN
k_list = list(range(1,50,2))
# Creating a dictionary with hyperparameters and possible values
param_grid = dict(n_neighbors=k_list)
# Instantiating the model
knn = KNeighborsClassifier()
# Instantiating the grid
```

```
grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')
# Fitting the model
grid.fit(X,y)
# Printing the optimal score and hyperparameters
print(grid.best_score_)
print(grid.best_params_)
print(grid.best_estimator_)
```

Prøv at afprøve dette på jeres data. Gerne alle jeres cases.

Del 8: Præsentation af resultater

Afslutningsvis skal I præsentere jeres resultater i plenum.