Clustering

Praktisk Maskinlæring Oblig 2, skrevet av Stian Larsen

I denne delen skal jeg bruke maskinlæring for å avgjøre hvilken klasse en blomst tilhører basert på lengde, bredde, tykkelse o.l på blomstene. Dette gjør jeg ved å benytte datasettet «iris» som er bygget inn i et bibliotek i Python.

Datasettet består av 4 kolonner som er aktuelle for å spå typen («species») på blomsten. Dette avgjøres basert på sammenheng mellom de 4 kolonnene. Hele datasettet består av 5 kolonner totalt. Hvor den siste er species, altså den vi skal spå.

Datasettet er som følger:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

(Merk av kolonne 5 som er «species» ikke er tatt med her.)

Jeg ville også vanligvis sjekke tom noen rader inneholder noe annet enn hva som er forespurt. Feks om noen ikke inneholder tall, som feks bokstaver i steden osv. Dette gjøres enkelt ved å kjøre en funksjon som heter isna(), denne returnerer True om noen verdier ikke er tall. Jeg kan se i mitt sett nå at den returnerer false i alle radene:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False
..	...	...	...	...
145	False	False	False	False
146	False	False	False	False
147	False	False	False	False
148	False	False	False	False
149	False	False	False	False

[150 rows x 4 columns]

(Nå vet jeg på forhånd at dette datasettet er «perfekt...»)

Ved å kjøre describe() funksjonen på datasettet, kan vi se nyttig informasjon for oss:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Bildet over viser oss informasjon om alle verdiene vi har oversikt over fra radene. Vi ser at verdiene ikke er normaliserte, og vi ønsker som regel ikke å jobbe med store tall, derfor velger jeg å normalisere verdiene slik at vi får verdier mellom 0 og 1, slik:

```
iris_data = ((iris_data - iris_data.min()) / (iris_data.max() - iris_data.min()))
```

Deretter kan vi se dataene vi har:

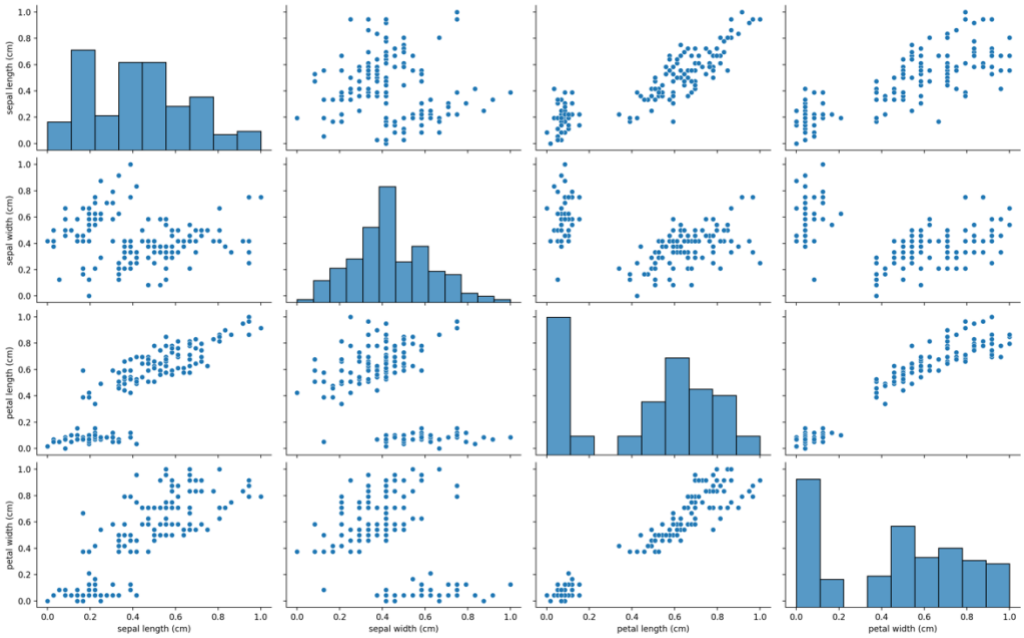
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	0.428704	0.440556	0.467458	0.458056
std	0.230018	0.181611	0.299203	0.317599
min	0.000000	0.000000	0.000000	0.000000
25%	0.222222	0.333333	0.101695	0.083333
50%	0.416667	0.416667	0.567797	0.500000
75%	0.583333	0.541667	0.694915	0.708333
max	1.000000	1.000000	1.000000	1.000000

(Nå normaliserte!)

Exploratory Data Analysis

For å analysere datasettet brukte jeg seaborn, og plottet dettet med matplotlib.

Sjekke analysen under:



På bildet av datasettet kan vi utforske korrelasjonen mellom alle kolonnevalgene vi har. Dette gjør at vi visuelt kan se litt sammenhenger mellom de forskjellige features´ene.

Modeller

Modellene jeg har tenkt til å bruke i denne deler er som følger:

1. K-means
2. Agglomerative
3. Birch

K-means

K-means is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid.

Denne algoritmen er centroid-basert. Dvs at den kalkulerer distansen for å tilknytte et plot/punkt til en cluster. Hver cluster tilhører en centroid, og vi kan derfor separere utfallene basert på denne beregningen.

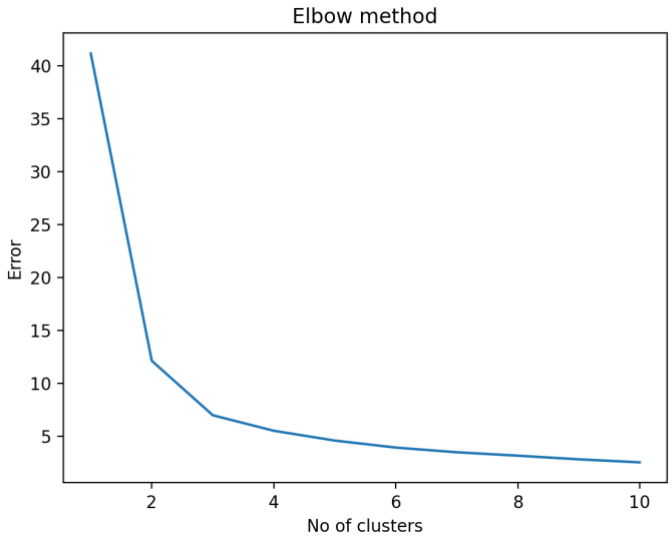
Fremgangsmåte

Jeg startet her med å finne ut hvor mange n clusters jeg skulle benytte. Her tok jeg i bruk Elbow metoden, som er laget for å finne det optimale antallet clusters i et datasett. Vi skal med andre ord bruke dette til å finne den optimale verdien k.

Ide kom fra her:
https://heartbeat.comet.ml/k-means-clustering-using-sklearn-and-python-4a054d67b187

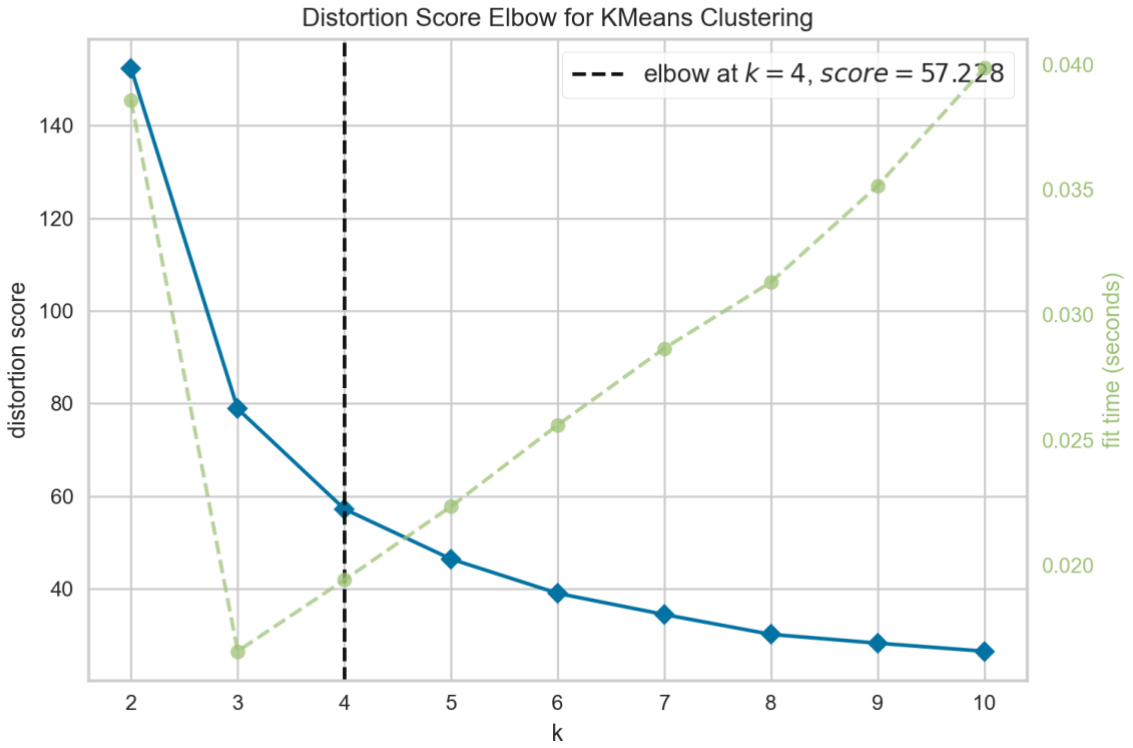
Kode:

```
Error =[]
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i).fit(iris_data)
    kmeans.fit(iris_data)
    Error.append(kmeans.inertia_)

plt.plot(range(1, 11), Error)
plt.title('Elbow method')
plt.xlabel('No of clusters')
plt.ylabel('Error')
plt.show()
```



(Visuelt kan vi se at det beste cluster-nummeret vil være mellom 2 og 4, men ejg vil gjerne utforske dette litt mer.)

Etter litt analysering fant jeg et annet bibliotek som virker bedre egnet for å regne ut hvilken k vi skal bruke videre. Biblioteket heter «Yellowbrick», og funksjonen/metoden heter KElbowVisualizer. Ved å initialisere denne får vi følgende visualisering:



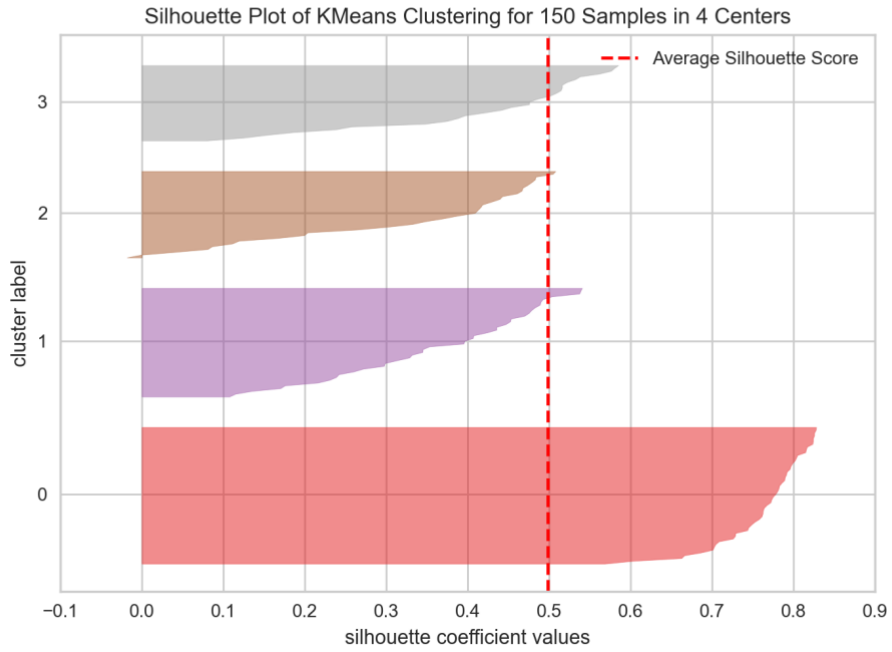
Vi kan fra bildet se at det er mye mer detaljert enn førstnevnte fremgangsmåte. Vi får et forslag på anbefaling av hva «k» burde være, som er k = 4.

Etter min mening vil ikke dette være det perfekte utgangspunktet, da vi vet at antall clusters egt skal være totalt 3. Dette vet vi etter å ha jobbet med datasettet tidligere, hvor species = 3.

Ved å teste ut en annen visualisering fra yellowbricks cluster-bibliotek kan vi sjekke om det vil være oppriktig å bruke k = 4.

Jeg tester da ut SilhouetteVisualiser, og sjekker hva resultatet av dette blir:

Resultatet er som følger:

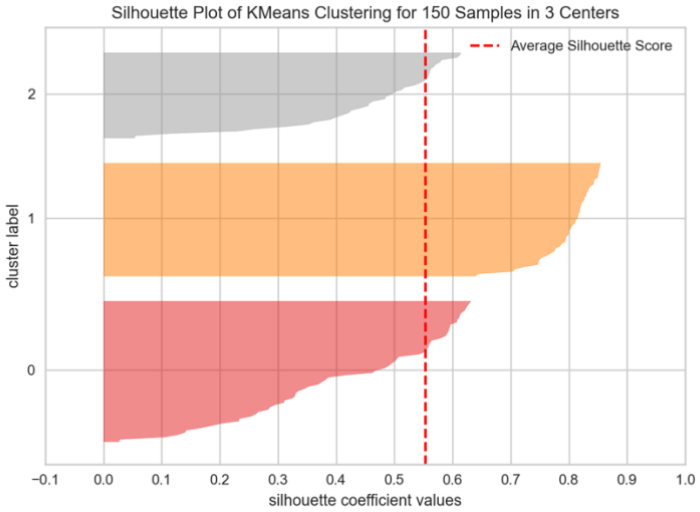


Koden displayer et silhouette plot basert på koden:

```
# tester ut k = 4 som anbefalt:
solhouette_model = KMeans(n_clusters = 4, random_state=42)

sil_visualizer = SilhouetteVisualizer(solhouette_model)
sil_visualizer.fit(iris_data)
sil_visualizer.show()
```

Vi kan se at cluster nr 0 er svært høy og nærmere 1 enn noen av de andre. Det er et gjennomsnitt på 0.5 hos de gjenværende 1, 2 og 3. Og jeg vil nå neste ut å sjekke hvordan modellen reagerer basert på at jeg vet a k = 3, er riktig.

Tester ut ny vending:



Og voila! Her kan vi se at modellen gjør et lite løft. Alle har en bedre score gjennomsnittlig, og det er mindre rom mellom de alle.

Basert på bibliotekene, som først viste Elbow oss anbefaling på 2-4, så testet vi ut KElbowVisualizer og fikk k = 4, dette ga oss en pekepinne visuelt. Vi kjørte silhouette visualizer som kunne vise oss et gap mellom 0 og de andre 3 gjenværende.

Til slutt testet vi ut med k = 3 som vi jo vet er riktig, og fikk da et bedre resultat som jeg er ganske fornøyd med. Nå gjenstår det å få en prediction score bare, og vi er good med denne modellen.

Etter dette er kan plotte silhouette scoren, og den er som følger:

```
K-Means Silhouette score:
 0.5528190123564102
```

Vi kan se at silhouette scoren er over 0.5, noe som er bra i og for seg. Scoren går fra 0 – 1, og den sier noe om hvor godt separert hver cluster er. En score på 1 vil si at de er veldig godt separert og ingen overlapping, imens en score på 0 vil tilsi at clusterne overlapper og det er lite som skiller de fra hverandre. Vi ligger sånn litt over midten på 0.55, og det betyr i all hovedsak at vi har de separert sånn någenlunde greit. Modellen klarer å separere de fra hverandre, uten noe særlig overlapp.

Så er det Rand score fra biblioteket fra Sklearn, denne returnerer en Raw Index, som er en score basert på antall "agreeing paird" / (delt på) antall par som eksisterer.

Resultatet blir slik:

```
K-Means Rand score:
 0.8705882352941177
```

Kode:

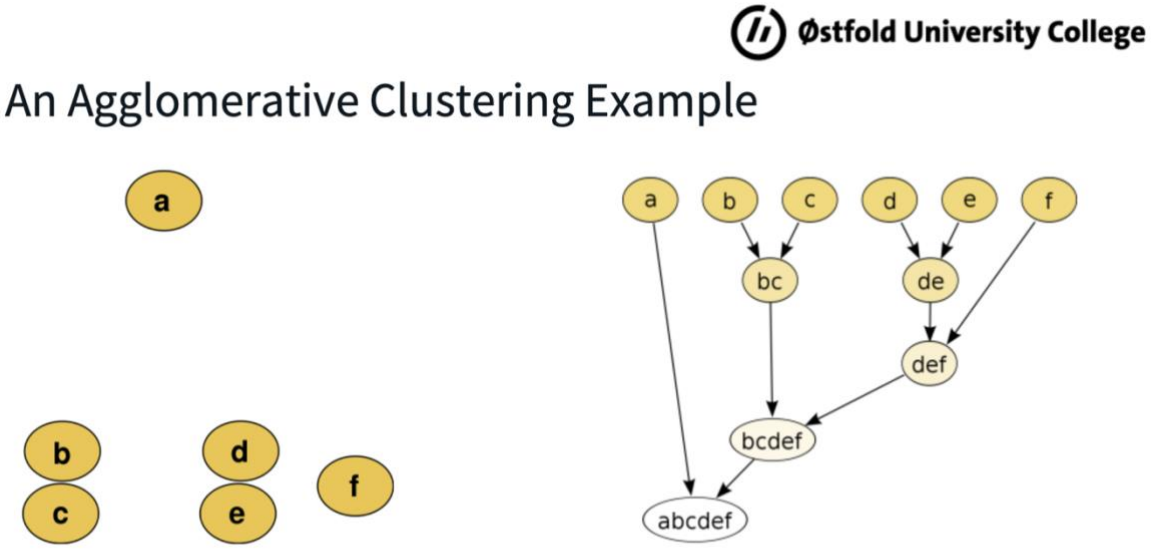
```python
# Silhouette score
kmeans_silhouette_score = silhouette_score(iris_data, kmeans.labels_)
print("K-Means Silhouette score: \n", kmeans_silhouette_score)

# RI score
kmeans_rand = rand_score(kmeans.predict(x_train), y_train)
print("K-Means Rand score: \n", kmeans_rand)
```

Agglomerative algoritme

Agglomerative stammer frs «hierarki» clustering == HCA «Hierarchical Cluster Analysis».

Denne typen cluster-algoritme har 2 kategorier som primært brukes, og jeg skal bruke en av de, som overskriften spesifiserer. Agglomerative funkerer ved at den tar å sjekker avstanden fra punkt til punkt, og velger hvordan den skal merge clusterne til hverandre.

Et eksmepel fra forelesning som gav meg et godt bilde av hva det var, er dette:



Images from https://en.wikipedia.org/wiki/Hierarchical_clustering

I min kode har jeg initialisert algoritme-modellen, og tatt den i bruk med default løsning. Siden vi fant hvor mye «k», altså koeffisienten skulle være i K-means fremgangsmåten, var det enklere å initialisere denne modellen.

Vi satt da n = 3, og brukte parameteren «linkage» = complete.

Linkage er da kriteriet til hvordan algoritmen skal beregne og kalkulere avstanden til datasettet for å danne clusterne.Vi skal tune denne og teste ut forskjellige resultater:

Out of the box med det defaulte oppsettet som beskrevet over, har jeg nå koden:

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'complete')
clusters = AggloClassifier.fit_predict(iris_data)
agglo_score = silhouette_score(iris_data, AggloClassifier.labels_)
print("Agglomerative score: \n", agglo_score)
```

```
Agglomerative score:
 0.5035159025156312
```

Vi kan se at vår silhouette_score er på 0.5035.. I forhold til K-means (0.5528), funker denne faktisk dårligere enn jeg først antok.

Tuning

Jeg vil teste ut forskjellige parametere for å se om det vil utgjøre en forskjell. Jeg fjernet alle parametere i initialiseringen av agglomerativeClustering-funksjonen, og hadde kun n_clusters = 3. dette er da uten linkage inkludert. Og den ga meg et resultat på:

```
Agglomerative score:
 0.5047999262278893
```

Altså litt bedre enn forgjengeren. Men jeg er ikke ferdig nei, langt i fra.

Jeg skal nå teste ut de forskjellige linkage parameterne:

Linkage

Linkage er et kriterie til modellen, den forteller modellen hvilken distanse som skal brukes mellom observasjoner.

«Average»:

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'average')
```

```
Agglomerative score:
 0.5047999262278893
```

«Complete»

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'complete')
```

```
Agglomerative score:
 0.5035159025156312
```

«Single»

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'single')
```

```
Agglomerative score:
 0.5323532834969982
```

(Overraskende resultat!)

«Ward» denne kan kun benyttes ved at affinity = manhattan

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'ward')
```

```
Agglomerative score:
 0.5047999262278893
```

Basert på dette, leverte linkage = «Single» veldig bra, nesten like bra som K-means. Jeg forholder meg videre derfor til «Single».

Affinity

«l1» m/linkage = single

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'single', affinity="l1")
```

```
Agglomerative score:
 0.48556605241891976
```

«l2» m/linkage = single

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'single', affinity="l2")
```

```
Agglomerative score:
 0.5323532834969982
```

(Det beste resultatet hittil.)

«manhattan» m/linkage = single

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'single', affinity="manhattan")
```

```
Agglomerative score:
 0.48556605241891976
```

NB! Gikk vekk fra 2 individuelle bilde,r og valgte å skrive score/konsoll-output som kommentar videre under.

«manhattan» m/linkage = single

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'single', affinity="cosine")
# Console output:
# Agglomerative score:
#   0.5323532834969982
```

Resultat

Ut i fra bruken av attributter har jeg fått en forståelse for hvilken modell som fungerer best. Basert på resultater viser det seg at i mitt tilfelle, og mitt datasett, er dette oppsettet det mest optimale for at modellen skal levere det beste resultatet:

```
AggloClassifier = AgglomerativeClustering(n_clusters = 3, linkage = 'single', affinity="l2")
# Console output:
# Agglomerative score:
#   0.5323532834969982
```

(Det var ekvivalent med bruk av affinity = «cosine»!)

Vi kan også sjekke ut RI scoren til agglomerative objektet:

```
Agglomerative score:
 0.5323532834969982
Agglomerative rand score:
 0.7719015659955257
```

Vi kan se at vi har en helt grei score, 75% + god match, det vil si at det er en ikke så alt for verst spredning av clusterne, og det er god space mellom dem. Det er derfor minimalt med overlapping av clusterne, noe som er bra!

BIRCH algoritme

Min siste algoritme blir BIRCH (balanced iterative reducing and clustering using hierarchies), dette er en «unsupervised» algortime som kjører hierarkisk clustering over datasettet.

Birch er mest brukt blant STORE datasett, og er faktisk mest ettertraktet blant de store. Men ved færre datasett bør den se seg slått av K-means & agglomerative clustering..

Mine resultater er som følger:

```
Score of Birch =  0.5047999262278893
```

Dette viser modellen etter å ha kjørt, og til sammenligning leverer den dårligere enn agglomerativ clustering, som forventet egentlig.

Kodebasen ser slik ut for BIRCH modellen:

```
# == BIRCH ==
Birch_model = Birch(threshold=0.01, n_clusters=3, copy=True, compute_labels=True)
Birch_model.fit(iris_data)
birch_prediction = Birch_model.predict(iris_data)
clusters_br = unique(birch_prediction)
print("Clusters of Birch", clusters_br)
labels_br = Birch_model.labels_

score_br = metrics.silhouette_score(iris_data, labels_br)
print("Score of Birch = ", score_br)

# Plot the results of BIRCH algorithm
x = iris_data.assign(predicted_label=Birch_model.labels_)
birch_plot = sns.pairplot(x, hue='predicted_label', markers=['D', 'o', 's'])
birch_plot.fig.suptitle("BIRCH results", y=1)
plt.show()
```

Tuning / parametere..

Jeg tenkte å teste å tune parameterne til BIRCH modellen for å sjekke resultatene.
Vi har noen parametere vi kan leke oss med:

- Threshold = 0.1:

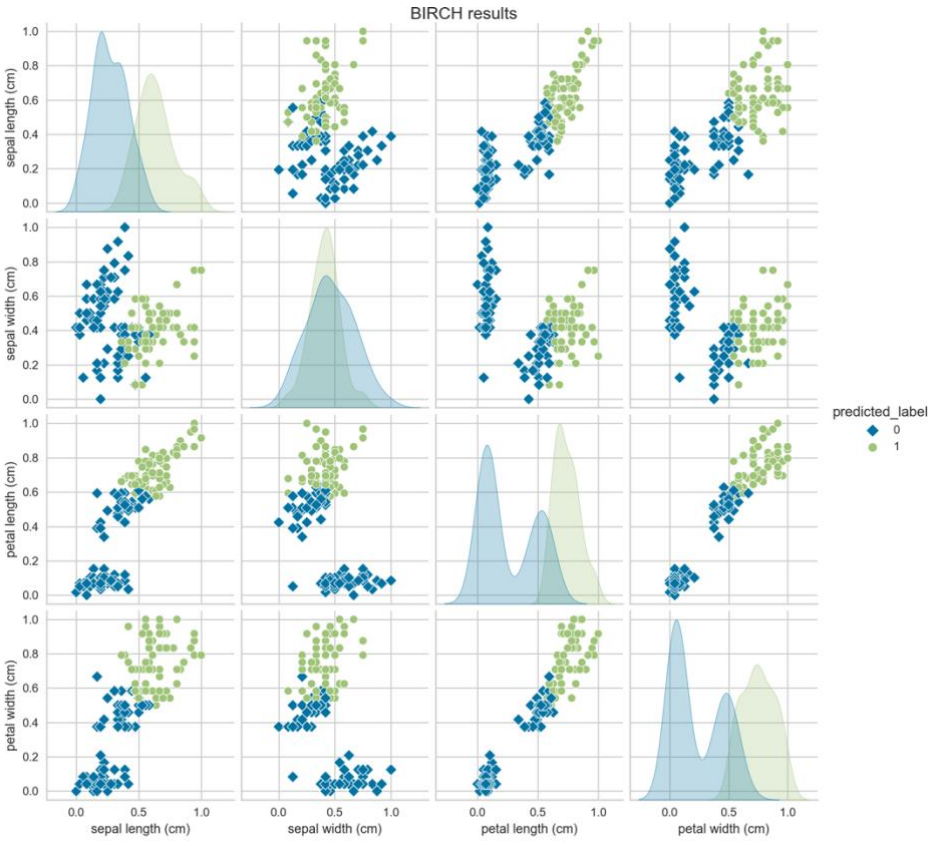
```
Score of Birch =  0.5047999262278893
```

Jeg startet ut med en threshold på 0.01, jeg valgte å endre denne til 0.1 og sjekket resultatet nærmere:

```
Score of Birch =  0.49180066621403
```

Vi ser at modellen leverer dårligere med en threshold på 0.1, dette betyr i alt at vi får mindre splitting blant clusterne som blir produsert. Jo høyere tall, jo mer skal til før en ny subcluster blir laget. Vi kan teste ved å sette threshold = 0.5:

Threshold = 0.5 gir et rart resultat, n_cluster på 3 blir ikke godtatt, og den viser kun 2 clusters:



Vi får også en overall dårlig score basert på denne thresholden:

```
Score of Birch =  0.43865080323191685
```

Jeg tester da ut threshold = 0.05 og får en score på:

```
Score of Birch =  0.5035481707490882
```

Virker som at jo lavere threshold, jo flere subclusters blir laget, som fører til bedre inndeling!

Kan teste med threshold = 0.005 bare for moroskyld:

```
Score of Birch =  0.5047999262278893
```

Har testet forskjellige verdier for danning av sub_clusters, og jeg er sikker på at 0.01 vil gjøre jobben bra nok for min del.

Resultat overall

Basert på alle modellene er jeg komfortabel med å si at jeg syntes k-means hadde best utslag på datasettet «iris». Til sammenligning har de

Vi kan kjapt sammenligne de beste resultatene fra hver modell testet ut:

K-means:

```
K-Means Silhouette score:
 0.5528190123564102
```

Tilhørende RI score:

0.8705

Agglomerative:

```
Agglomerative score:
 0.5323532834969982
Agglomerative rand score:
 0.7719015659955257
```

Birch:

```
Score of Birch =  0.5047999262278893
```

Vi kan se at listen ser slik ut, sortert fra Best til dårligst:

1. K-means
2. Agglomerative
3. Birch

Det var en speppende oblig, men kule algortimer å teste ut. Det var noen errors jeg fikk som gjode at det stoppet litt opp, som gjode at jeg måte feilsøke og sjekke rundt for å finne feiler. Men man lærer av feil, gjør man ikke?

Takk for meg

Stian Larsen