mlp-week10

March 30, 2021

1 Machine Learning in Python - Workshop 10

In this week's workshop we will be exploring several methods for unsupervised learning, specifically some of sklearn's tools for clustering data.

2 1. Setup

2.1 1.1 Packages

In the cell below we will load the core libraries we will be using for this workshop and setting some sensible defaults for our plot size and resolution.

[1]:
```
!pip install -q -r requirements.txt
```

[2]:
```
# Display plots inline
%matplotlib inline

# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# ipywidgets
from ipywidgets import interact

# Plotting defaults
plt.rcParams['figure.figsize'] = (10,6)
plt.rcParams['figure.dpi'] = 80

# sklearn modules
import sklearn
import sklearn.cluster
import sklearn.manifold
import sklearn.preprocessing
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from scipy.cluster.hierarchy import dendrogram
import scipy.cluster
```

2.2 1.2 Helper Functions

For this lab we will make use of a the `plot_dendrogram` function provided in sklearn's clustering documentation. This function takes an `AgglomerativeClustering` model and plots the hierarchical merging of clusters as a dendrogram, more on this later.

[3]:
```
def get_labels(m):
    if hasattr(m, 'labels_'):
        return(m.labels_)
    else:
        return(m.steps[-1][1].labels_)


def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram
    # from sklearn documentaion:
    # https://scikit-learn.org/stable/auto_examples/cluster/
 ↪plot_agglomerative_dendrogram.html

    # create the counts of samples under each node

    if (type(model) == sklearn.pipeline.Pipeline):
        model = model.steps[-1][1]

    counts = np.zeros(model.children_.shape[0])
    labels = get_labels(model)
    n_samples = len(labels)


    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    Z = np.column_stack([model.children_, model.distances_,
                                        counts]).astype(float)

    palette = [mpl.colors.rgb2hex(c) for c in sns.color_palette(n_colors =␣
↪model.n_clusters_)]

    link_cols = {}
    for i, i12 in enumerate(Z[:,:2].astype(int)):
        c1, c2 = (link_cols[x] if x > len(Z) else palette[labels[x]] for x in␣
↪i12)
        link_cols[i+1+len(Z)] = c1 if c1 == c2 else "#808080"

    # Plot the corresponding dendrogram
    dendrogram(Z, link_color_func=lambda x: link_cols[x], **kwargs)


    if (model.n_clusters_ == 1):
        threshold = np.max(model.distances_)
    elif (model.distance_threshold is None):
        i = model.n_clusters_-1
        threshold = (model.distances_[-i] + model.distances_[-(i+1)])/2
    else:
        threshold = model.distance_threshold

    plt.axhline(y=threshold, color="0.25", linestyle="--")
```

2.3 1.3 Data

We will look at several toy data examples to start to get a better understanding of sklearn's clustering algorithms.

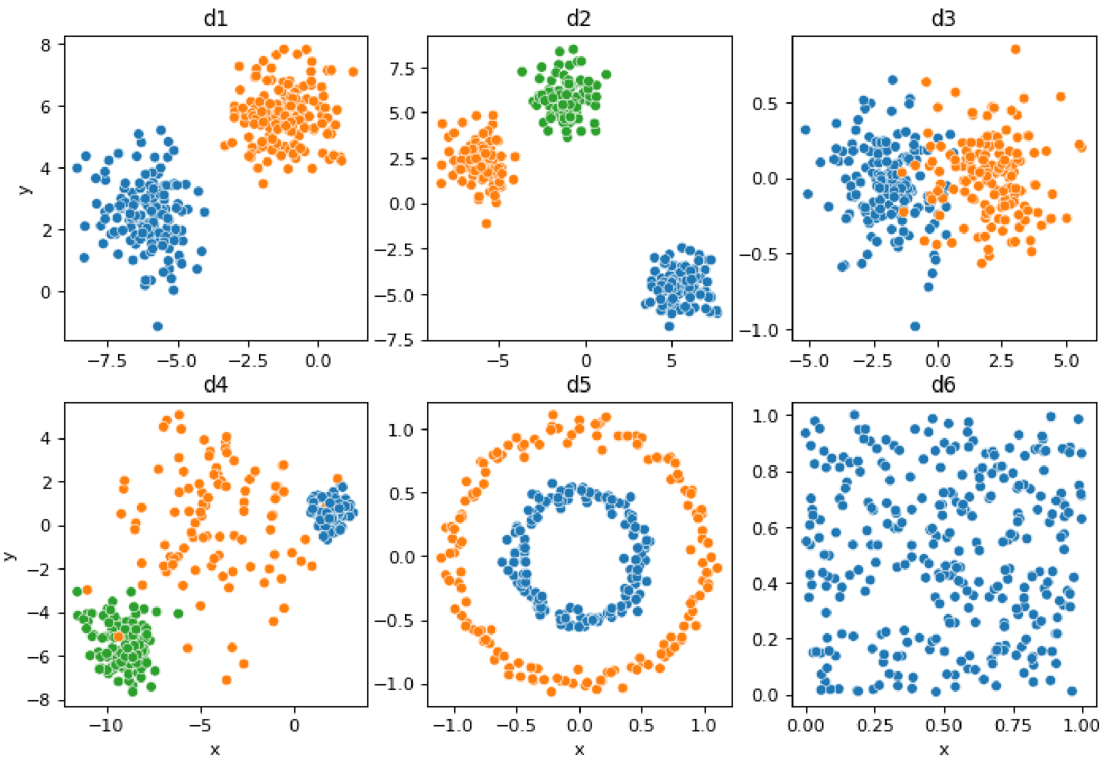
[4]:
```
data = {
    "d1": pd.read_csv("d1.csv"),
    "d2": pd.read_csv("d2.csv"),
    "d3": pd.read_csv("d3.csv"),
    "d4": pd.read_csv("d4.csv"),
    "d5": pd.read_csv("d5.csv"),
    "d6": pd.read_csv("d6.csv"),
}
```

In all six of the data sets we have observations in 2 dimensions with 1, 2, or 3 classes. Our goal will be to attempt to recover these labels as best we are able without providing the true labels to the underlying algorithm fitting our model(s), which is why these methods are called as unsupervised.

[5]:
```
plt.figure(figsize=(10.5,7))

for i, d in zip(range(len(data)), data):
    plt.subplot(231+i)
    ax = sns.scatterplot(x='x', y='y', hue='label', data=data[d], legend=False)
    ax.set_title(d)
    if (i < 3): ax.set_xlabel("")
    if (i % 3 != 0): ax.set_ylabel("")

plt.show()
```



---

3 2. K-Means

We will begin by fitting k-means models to the data sets while adjusting several of the key model parameters. k-means models are straight forward to fit and use a the same sklearn interface that we are used to, they only differ in that they only require a model / data matrix `X` and do not require or use the outcome vector `y`. For the sake of compatibility with the supervised models in sklearn the `fit` method of these models still has an argument `y`, but this is argument is neither required nor used.

Since we are often only interested in the result of the clustering, and not the underlying model, k-means and the other clustering models offer a `fit_predict` method which both fits the model and then predicts cluster labels as a single step for a data matrix `X`. This is similar in spirit to the fit_transform method that is present with the preprocessing transformer like StandardScaler and OneHotEncoder.

Below we implement a simple interactive tool for exploring all six data sets with the k-means algorithm. We have provided widgets that allow you to alter the number of clusters and the method for initializing the cluster centroids (for more on the k-means++ method see this article) and the random state value to seed the random number generator before fitting the model.

[6]:
```
@interact(
    d=["d1", "d2"],
    n_clusters=(1,5),
    init=['k-means++', 'random'],
    random_state='1234'
)

def fit_kmeans(d, n_clusters=2, init='random', random_state='1234'):
    # Convert from string to int
    random_state = int(random_state)

    df = data[d]
    X = df.drop(['label'], axis=1)

    # Fit and predict
    df['cluster'] = sklearn.cluster.KMeans(
        n_clusters=n_clusters, init=init, random_state=random_state
    ).fit_predict(X)
    df = df.astype({'cluster': 'category'})

    # Plot
    plt.figure(figsize=(7,7))
    ax = sns.scatterplot(x='x', y='y', hue='cluster', style='label', data=df)
    ax.set_title(d)

    plt.show()
```

```
interactive(children=(Dropdown(description='d', options=('d1', 'd2'), value='d1'), IntSlider(va
```

3.0.1 Exercise 1

For data set d1 what happens when you change the value of random_state with n_clusters=2? *Hint* - keep trying until something interesting happens.

For every value of random_state with n_clusters=2, we have that the K-means algorithm always creates two separate clusters, one on the top right and the other on the bottom left. The only interesting thing is that the cluster labels flip with different values of random state.

3.0.2 Exercise 2

For data set `d1` set `n_clusters=3` do the clusters change as you vary `init` and `random_state`?

The clusters do not vary as we alter `init` and `random_State`.

3.0.3 Exercise 3

Explore the different data sets with different values of `n_cluster`, describe the nature of the boundaries produced by a k-means clustering. *Hint* - don't just consider the binary (`k=2`) case.

For dataset d1: - k=1 classifies all instances into one cluster - k=2 classifies the intstances into two clusters, one on the top left and one on the bottom right. We would expect the top left two clusters to be more similar to eachother than to the bottom right cluster. - k=3 splits the bottom left cluster into two separate clusters where the algorithm thinks there could be another cluster. It appears to be placing another centeroid at (-6,1) - k=4 splits the bottom cluster as in the k=3 case. There is also a splitting of the top right cluster into two clusters with a centeroid at (0,6) - k=5 splits the bottom cluster again into three clusters. Placing centeroids at approximately (-5,3.75) , (-7,3)

For dataset d2: - k=1 classifies all instances into one cluster - k=2 classifies the intstances into two clusters, one on the top left and one on the bottom right. We would expect the top left two clusters to be more similar to eachother than to the bottom right cluster. - k=3 splits the data into three clusters as a human would do in this task. - k=4 splits the top left cluster into two clusters - k=5 splits the top left cluster into two and also splits the topmost cluster also into two

In each of these cases the decision boundary for the clusters is linear, since the k-means algorithm uses euclidean distance to classify instances.

3.0.4 Exercise 4

Explain why we can or cannot construct a confusion matrix using the predictions from the k-means model (for the purposes of model scoring).

We cannot construct a confusion matrix since we are performing unsupervised learning and do not wish to give the algorithm access to the actual data scores. Therefore, we cannot predict the numbers of TP,FP,TN and FN.

In all of the provided data sets the true number of clusters is provided and is also fairly obvious just by visual inspection of the data. However, with real world data this value is unlikely to be known ahead of time. A simple way of potentially determining this value is to fit a series of k-means models with different values of `k` an then comparing the resulting within-cluster sum-of-squares (or what sklearn calls the inertia).

3.0.5 Exercise 5

In general, would you expect the within-cluster sum-of-squares (inertia) to increase, decrease or stay the same as `k` is increased?

As K increases, the within-cluster sum-of-squares will decrease since each cluster becomes more compact with instances closer together. This is why this method is not good as a tool to examine model performance or pick the value of K, since it will prefer an arbitarily high value of K, that may not be best for our example.

Below we have reused the code from our previous interactive plot but now we include a second plot that shows the inertia as a function of `n_cluster`. For each of the data sets play with different values of `n_clusters` and try to develop some intuition on how the inertia relates to the model's fit.

[7]:
```
@interact(
    d=data.keys(),
    n_clusters=(1,5)
)

def fit_kmeans(d, n_clusters=2):
    # Read the data, create the model matrix, and fit and predict
    df = data[d]
    X = df.drop(['label'], axis=1)

    models = [ sklearn.cluster.KMeans(n_clusters=n).fit(X) for n in range(1,6)]
    scores = [m.inertia_ for m in models]

    df['cluster'] = models[n_clusters-1].predict(X)
    df = df.astype({'cluster': 'category'})

    # Plot
    plt.figure(figsize=(12,6))

    plt.subplot(121)
    sns.scatterplot(x='x', y='y', hue='cluster', style='label', data=df)

    plt.subplot(122)
    plt.plot(np.arange(1,6), scores, 'o-')
    plt.xlabel("n_clusters")
    plt.ylabel("inertia")
    plt.xticks(range(1,6))
    plt.show()
```

```
interactive(children=(Dropdown(description='d', options=('d1', 'd2', 'd3', 'd4', 'd5', 'd6'), v
```

3.0.6 Exercise 6

Based on these plots, for which data sets does k-means appear to be doing a good job - i.e. it fits well and the correct number of clusters can be inferred from the data. For which data sets does it not do a good job?

Based on these plots, k-means does a good job on datasets d1 and d2, since we can see there is very clearly defined spaces between the clusters and we can easily identify where the centeroids should be located.

k-means does a resonable job on datasets d3 and d4, where we see the clear elbow on the graph prefering k=2 and k=3 respectively.

k-means does a poor job on dataset d5 since the clusters are spherical and k-means gives linear decision boundaries based on distance from the centeroids. We would prefer a DBSCAN algorithm here.

The plot does not give us much information for dataset d6 since there is no clear elbow.

3.0.7 Exercise 7

Describe the features of the inertia plots that suggest whether a k-means model is fitting well and how this relates to determining $n_{clusters}$.

When we are looking at an inertia plot, we are looking for the clear elbow point, past which we see diminishing returns in the effect on the inertia by increasing the number of clusters. We then select n_{clusters} to be equal to this elbow point

4 3. Hierarchical Clustering

While there is a large number of different methods for hierarchical clustering, the methods implemented in sklearn under the `AgglomerativeClustering` model use a bottom up approach by starting each observation as its own cluster and then merging existing clusters to create new larger clusters. The rule(s) used for merging these clusters are determined by the `linkage` argument which takes one of the following values:

- `"ward"` - minimizes the sum of squared differences within all clusters.
- `"complete"` - minimizes the maximum distance between observations of pairs of clusters.

- "average" - minimizes the average of the distances between all observations of pairs of clusters.
- "single" - minimizes the distance between the closest observations of pairs of clusters.

The interactive widgets below will let you play with these different approaches. We have also provided the option of specifying `n_clusters` or `distance_threshold` - the former is ignored if the latter is not `None`.

[8]:
```
@interact(
    d=data.keys(),
    n_clusters=(1,5),
    distance_threshold="",
    linkage = ["ward", "complete", "average", "single"]
)

def fit_aggclust(d, n_clusters, distance_threshold, linkage):
    #distance_threshold = float(distance_threshold)

    try:
        distance_threshold = float(distance_threshold)
    except:
        distance_threshold = None

    if (distance_threshold is not None):
        n_clusters = None

    d = data[d]
    X = d.drop(['label'], axis=1)

    # Fit model
    m = sklearn.cluster.AgglomerativeClustering(
        distance_threshold=distance_threshold, n_clusters=n_clusters,
        linkage=linkage, compute_full_tree = True, compute_distances = True
    ).fit(X)

    d['cluster'] = m.labels_
    d['cluster'] = d['cluster'].astype('category')

    # Plot
    plt.figure(figsize=(12,6))

    plt.subplot(121)
    sns.scatterplot(x='x', y='y', hue='cluster', style='label', data=d)

    plt.subplot(122)
    plot_dendrogram(m, show_leaf_counts=False, no_labels=True)
    plt.show()
```

```
interactive(children=(Dropdown(description='d', options=('d1', 'd2', 'd3', 'd4', 'd5', 'd6'), 
```

4.0.1 Exercise 8

How does the `distance_threshold` argument relate to the number of clusters predicted by the model?

The `distance_threshold` argument is "the linkage distance threshold above which, clusters will not be merged". If we pick a value of the `distance_threshold`, then we require number of clusters to be none, since we are now stopping to link clusters related to their distance, we cannot force new clusters if we have not met the desired number of clusters.

4.0.2 Exercise 9

For `d1` and `d2` which of the linkage method(s) work best for identifying the correct number of clusters present?

For d1, we know that we are looking for 2 clusters. The Ward linkage method gives the greatest height (dissimilarity) between the two clearly defined clusters.

For d2, we know that we are looking for 3 clusters. The Ward and complete methods give the most compelling reason to pick three clusters.

4.0.3 Exercise 10

What about for `d3` and `d4`? Which of the linkage method(s) work best for identifying the correct number of clusters present?

For d3, we know that we are looking for 2 clusters. The Ward method would lead us to correctly selecting 2 clusters. Average and single are poor here since the greatest heigh would be at 3 clusters.

For d4, we know that we are looking for 3 clusters. The Ward method would lead to us correctly selecting 3 clusters most likely. Complete looks reasonable. The single and average methods do not give compelling reasons for selections other than 2.

4.0.4 Exercise 11

Finally, what about for `d5` and `d6`?

For d5, we know that we are looking for 2 clusters. Since we are using k-means and the decision boundaries of our clusters should be spherical rather than linear, none of the methods are very helpful.

For d6, we know that we are looking for 1 clusters. We do not have a method with dendrograms to pick only one cluster so our method falls apart here. Most of the methods would reccomend 2 or 3 clusters.

4.0.5 Exercise 12

Based on what you found with Exercises 9-11 can you come up with a reasonable heuristic for choosing which linkage function to use?

If we have only one cluster, then these methods are useless (d6). If we have clusters that cannot be clearly separated by linear decsiion boundaries then the methods are useless too (d5).

Otherwise, when we have clusters that can easily be separated by linear decision boundaries (d1,d2) I would recommend ward or complete, simple and average perform poorly. When we have clusters that are harder to clearly separate (d3,d4) I would recommend using Ward as the method.

[ ]:
```

```

---

5 4. Clustering Animals

We will wrap this workshop up by looking at a slightly more complicated example. These data come from the `palmerpenguin` R package by Allison Horst and represent data collected on penguin species occuring around Palmer Station in Antarctica by Dr. Kristen Gorman. The data contain various measurement details of the sample penguin's physical characteristics as well as the specific island they were located on. The data provided in `palmerpenguin.csv` reflect a lightly modified version of the data available in the package (missing values and the `year` variable have been removed.)

[9]:
```
penguin = pd.read_csv("palmerpenguins.csv")
penguin
```

[9]:
```
       species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0       Adelie  Torgersen            39.1           18.7                181
1       Adelie  Torgersen            39.5           17.4                186
2       Adelie  Torgersen            40.3           18.0                195
3       Adelie  Torgersen            36.7           19.3                193
4       Adelie  Torgersen            39.3           20.6                190
..         …          …             …              …                  …
328  Chinstrap      Dream            55.8           19.8                207
329  Chinstrap      Dream            43.5           18.1                202
```

```
330  Chinstrap      Dream             49.6            18.2                193
331  Chinstrap      Dream             50.8            19.0                210
332  Chinstrap      Dream             50.2            18.7                198

     body_mass_g     sex
0           3750    male
1           3800  female
2           3250  female
3           3450  female
4           3650    male
..           …       …
328         4000    male
329         3400  female
330         3775    male
331         4100    male
332         3775  female

[333 rows x 7 columns]
```

Our goal is to now use the clustering methods we've just learned and see how well we are able to distinguish between the different species of penguin using an *unsupervised* modelling approach. Specifically, in this case we already know the species, but it is conceivable that a similar project could have been undertaken where the question of interest is how many species are present amoung the sampled individuals.

Below we set up a dictionary containing two pipelines, one for agglomerative clustering and the other for k means clustering. These pipelines are configured so that they take care of the dummy coding for the categorical variables and the basic configuration of the clustering models.

[10]:
```
models = {
    "AgglomerativeClustering": make_pipeline(
        make_column_transformer(
            (sklearn.preprocessing.OneHotEncoder(), ['island', 'sex']),
            remainder = "passthrough"
        ),
        sklearn.cluster.AgglomerativeClustering(
            compute_full_tree = True, compute_distances = True
        )
    ),
    "KMeans": make_pipeline(
        make_column_transformer(
            (sklearn.preprocessing.OneHotEncoder(), ['island', 'sex']),
            remainder = "passthrough"
        ),
        sklearn.cluster.KMeans()
    ),
    "KMeans_Scaled": make_pipeline(
        make_column_transformer(
            (sklearn.preprocessing.OneHotEncoder(), ['island', 'sex']),
            (sklearn.preprocessing.StandardScaler(), ['bill_length_mm',
                                                       'bill_depth_mm',
                                                       'flipper_length_mm',
                                                       'body_mass_g']),
            remainder = "passthrough"
        ),
        sklearn.cluster.KMeans()
    )
}
```

[11]:
```
models
```

[11]:
```
{'AgglomerativeClustering': Pipeline(steps=[('columntransformer',
                  ColumnTransformer(remainder='passthrough',
                                    transformers=[('onehotencoder',
                                                   OneHotEncoder(),
                                                   ['island', 'sex'])])),
                 ('agglomerativeclustering',
                  AgglomerativeClustering(compute_distances=True,
                                          compute_full_tree=True))]),
 'KMeans': Pipeline(steps=[('columntransformer',
                  ColumnTransformer(remainder='passthrough',
                                    transformers=[('onehotencoder',
                                                   OneHotEncoder(),
                                                   ['island', 'sex'])])),
                 ('kmeans', KMeans())]),
 'KMeans_Scaled': Pipeline(steps=[('columntransformer',
                  ColumnTransformer(remainder='passthrough',
                                    transformers=[('onehotencoder',
                                                   OneHotEncoder(),
                                                   ['island', 'sex']),
                                                  ('standardscaler',
                                                   StandardScaler(),
                                                   ['bill_length_mm',
                                                    'bill_depth_mm',
                                                    'flipper_length_mm',
                                                    'body_mass_g'])])),
                 ('kmeans', KMeans())])}
```

We will now setup a function which allows us to interactively select the model of our choice and change the basic parameters `n_clusters` for both and `linkage` for the agglomerative clustering. The function will then report the performance of the model by showing the allocation of species within each of the predicted clusters as well as show the calculated homogenity and completeness of these clusters. In the case of the agglomerative clustering, a dendrogram of the cluster will also be shown.

```
[12]: @interact(
          m = models.keys(),
          n_clusters=(1,8),
          linkage = ["ward", "complete", "average", "single"]
      )

      def fit_penguin(m, n_clusters=4, linkage="ward"):
          m = models[m]

          model_type = m.steps[-1][0]
          params = {}
          params[model_type+"__n_clusters"] = n_clusters

          if (model_type == "agglomerativeclustering"):
              params[model_type+"__linkage"] = linkage

          m.set_params(**params)


          # Fit model
          X = penguin.drop(["species"], axis=1)
          m_fit = m.fit(X)

          # Calc scores
          homogenity = sklearn.metrics.homogeneity_score(penguin.species,␣
       ↪get_labels(m_fit))
          completeness = sklearn.metrics.completeness_score(penguin.species,␣
       ↪get_labels(m_fit))

          # Count species in clusters
          pred = pd.DataFrame(penguin.species)
          pred["cluster"] = get_labels(m_fit)

          clusters = pred.value_counts(sort=False).rename("n").to_frame().
       ↪reset_index()

          # Plot
          plt.figure(figsize=(12,6))

          plt.subplot(121)
          ax = sns.barplot(x="cluster", y ="n", hue="species", data = clusters,␣
       ↪palette = sns.color_palette("Set2"))
          ax.set_title("Homogenity: {:.3f},   Completeness: {:.3f}".
       ↪format(homogenity, completeness))


          if (model_type == "agglomerativeclustering"):
```

```
        plt.subplot(122)
        plot_dendrogram(m, no_labels = True)

    plt.show()
```

```
interactive(children=(Dropdown(description='m', options=('AgglomerativeClustering', 'KMeans',
```

5.0.1 Exercise 13

Experimenting with the given parameter choices, which approach (if any) appears to do the best job of correctly identifying the three species?

None of the k-means algorithms do a very good job of identifying the three species, even forcing 3 clusters we have many penguin species being misclassified.

In the agglomerative clustering case, when forcing number of clusters to 3, none of the linkage methods give very satisfactory results. Ward seems to be the best but with still low Homogeneity and completness scores. Most of the methods would recommend selecting 2 clusters.

5.0.2 Exercise 14

We have previously seen that certain machine learning methods are sensitive to the scaling of the features, do you believe this is likely to be the case here, explain why or why not.

Yes, I believe that feature scaling is import. Particularly in this case where the weight of the penguins is very large (as measured in grams) compared to the scales of the other features. Therefore, we would prefer to scale the data before running the algorithm.

5.0.3 Exercise 15

Add two new models to the `models` dictionary that implements scaling of the numeric features as part of the pipeline (e.g. use `sklearn.preprocessing.StandardScaler`), name these `KMeans_Scaled` respectively.

Does this improve the performance of the clustering? Which approach (if any) now appears to be best?

This improves the model performance dramatically, the gentoos are perfectly separated from the adelies and Chinstrap penguins in the 2 and 3 cluster cases. However, the algorithm is still struggling to separate the adelies and chinstraps for each of the linkage methods.

It is interesting to note that if we were to pair the 2 relevant classes containing only one species in the 6-cluster case, we would be almost perfectly classifying the penguin species.

5.1 4. Competing the worksheet

At this point you have hopefully been able to complete all the preceeding exercises. Now is a good time to check the reproducibility of this document by restarting the notebook's kernel and rerunning all cells in order.

Once that is done and you are happy with everything, you can then run the following cell to generate your PDF and turn it in on gradescope under the `mlp-week10` assignment.

```
[ ]: !jupyter nbconvert --to pdf mlp-week10.ipynb
```

```
[NbConvertApp] Converting notebook mlp-week10.ipynb to pdf
[NbConvertApp] Support files will be in mlp-week10_files/
[NbConvertApp] Making directory ./mlp-week10_files
[NbConvertApp] Writing 70698 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
```