Week5

October 15, 2024

0.1 Unsupervised Learning Final Project: Skin Disease Classification with Clustering Models

0.1.1 Overview

For my final project, I have chosen to work on a multi-class classification problem involving a dermatology dataset from Kaggle. The aim is to diagonise the patient's skin condition as one of a set of known diseases based on a list of attributes. The dataset description notes that there are 34 attributes per patient, 12 of which come from a clinical examination of the patient and 22 of which come from analysis of skin samples under a microscope. The list of six diseases includes: psoriasis, seborrheic dermatitis, lichen planus, pityriasis rosea, chronic dermatitis, and pityriasis rubra pilaris.

In this report, I will apply unsupervised learning to this problem. I will first import and clean the data, perform exploratory data analysis, and convert the data to a format suitable for clustering algorithms. I will then train, test, and optimize models using two different approaches: K-Means Clustering and Hierarchical Clustering.

[11]:
```
#Import Python libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
#!pip install plotly
import plotly.express as px
from itertools import permutations
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.metrics import confusion_matrix
```

0.1.2 Data Description and Cleaning

The first step is to import the data and do a high-level inspection. Using the Pandas library, I will load the data into a DataFrame and use the "describe" and "info" functions for an overview.

[12]:
```
#Import data
df = pd.read_csv("dermatology_database_1.csv")
pd.set_option('display.max_columns', None)
df.describe()
```

[12]:

	erythema	scaling	definite_borders	itching
count	366.000000	366.000000	366.000000	366.000000
mean	2.068306	1.795082	1.549180	1.366120
std	0.664753	0.701527	0.907525	1.138299
min	0.000000	0.000000	0.000000	0.000000
25%	2.000000	1.000000	1.000000	0.000000
50%	2.000000	2.000000	2.000000	1.000000
75%	2.000000	2.000000	2.000000	2.000000
max	3.000000	3.000000	3.000000	3.000000

	koebner_phenomenon	polygonal_papules	follicular_papules
count	366.000000	366.000000	366.000000
mean	0.633880	0.448087	0.166667
std	0.908016	0.957327	0.570588
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	1.000000	0.000000	0.000000
max	3.000000	3.000000	3.000000

	oral_mucosal_involvement	knee_and_elbow_involvement
count	366.000000	366.000000
mean	0.377049	0.614754
std	0.834147	0.982979
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	1.000000
max	3.000000	3.000000

	scalp_involvement	family_history	melanin_incontinence
count	366.000000	366.000000	366.000000
mean	0.519126	0.125683	0.404372
std	0.905639	0.331946	0.869818
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	1.000000	0.000000	0.000000
max	3.000000	1.000000	3.000000

	eosinophils_infiltrate	PNL_infiltrate	fibrosis_papillary_dermis
count	366.000000	366.000000	366.000000

mean	0.139344	0.546448	0.336066
std	0.411790	0.815451	0.853139
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	1.000000	0.000000
max	2.000000	3.000000	3.000000

	exocytosis	acanthosis	hyperkeratosis	parakeratosis	\
count	366.000000	366.000000	366.000000	366.000000	
mean	1.368852	1.956284	0.527322	1.289617	
std	1.104418	0.712512	0.757116	0.917562	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	2.000000	0.000000	1.000000	
50%	2.000000	2.000000	0.000000	1.000000	
75%	2.000000	2.000000	1.000000	2.000000	
max	3.000000	3.000000	3.000000	3.000000	

	clubbing_rete_ridges	elongation_rete_ridges	\
count	366.000000	366.000000	
mean	0.663934	0.991803	
std	1.056829	1.162161	
min	0.000000	0.000000	
25%	0.000000	0.000000	
50%	0.000000	0.000000	
75%	2.000000	2.000000	
max	3.000000	3.000000	

	thinning_suprapapillary_epidermis	spongiform_pustule	\
count	366.000000	366.000000	
mean	0.633880	0.295082	
std	1.034924	0.670578	
min	0.000000	0.000000	
25%	0.000000	0.000000	
50%	0.000000	0.000000	
75%	1.000000	0.000000	
max	3.000000	3.000000	

	munro_microabcess	focal_hypergranulosis	disappearance_granular_layer	\
count	366.000000	366.000000	366.000000	
mean	0.363388	0.393443	0.464481	
std	0.759721	0.849406	0.864899	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	
75%	0.000000	0.000000	1.000000	
max	3.000000	3.000000	3.000000	

```
       vacuolisation_damage_basal_layer  spongiosis  \
count                        366.000000  366.000000
mean                           0.456284    0.953552
std                            0.954873    1.130172
min                            0.000000    0.000000
25%                            0.000000    0.000000
50%                            0.000000    0.000000
75%                            0.000000    2.000000
max                            3.000000    3.000000

       saw_tooth_appearance_retes  follicular_horn_plug  \
count                  366.000000            366.000000
mean                     0.453552              0.103825
std                      0.954744              0.450433
min                      0.000000              0.000000
25%                      0.000000              0.000000
50%                      0.000000              0.000000
75%                      0.000000              0.000000
max                      3.000000              3.000000

       perifollicular_parakeratosis  inflammatory_mononuclear_infiltrate  \
count                    366.000000                           366.000000
mean                       0.114754                             1.866120
std                        0.488723                             0.726108
min                        0.000000                             0.000000
25%                        0.000000                             1.000000
50%                        0.000000                             2.000000
75%                        0.000000                             2.000000
max                        3.000000                             3.000000

       band_like_infiltrate       class
count            366.000000  366.000000
mean               0.554645    2.803279
std                1.105908    1.597803
min                0.000000    1.000000
25%                0.000000    1.000000
50%                0.000000    3.000000
75%                0.000000    4.000000
max                3.000000    6.000000
```

[13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 35 columns):
 #   Column                                Non-Null Count  Dtype
```

```
---  ------                                 --------------  -----
 0   erythema                               366 non-null    int64
 1   scaling                                366 non-null    int64
 2   definite_borders                       366 non-null    int64
 3   itching                                366 non-null    int64
 4   koebner_phenomenon                     366 non-null    int64
 5   polygonal_papules                      366 non-null    int64
 6   follicular_papules                     366 non-null    int64
 7   oral_mucosal_involvement               366 non-null    int64
 8   knee_and_elbow_involvement             366 non-null    int64
 9   scalp_involvement                      366 non-null    int64
 10  family_history                         366 non-null    int64
 11  melanin_incontinence                   366 non-null    int64
 12  eosinophils_infiltrate                 366 non-null    int64
 13  PNL_infiltrate                         366 non-null    int64
 14  fibrosis_papillary_dermis              366 non-null    int64
 15  exocytosis                             366 non-null    int64
 16  acanthosis                             366 non-null    int64
 17  hyperkeratosis                         366 non-null    int64
 18  parakeratosis                          366 non-null    int64
 19  clubbing_rete_ridges                   366 non-null    int64
 20  elongation_rete_ridges                 366 non-null    int64
 21  thinning_suprapapillary_epidermis      366 non-null    int64
 22  spongiform_pustule                     366 non-null    int64
 23  munro_microabcess                      366 non-null    int64
 24  focal_hypergranulosis                  366 non-null    int64
 25  disappearance_granular_layer           366 non-null    int64
 26  vacuolisation_damage_basal_layer       366 non-null    int64
 27  spongiosis                             366 non-null    int64
 28  saw_tooth_appearance_retes             366 non-null    int64
 29  follicular_horn_plug                   366 non-null    int64
 30  perifollicular_parakeratosis           366 non-null    int64
 31  inflammatory_mononuclear_infiltrate    366 non-null    int64
 32  band_like_infiltrate                   366 non-null    int64
 33  age                                    366 non-null    object
 34  class                                  366 non-null    int64
dtypes: int64(34), object(1)
memory usage: 100.2+ KB
```

On inspection of the data, we see that the dataframe consists of 366 rows and 35 columns. This is a fairly small dataset, so memory usage and disk space is not an issue. Most of the attributes are rated on an integer scale from 0 to 3, with family history as an exception (0 to 1). The "class" column contains target labels from 1 to 6 which represent the six diseases. There is no information provided about which number corresponds to which disease, but this is not necessary to evaluate the performance of a model.

There are no null values to worry about, but one issue which will need to be cleaned up is the "age" column: it's datatype is listed as "object", when it should be a numerical datatype. I will inspect the column using the value_counts function:

[14]:
```
df['age'].value_counts()[:-1]
```

[14]:
```
50    17
40    17
27    16
36    16
22    15
55    14
35    14
25    14
30    13
33    12
52    11
60    11
42    10
18     9
?      8
34     8
20     8
62     7
10     7
45     7
51     7
8      7
46     6
19     6
47     6
32     6
56     5
17     5
16     5
48     5
28     5
44     5
41     4
7      4
43     4
70     4
26     3
38     3
12     3
23     3
29     3
21     3
39     2
37     2
15     2
```

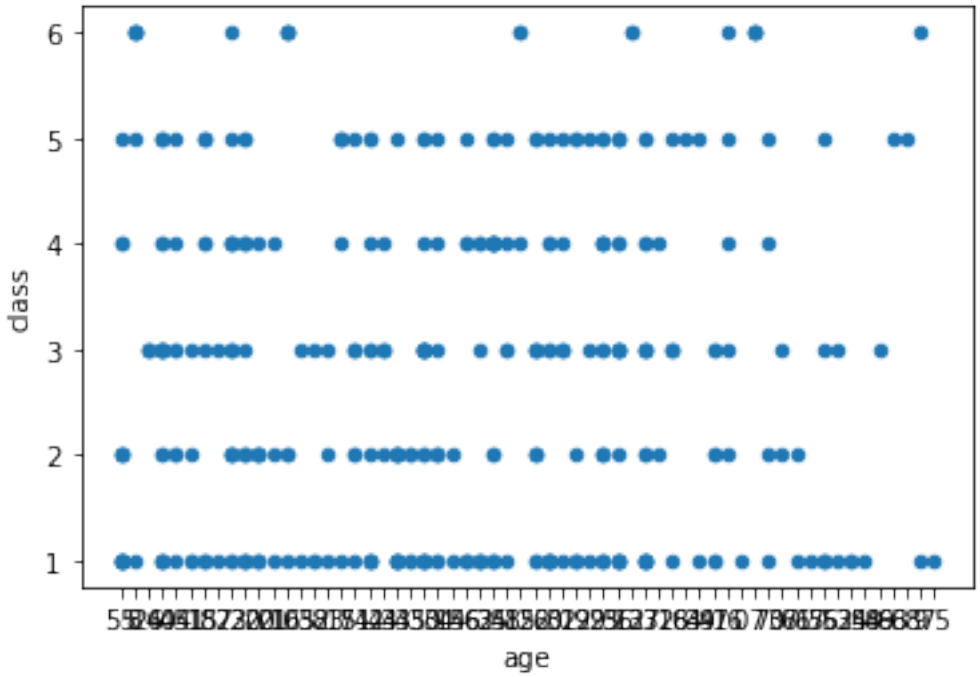
```
65     2
9      2
53     2
24     2
13     2
31     2
57     2
61     2
75     1
63     1
0      1
49     1
68     1
58     1
64     1
Name: age, dtype: int64
```

This function reveals that the age column is missing values for 8 patients, marked as "?". There are a few ways of dealing with this. We could drop the 8 rows from the dataset, drop the entire age column, or impute the missing values with the average age of the sample. Before deciding what to do, I will look at the class labels for the missing values and also plot the available age data vs. class label:

[15]:
```
print(df[df['age'] == '?']['class'])

df2 = df[df['age'] != '?']
df2.plot.scatter(x='age', y='class')
```

```
33     1
34     4
35     2
36     3
262    5
263    5
264    5
265    5
Name: class, dtype: int64
```

[15]: `<matplotlib.axes._subplots.AxesSubplot at 0x77e848d0bb50>`



From the plot, it appears that the age values are fairly uniformly distributed for each of the classes, and thus aren't likely to be crucial to the model. The 8 samples with missing values contain an oversample of label 5, so I don't want to remove them or impute values which could distort the model. Therefore, I will remove the age column.

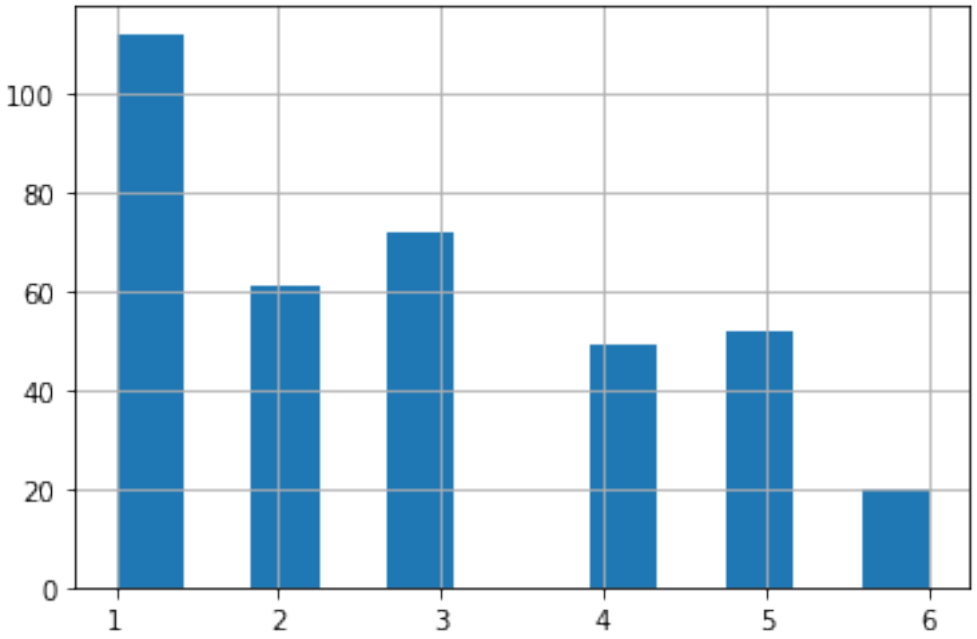
[16]:
```
data = df.drop(['age'], axis=1)
```

0.1.3 Exploratory Data Analysis

Before choosing a model I will perform exploratory data analysis (EDA) on the data. First, I will look at the distribution of the class labels. It seems that there is a fairly equal sample of classes 2-5, but a larger amount of class 1 and a smaller amount of class 6:

[17]:
```
data['class'].hist(bins=12)
```

[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x77e848dd9350>
```



With only 33 remaining features, I can easily visualize histograms for all of them:

[18]:
```
data.drop(['class'], axis=1).hist(figsize=(25,25))
```

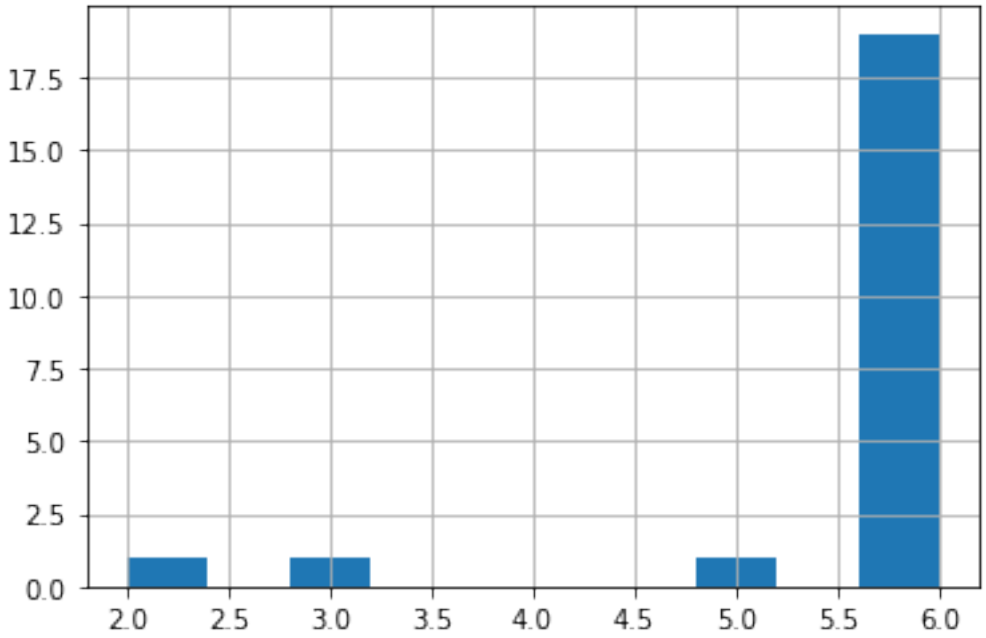
[18]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x77e848c4c9d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848bef6d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848ba4b90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848b68250>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848b1c8d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848b52f50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x77e848b14690>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848ac9c50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848ac9c90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848a8c450>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e8489eeb90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e8489bb710>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x77e848971d90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848933450>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e8488e8ad0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e8488ab190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848860810>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e848897e90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd2cd550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd302bd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd2c5290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd274fd0>,
```

```
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd2365d0>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd1ecb50>],
 [<matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd1ad110>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd165690>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd11ac10>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd0dc1d0>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd091750>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd0c7cd0>],
 [<matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd00b290>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dd041810>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dcff4d90>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dcfb9350>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dcf6e8d0>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x77e7dcf25e50>]],
dtype=object)
```

What is immediately noticable amount these features is that some such as “itching” and “parakeratosis” are vary common in all degrees, while others such as “perifollicular_parakeratosis” and “follicular_papules” are rarely present (above 0). My suspicion is that these rare attributes could be highly predicative of a certain disease. Let’s have a look at the class distributions for some of them:

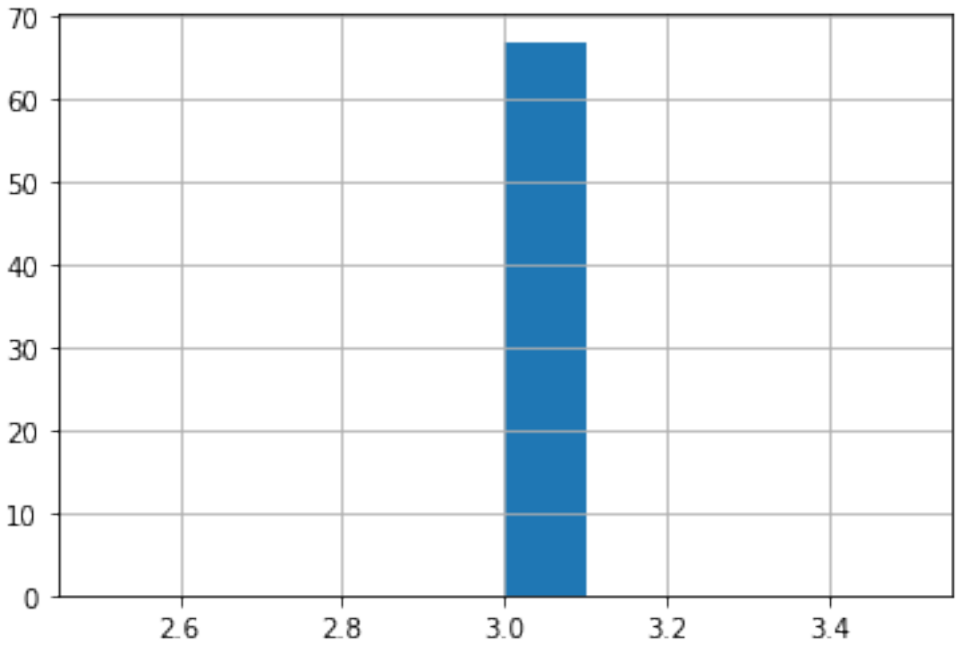
[19]:
```
data[data['follicular_horn_plug'] > 0]['class'].hist()
```

[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x77e7dce241d0>
```



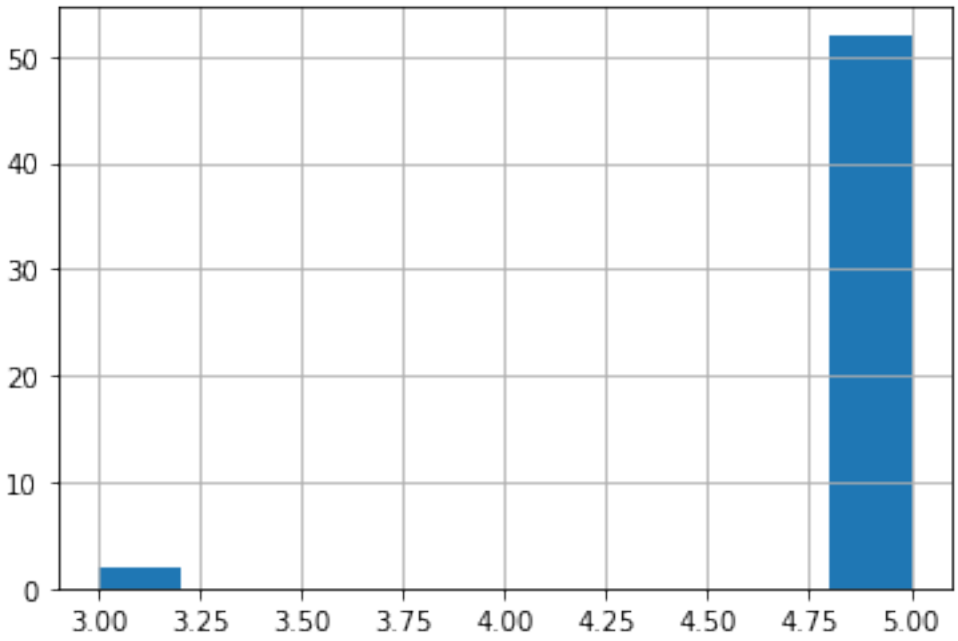
[20]:
```
data[data['oral_mucosal_involvement'] > 0]['class'].hist()
```

[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x77e7db954590>
```



[21]:
```
data[data['fibrosis_papillary_dermis'] > 0]['class'].hist()
```

[21]: <matplotlib.axes._subplots.AxesSubplot at 0x77e7dbbc4810>

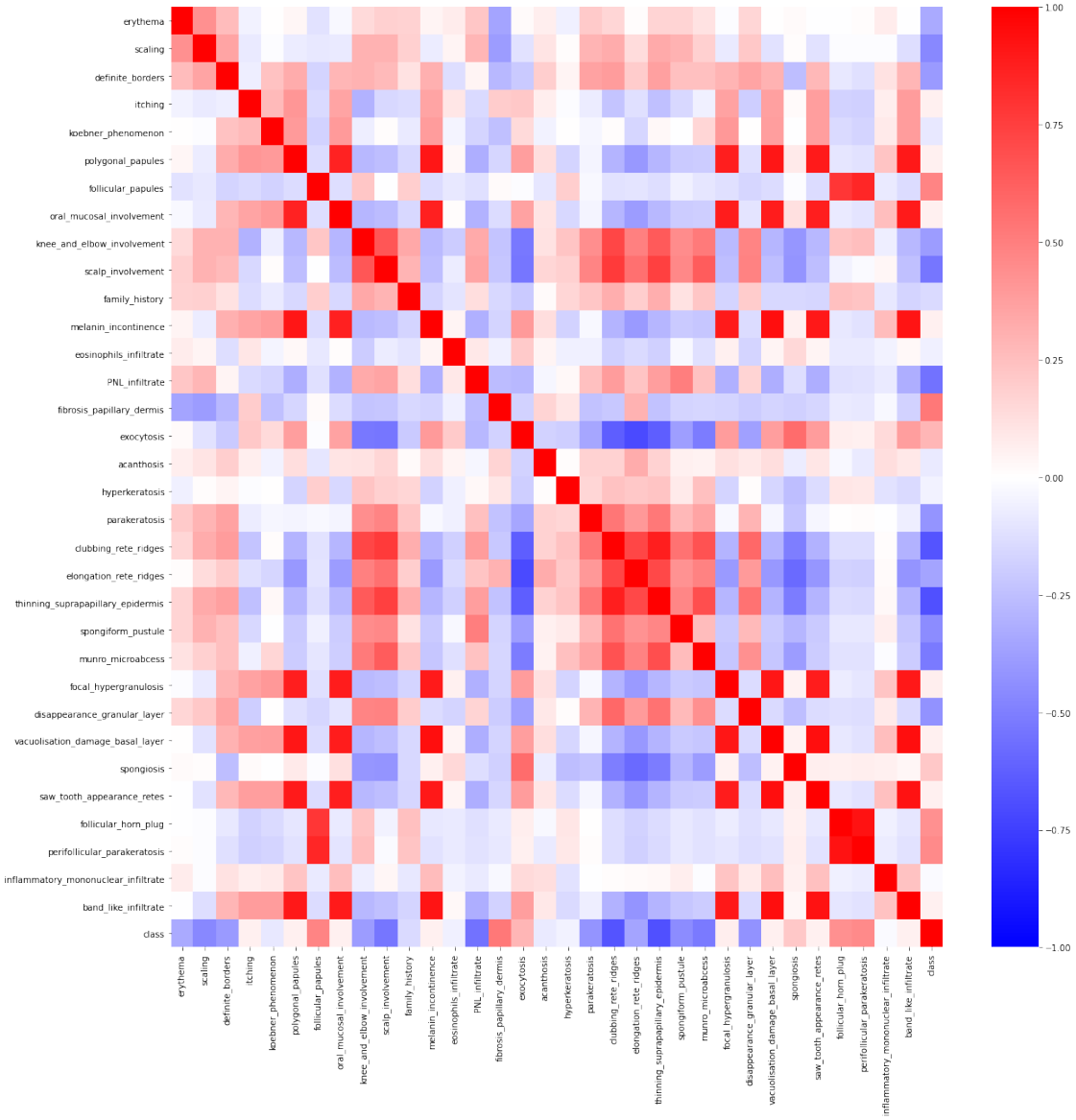


Indeed, the presence of rare attributes appears highly predictive of the class of disease: for "follicular_horn_plug" is is class 6, for "oral_mucosal_involvement" it is class 3, and for "fibrosis_papillary_dermis" it is class 5.

The next step in my analysis is to create a correlation matrix to see which of these features may be correlated.

[22]:
```
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(),vmin=-1,vmax=1,cmap='bwr')
```

[22]: <matplotlib.axes._subplots.AxesSubplot at 0x77e7dbae7b50>



From the correlation matrix, it is evident that some attribute pairs are strongly correlated (or

anti-correlated) with each other, while others are not. For example, "follicular_horn_plug" and "follicular_papules" are strongly correlated, and therefore likely to both be associated with class 6 based on the analysis above. To avoid problems associated with the "curse of dimensionality", it is essential not to treat features like these as independent dimensions. However, rather than picking and choosing which features to remove and which to keep, I will use Principal Component Analysis (PCA) for dimensionality reduction.

0.1.4 Data Transformation

Before applying PCA, I want to transform the features so that rare attributes are treated as more significant than common ones. A value of 3 for "spongiform_pustule", for example, should be weighed more heavily in the model than a 3 for "scaling". To achieve this, I will sklearn's StandardScaler(), which transforms the data into Z-values, representing difference from the mean in terms of a multiple of the standard deviation. These transformed features are saved below in the matrix X:

[23]:
```
#scaler = MinMaxScaler()
scaler = StandardScaler()
df_X = data.drop('class',axis=1)
scaler.fit(df_X)
X = scaler.transform(df_X)
yt = data['class']

n_features = X.shape[1]
```

Now I will apply PCA using all of the features:

[24]:
```
pca = PCA(n_components=n_features)
pca.fit(X)
```

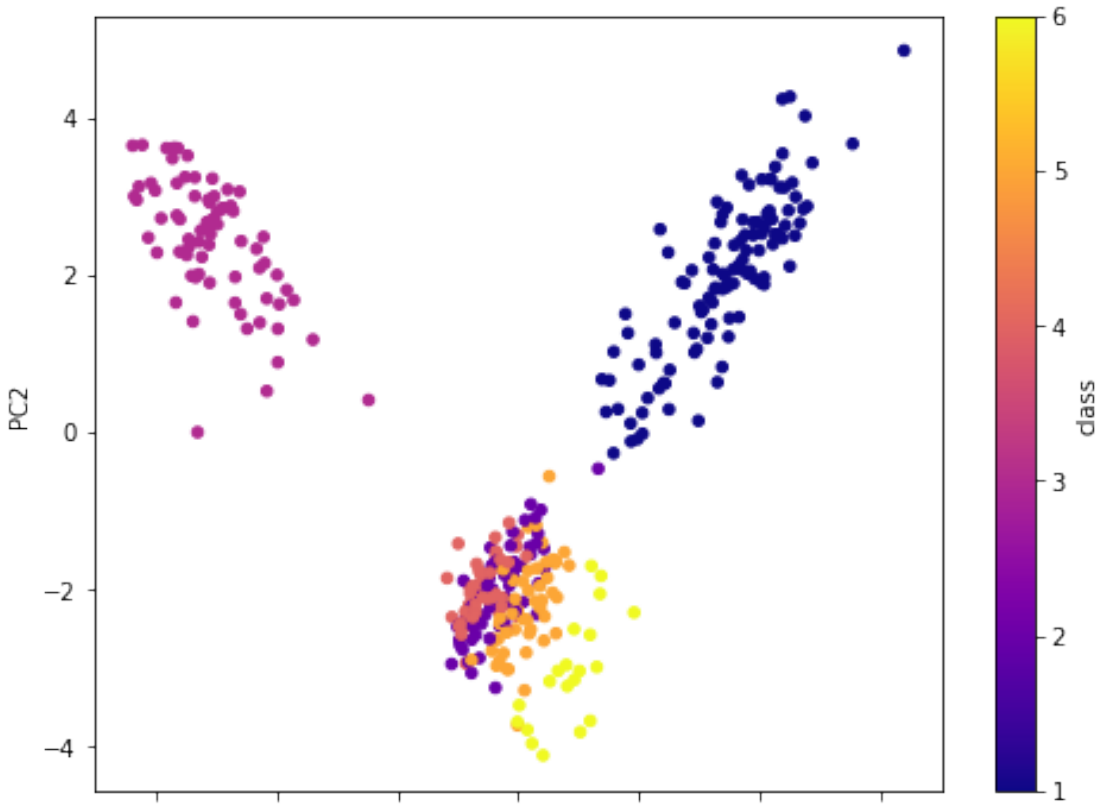
[24]:
```
PCA(copy=True, iterated_power='auto', n_components=33, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

The principal components are saved in the Z matrix, with the most relevant components listed first. Below is a scatter plot of the first two principal components, which together account for 53% of the variance in the data, with class labels indicated by color.

[26]:
```
Z = pca.transform(X)

#Cumulative explained variance
cum_exp_var = np.cumsum(pca.explained_variance_ratio_)
print(cum_exp_var[2])

df_Z = pd.DataFrame({'PC1': Z[:,0], 'PC2': Z[:,1], 'class': data['class']})
df_Z.plot.scatter(x='PC1', y='PC2', c='class', colormap='plasma', figsize=(8,6))
```

```
0.5336700602001634
```

[26]: <matplotlib.axes._subplots.AxesSubplot at 0x77e7dbd58650>



With just these first two principal components, there are already clusters forming which correspond to class labels 1 and 3, but more work is needed to separate labels 2, 4, 5, and 6. Here is a 3D plot using the first three principal components, which account for 60% of the variance.

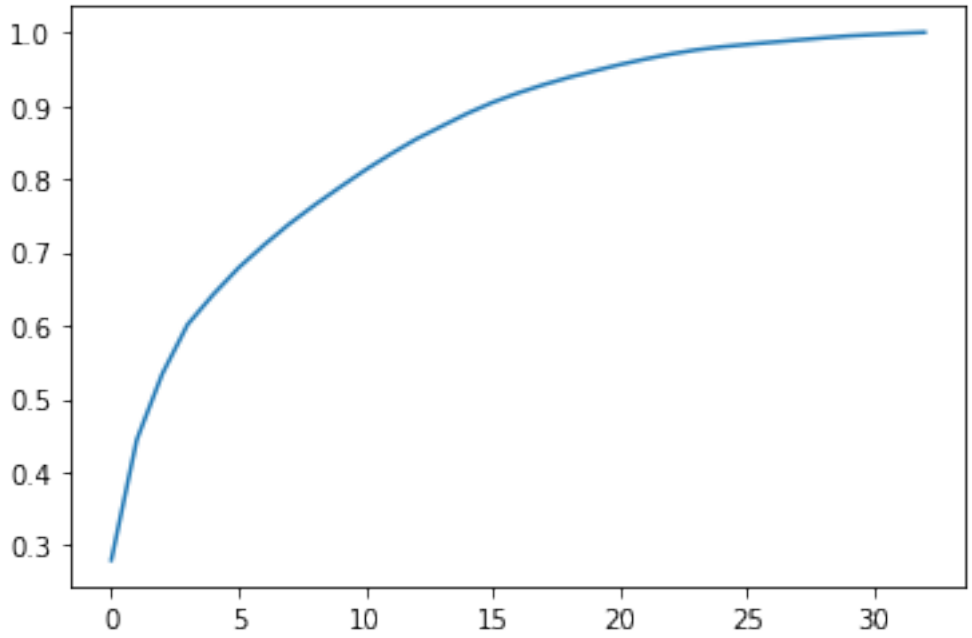
[27]:
```
print(cum_exp_var[3])

df_Z = pd.DataFrame({'PC1': Z[:,0],
                     'PC2': Z[:,1],
                     'PC3': Z[:,2],
                     'class': data['class']})

fig = px.scatter_3d(df_Z, x='PC1', y='PC2', z='PC3',color='class')
fig.show()
```

```
0.6013064384036104
```

Now we seee class 6 starting to form its own cluster, but classes 2, 4, and 5 remained tightly bunched. To get an idea of how many components might be needed, I will plot the number of components vs. cumulative explained variance:

[31]:
```
plt.plot(range(0,33), cum_exp_var)
```

[31]:
```
[<matplotlib.lines.Line2D at 0x77e7d11f5c90>]
```



0.1.5 Model Training and Evaluation

Based on the results of the scatterplots above, a clustering algorithm is suitable for this problem. Because it is an unsupervised learning algorithm that doesn't use training labels to train, there is no need to do a train-test split. The first model I will try is the K-Means Clustering algorithm, an unsupervised learning algorithm that separates data into a specified number of clusters. I will choose 6 for the number of class labels, and to start, include all of the features.

[36]:
```
kmeans = KMeans(n_clusters=6, random_state=42).fit(Z)

#Resulting labels from clustering
y_lab = kmeans.labels_
```

The ordering of the cluster labels does not necessarily matching the ordering of the training labels. To figure out how to map cluster numbers to class numbers, the function below tests all permutations of the class label order and chooses the one with the highest accuracy.

[37]:
```
def get_accuracy(yt, yp):
    return np.sum(yt == yp)/len(yp)

def label_order(yt, y_lab):
```

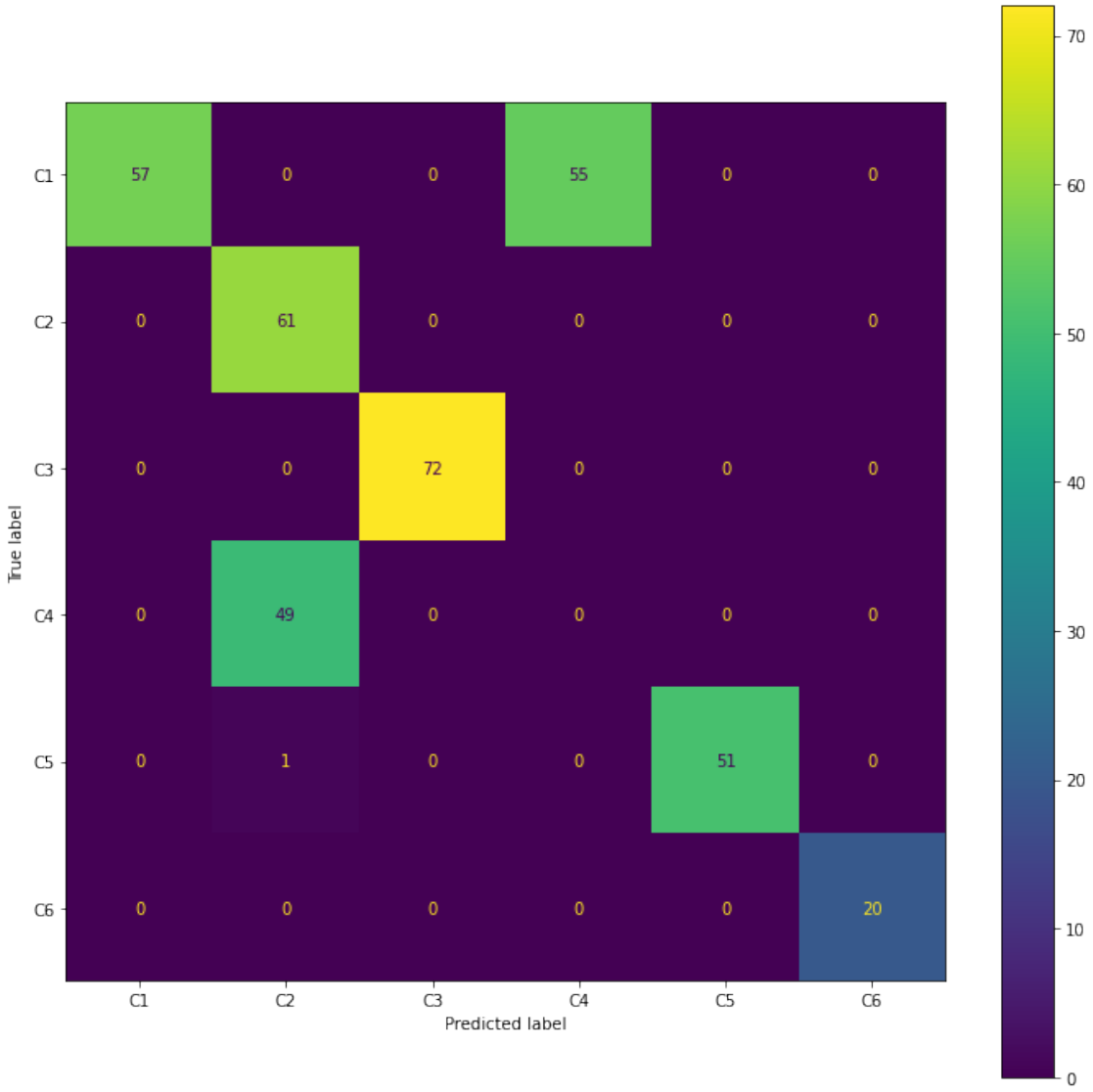
```
    n = len(yt)
    perms = list(permutations(range(1,7))) #Class labels are from 1-6

    acc_max = 0
    order = perms[0]

    for p in perms:

        yp = [p[int(y)] for y in y_lab]
        acc = get_accuracy(yt, yp)

        if acc > acc_max:
            acc_max = acc
            order = p

    return (order, acc_max)
```

[38]:
```
(order, accuracy) = label_order(yt, y_lab)
yp = [order[int(y)] for y in y_lab]
print(accuracy)
```

```
0.7131147540983607
```

This first model achieves an accuracy of only 71%. To see what went wrong, let's have a look at the confusion matrix, which reveals how the incorrect results were misclassified.

[39]:
```
cfm = confusion_matrix(yt, yp, normalize=None)
disp = sklearn.metrics.
 ↪ConfusionMatrixDisplay(cfm,display_labels=['C1','C2','C3','C4','C5','C6'])
fig, ax = plt.subplots(figsize=(12,12))
disp.plot(ax=ax)
plt.show()
```



The confusion matrix shows an uneven distribution of errors. The true labels for classes 2, 3, 5, and 6 are all classified correctly, but the labels for class 1 are split between 1 and 4 and class 4 is entirely misclassified as class 2. What seems to have occurred is that classes 2 and 4 were clustered together and class 1 was broken up into two clusters. This looks like a plausible outcome based on the 3D plot above.

Including all of the features in the model likely led to some overfitting, and defeats the purpose of PCA. To refine the model, I will adjust the number of PCA components included in the model as a hyperparameter.

[40]:
```
def k_means_model(Z_n):

    kmeans = KMeans(n_clusters=6, random_state=42).fit(Z_n)
    y_lab = kmeans.labels_
```

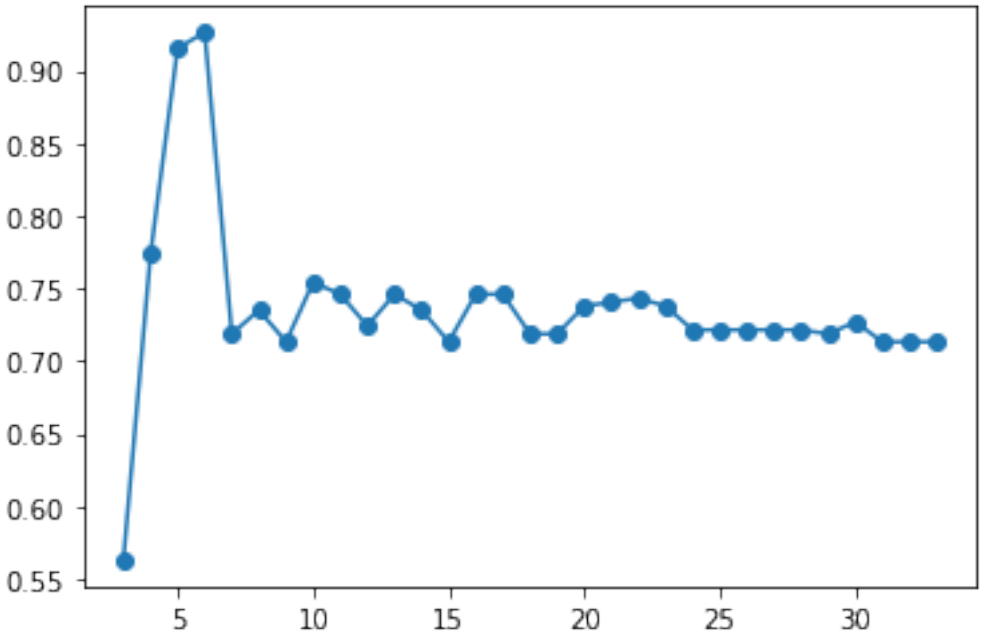
```
    (order, accuracy) = label_order(yt, y_lab)
    yp = [order[int(y)] for y in y_lab]
    return(yp, accuracy)
```

[41]:
```
#Calculated model accuracy based on number of components
acc = []
for n in range(3,n_features+1):
    Z_n = Z[:,:n]
    acc.append(k_means_model(Z_n)[1])
```

Here is a plot of the accuracy of the model vs. number of components included. The optimal value is 6, with an accuracy of 92.6%.

[42]:
```
plt.plot(range(3,n_features+1),acc,marker='o')
print(np.argmax(acc),np.max(acc))
```

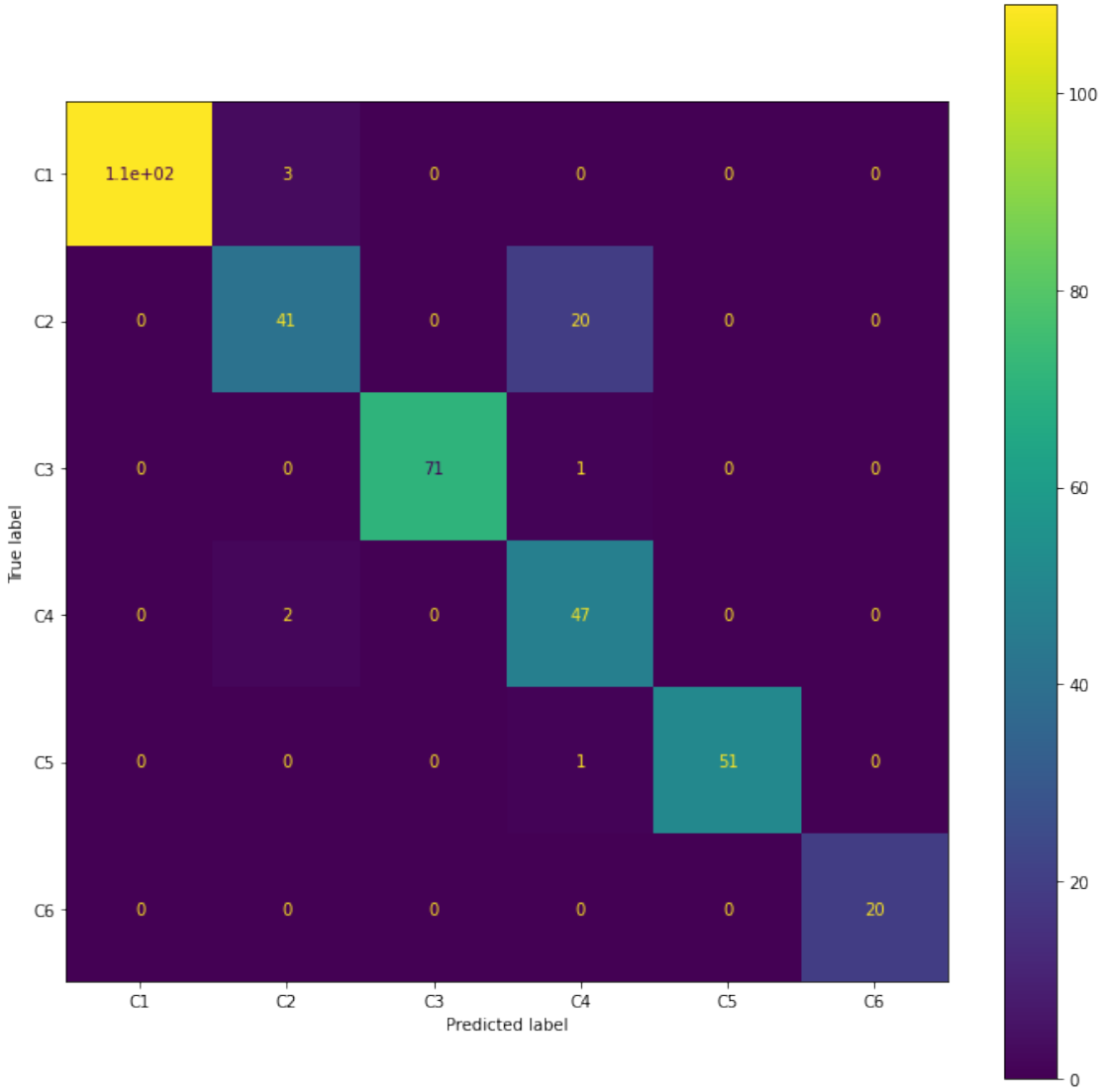
```
3 0.9262295081967213
```



Let's have a look at the confusion matrix for this version of the model. The are fewer errors, but there is still an issue of overlap between classes 2 and 4 which is mostly on one side of the diagonal. Class 2 has high precision (true positive rate) and low specificity (true negative rate), while Class 4 has high specificity and low precision.

[43]:
```
yp = k_means_model(Z[:,:6])[0]
cfm = confusion_matrix(yt, yp, normalize=None)
```

```
disp = sklearn.metrics.
 ↪ConfusionMatrixDisplay(cfm,display_labels=['C1','C2','C3','C4','C5','C6'])
fig, ax = plt.subplots(figsize=(12,12))
disp.plot(ax=ax)
plt.show()
```



Next, I am going to try a different approach and build a Hierarchical Clustering model. This type of algorithm has more hyperparameters to tune and therefore presents more opportunities for optimization.

The model I will be tuning is sklearn's AgglomerativeClustering model, a "bottom-up" approach which iteratively merges clusters based on distance from each other. Two of the most important hyperparameters for this model are the distance metric - method of measuring distance between clusters - and linkage, which determines the point within a cluster to measure from. I will perform

a parameter search accross five different distance metrics and four linkage types, as well as the number of PCA components as a third hyperparameter.

[309]:
```
Metrics=['euclidean', 'l1', 'l2', 'manhattan', 'cosine']
Linkages=['ward', 'complete', 'average', 'single']
N = range(3,n_features+1)

def h_clustering_model(Metric, Linkage, n_components):

    model = AgglomerativeClustering(n_clusters=6, affinity=Metric,␣
 ↪linkage=Linkage)
    model.fit_predict(Z[:,:n_components])
    y_lab = model.labels_

    (order, accuracy) = label_order(yt, y_lab)
    yp = [order[int(y)] for y in y_lab]
    return(yp, accuracy)

best_acc = 0

for m in Metrics:
    for l in Linkages:
        for n in N:

            if l == 'ward' and m != 'euclidean':
                continue

            yp, accuracy = h_clustering_model(m, l, n)

            if accuracy > best_acc:
                best_acc = accuracy
                best_params = [m,l,n]

print(best_acc, best_params)
```

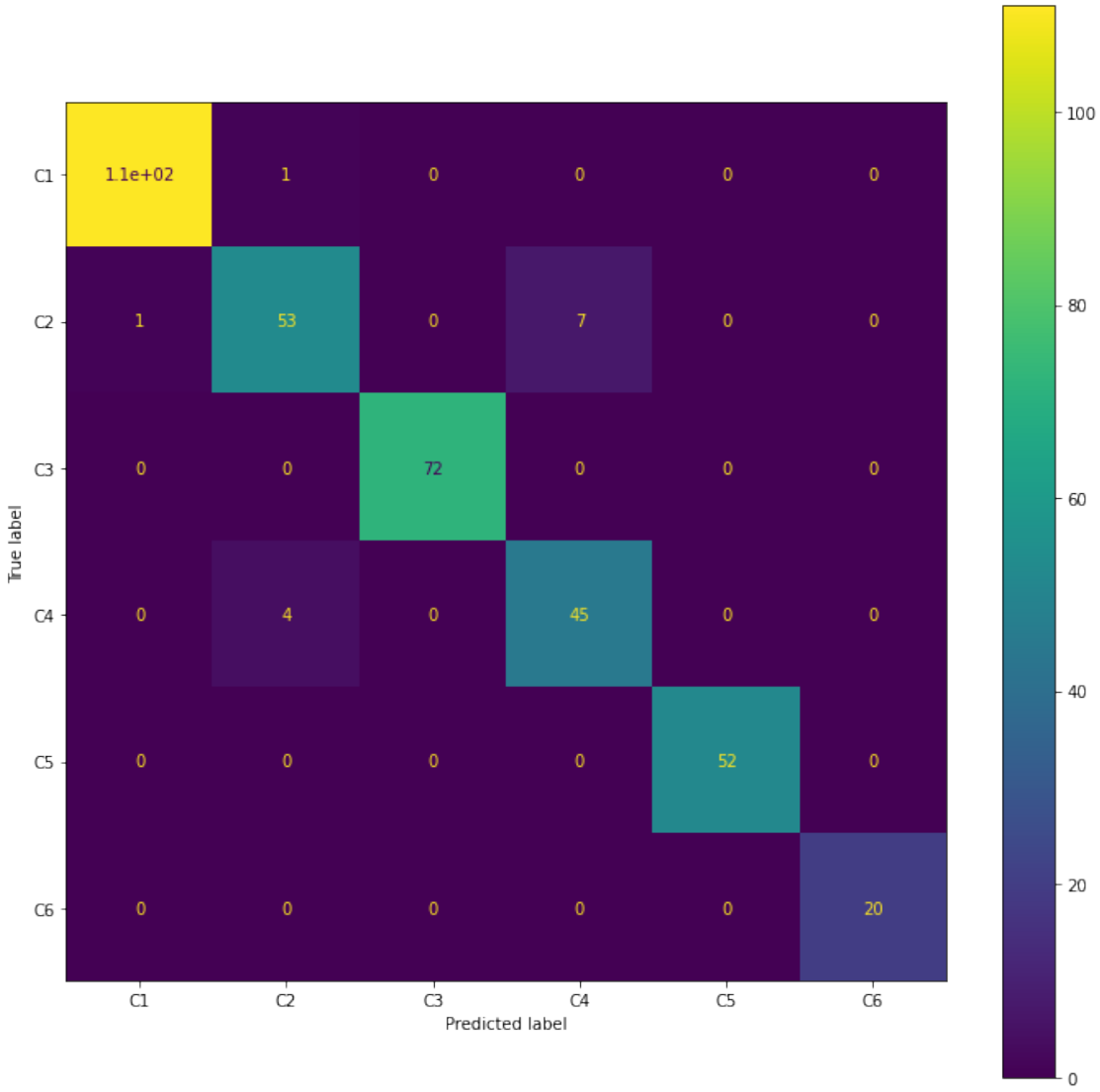
```
0.9644808743169399 ['euclidean', 'ward', 6]
```

The parameter search returns a combination of Euclidean distance, "Ward" linkage method (a method based on minimizing the variance within clusters), and 6 components. This model is more accurate than K-Means, with an accuracy score of 96.4%.

Here is the confusion matrix:

[310]:
```
yp = h_clustering_model('euclidean', 'ward', 6)[0]

cfm = confusion_matrix(yt, yp, normalize=None)
disp = sklearn.metrics.
 ↪ConfusionMatrixDisplay(cfm,display_labels=['C1','C2','C3','C4','C5','C6'])
fig, ax = plt.subplots(figsize=(12,12))
```

```
disp.plot(ax=ax)
plt.show()
```



These results are encouraging, as not only are there fewer errors, but the errors are more balanced on both sides of the diagonal, indicating that the classes are more equal in terms of precision and specificity.

0.1.6 Discussion and Conclusion

In this project, I have demonstrated that types of skin diseases can be classified by an unsupervised agglomerative clustering model to a high degree of accuracy. The model worked nearly perfectly for four out of six categories, with the two remaining ones proving more difficult to separate. One important takeaway from this project is the importance of dimensionality reduction for clustering

algorithms. Both K-Means and Hierarchical Clustering worked best using only 6 components - engineered for maximum significance using PCA - out of 33 total features. This is beneficial for reducing time complexity as well as improving accuracy.

One possible limitation of this project is the limited scope of the data. Every patient sampled had one of the six diseases that we were trying to predict; there were no examples of patients having none of the diseases or more than one of the diseases. In addition, we don't know how representative the 366 patients included in this dataset are of the general population, so it is unclear how generalizable the model will be.

One potential way to improve the results would be to do a more rigorous analysis of the correlations between variables and removing or combining some of them prior to transforming the data. It might also help to have more information about the patient - information such as weight, diet, drug use, etc.

GitHub Link: https://github.com/edejongh1/USL-Final-Project

Data Citation

OLCAY_BOLAT. 2023. "Dermatology Dataset (Multi-class classification)". Retrieved from https://www.kaggle.com/datasets/olcaybolat1/dermatology-dataset-classification/data

[ ]: