Exercise Set 3

Problem 17

Task a

We can use k-means for clustering, grouping into k-amount of groups data based on similarities. K-means works by finding a set of centroids for these clusters, so that the loss function is minimized. Lloyd's alorithm is one way to calculate the centroids for k-means. It works by assigning data points to random clusters. After that it sets each cluster centroid to average of the points in the cluster and assigns each data point to the closest centroid, until convergence.

Inputs for the algorithm are the data and k (n# of clusters). Output is the clustering. The results are an approximation of the true clustering.

Task b

Cost function:

$$L_{kmeans} = \sum_{i=1}^{n} \| x_i - \mu_{j(i)} \|_2^2$$

When we assign the point i to the closest centroid we are minimizing $\| x_i - \mu_{j(i)} \|_2^2$. When we choose μ_j to be the mean of the points we are minimizing $\| x_i - \mu_j \|_2^2$. The algorithm is finite and will stop, usually sooner than later.

Task c

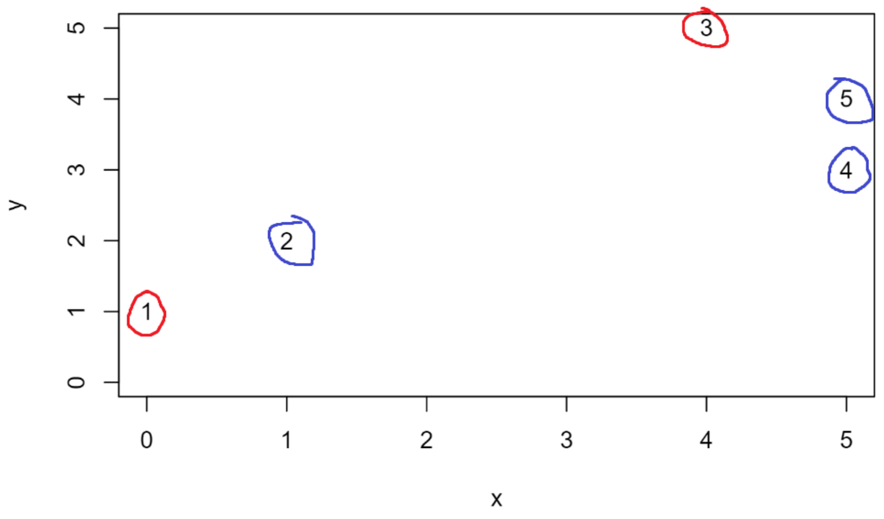


Figure 1: Assing data points to random clusters

Now we set every centroid to the average of the points in the cluster. We can calculate this by

$$x_m = \frac{1}{n}\sum_i x_i \quad y_m = \frac{1}{n}\sum_i y_i$$

In figure 2, the red centroid is now positioned in $(2, 3)$, since this is the average of $(0, 1)$ and $(4, 5)$, and the blue centroid is positioned in $(3, 3)$, since this is the average of $(1, 2)$, $(5, 3)$ and $(5, 4)$.

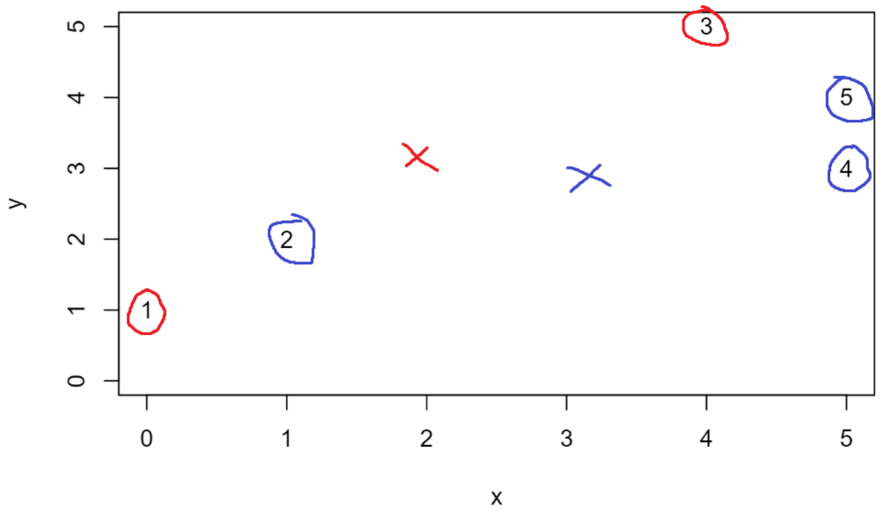


Figure 2: First step

Now we assing the points to the nearest centroid. Distance between a point and a centroid can be calculated by:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

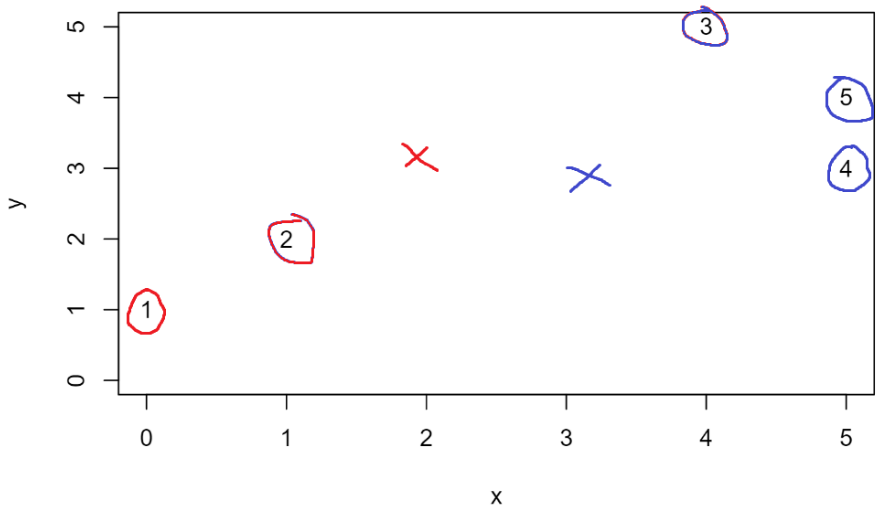


Figure 3: Second step

The new centroids are $\mu_{red} = (0.5, 1.5)$ and $\mu_{blue} = (4.5, 4)$.

Next step converges and stops the algorithm.

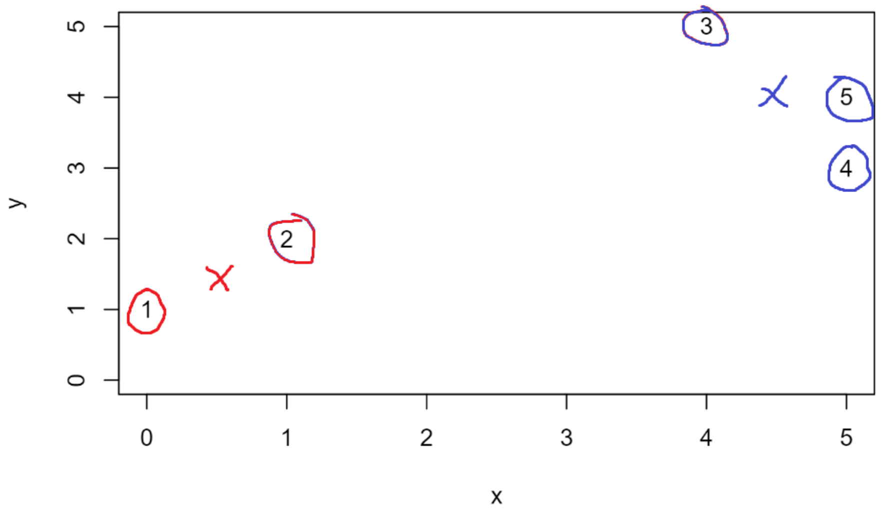


Figure 4: Third step

Problem 18

Task a

The algorithm works as follows: 1. clustering of 3 and 4, with a joint cost of 1.36 2. clustering of 6 and 7, with a joint cost of 1.53 3. clustering of 1 and 2, with a joint cost of 2.24 4. clustering of 5 and (1,2), with a joint cost of 2.35, that is the distance between 2 and 5 5. clustering of (5, (1,2)) and (6,7), with a joint cost of 2.42, that is the distance between 5 and 6 6. clustering of 2.66, that is the distance between 4 and 6

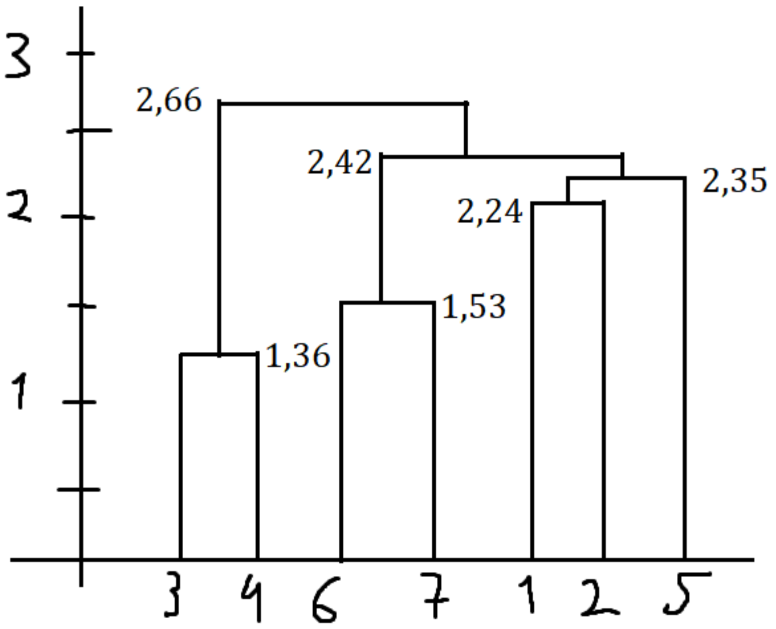
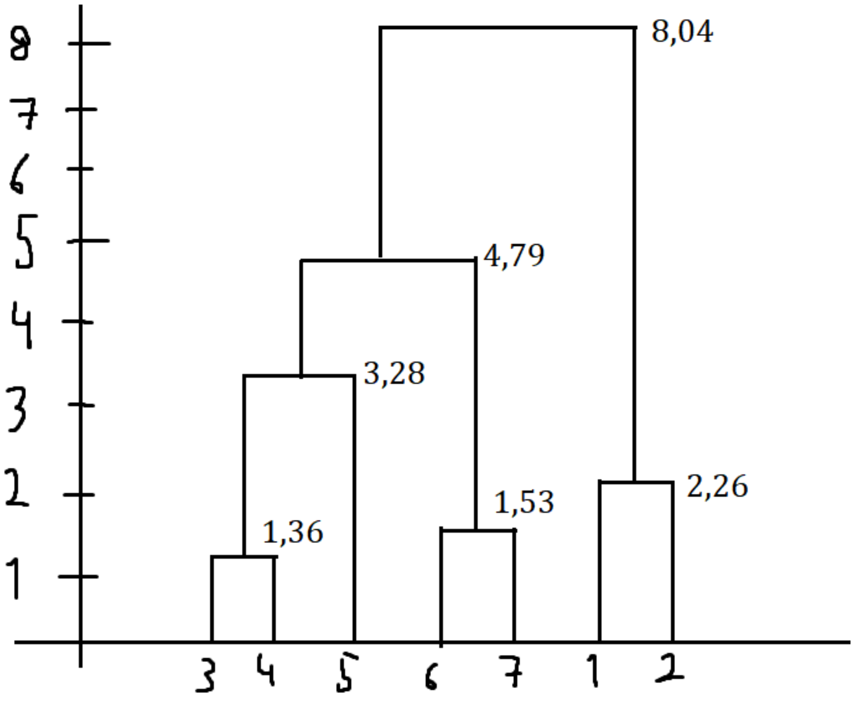


Figure 5: Dendogram

Task b



The algorithm works as follows: 1. clustering of 3 and 4, with a joint cost of 1.36 2. clustering of 6 and 7, with a joint cost of 1.53 3.

clustering of 1 and 2, with a joint cost of 2.24 4. clustering of 5 and (3,4), with a joint cost of 3.28, that is the distance between 4 and 5 5. clustering of (5, (3,4)) and (6,7), with a joint cost of 4.79, that is the distance between 3 and 7 6. clustering with a joint cost of 8.04, that is the distance between 1 and 7

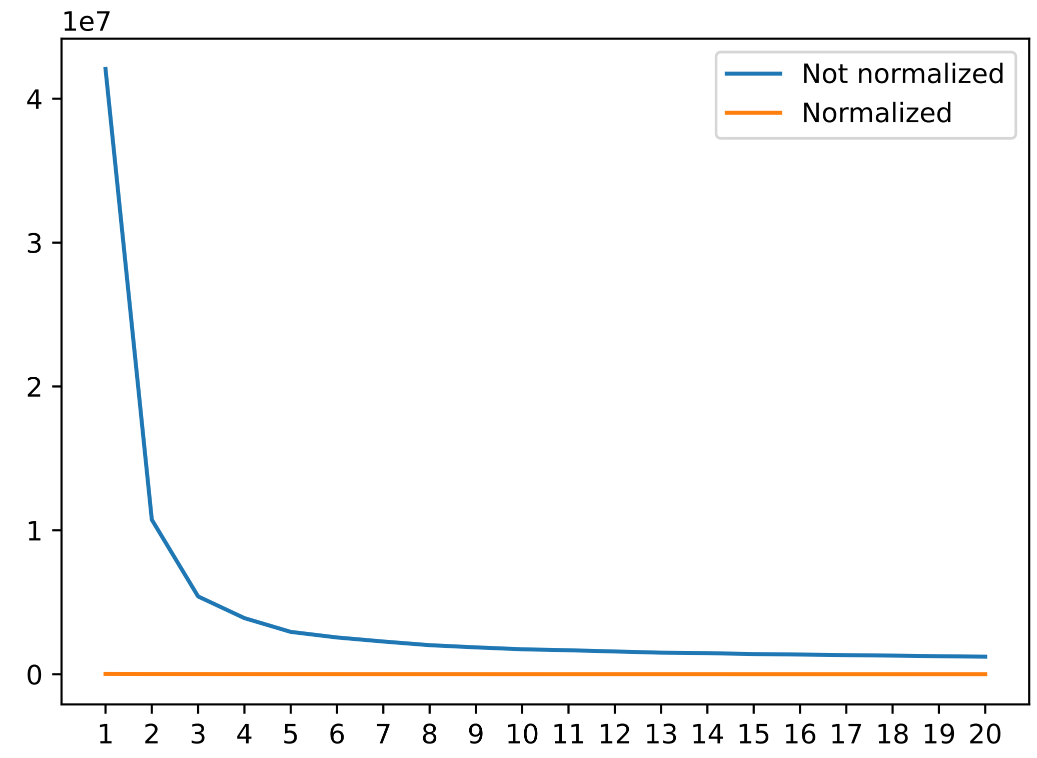
Problem 19

Task a

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt

npf = pd.read_csv("npf_train.csv")
del npf['id']
del npf['partlybad']
del npf['date']
y = npf['class4'].astype('category')

npf = npf[[c for c in npf if 'mean' in c]].rename(columns= lambda x: x.replace('.mean', 'mean'))

npf = pd.concat([npf, y], axis=1)
colss = npf.columns.difference(['class4'])
scaled = preprocessing.scale(npf[npf.columns.difference(['class4'])])

npf['class4'] = npf['class4'].cat.codes
```

Plotting:

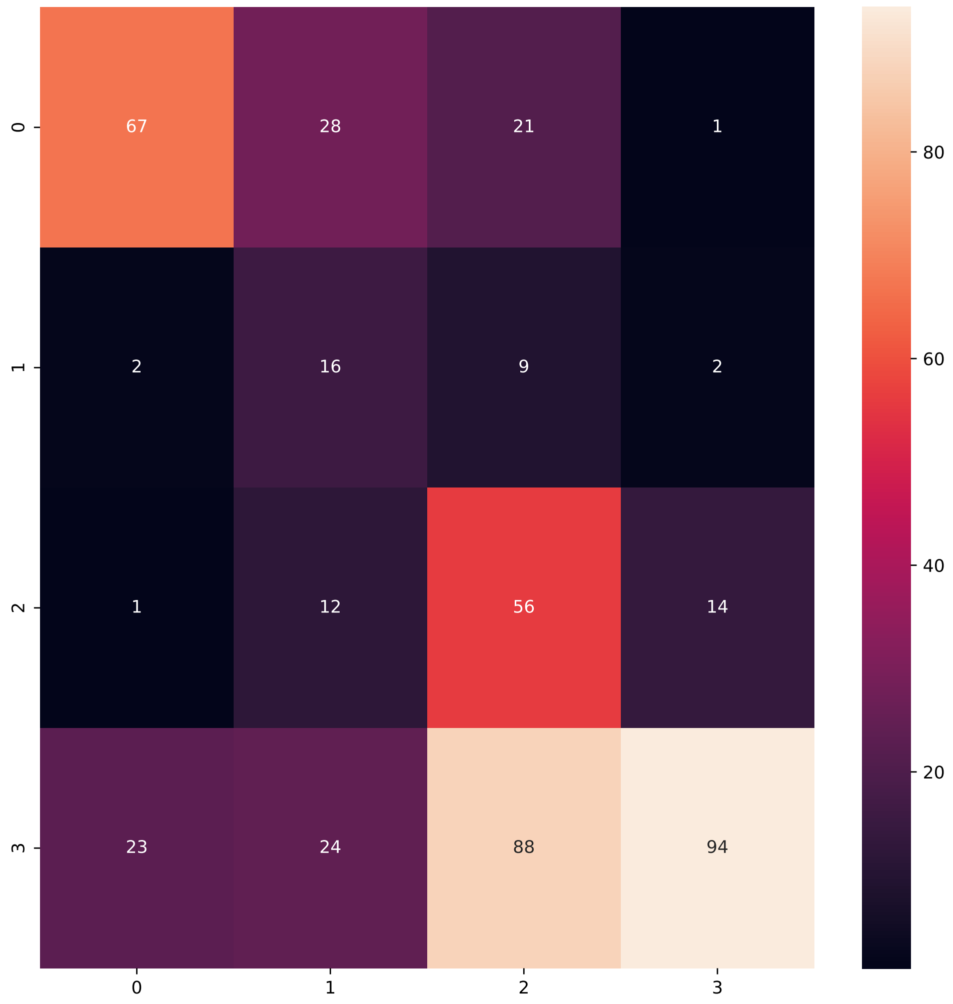
```
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings('ignore')

toplotY_un = []
toplotX_un = []
toplotX_sc = []
toplotY_sc = []
for i in range(1, 21):
    km = KMeans(n_clusters=i).fit(npf[npf.columns.difference(['class4'])])
    toplotY_un.append(-1*km.score(npf[npf.columns.difference(['class4'])]))
    toplotX_un.append(str(i))

    km = KMeans(n_clusters=i).fit(scaled)
    toplotY_sc.append(-1*km.score(scaled))
    toplotX_sc.append(str(i))

plt.plot(toplotX_un, toplotY_un, label='Not normalized')
plt.plot(toplotX_sc, toplotY_sc, label='Normalized')
plt.legend()
plt.show()
```



Normalization made impact to the result. On the normalized data the loss was very minimal (not probably even showing on the plot) compared to the huge loss on the non-normalized data. Non-normalized data loss started to fall of drastically the more number of clusters there is but still the loss was huge.

Task b

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

km = KMeans(n_clusters=4).fit(scaled)
y_pred = km.predict(scaled)
y_true = npf['class4']

cm = confusion_matrix(y_true, y_pred)

new_cm = [[67, 28, 21, 1], [2,16,9,2], [1,12,56,14], [23,24,88,94]]
plt.figure(figsize=(10,10))
sns.heatmap(new_cm, annot=True)
plt.show()
```



From this matrix it's pretty clear that the most errors happen were the values outside the diagonal are higher. This is particularly true for the last row: the first, second and third values are in fact wrong "predictions" with a really high count.

Task c

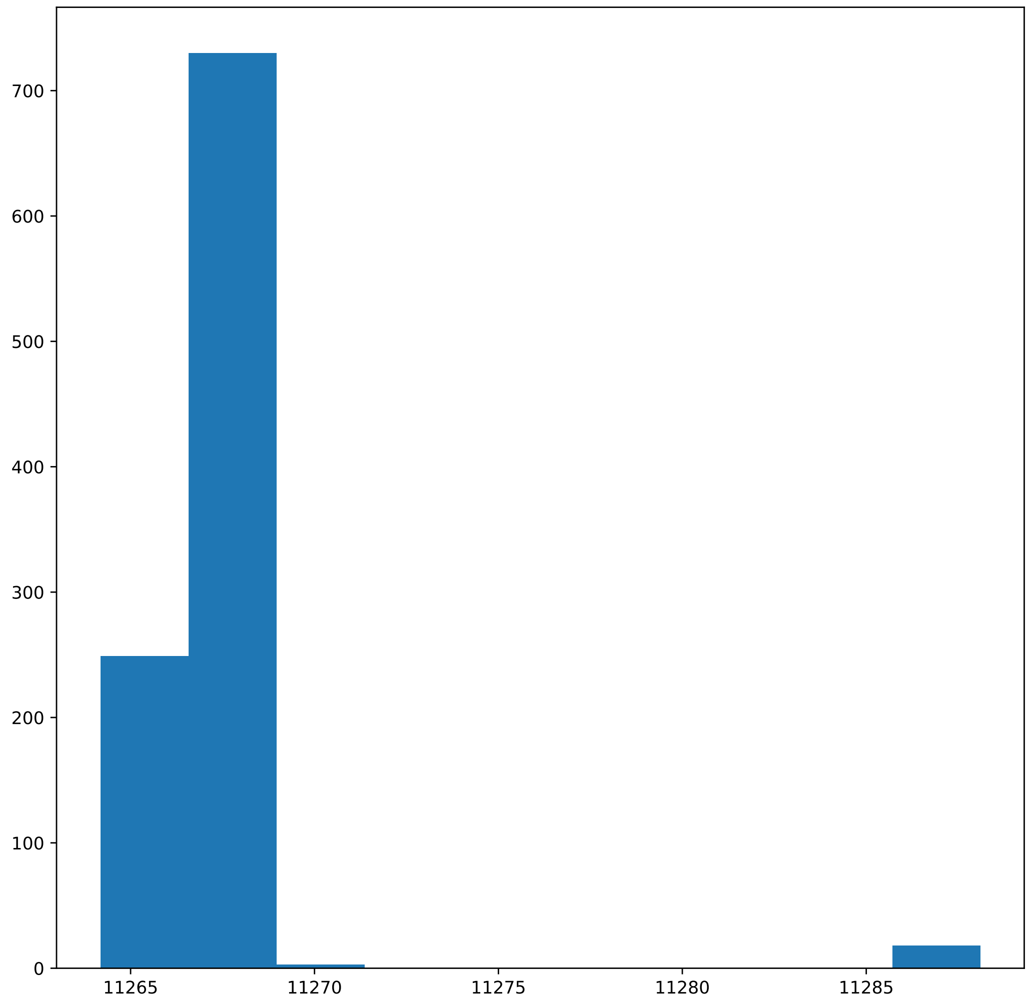
```
import sys

toplotX = []
toplotY = []
min = float('inf')
max = 0
```

```
for i in range(1, 1001):
    km = KMeans(n_clusters=4, init='random').fit(scaled)
    l = -1*km.score(scaled)
    toplotY.append(l)
    toplotX.append(i)
    if l < min:
        min = l
    if l > max:
        max = l
    sys.stdout.flush()
```

```
plt.clf()
plt.hist(toplotY)
```

```
## (array([249., 730.,   3.,   0.,   0.,   0.,   0.,   0.,   0.,  18.]), array([11264.18219338, 11266.5
##        11273.74883824, 11276.14049945, 11278.53216066, 11280.92382188,
##        11283.31548309, 11285.7071443 , 11288.09880552]), <BarContainer object of 10 artists>)
```

```
plt.show()
```



```
print(min, max)
```

```
## 11264.182193382203 11288.09880551541
```

The minimum loss I got was 11264.182193382203, and the maximum 11290.868518678522. The minimum and the maximum are close to each other so optimal result could be achieved with only a small amount of executions.

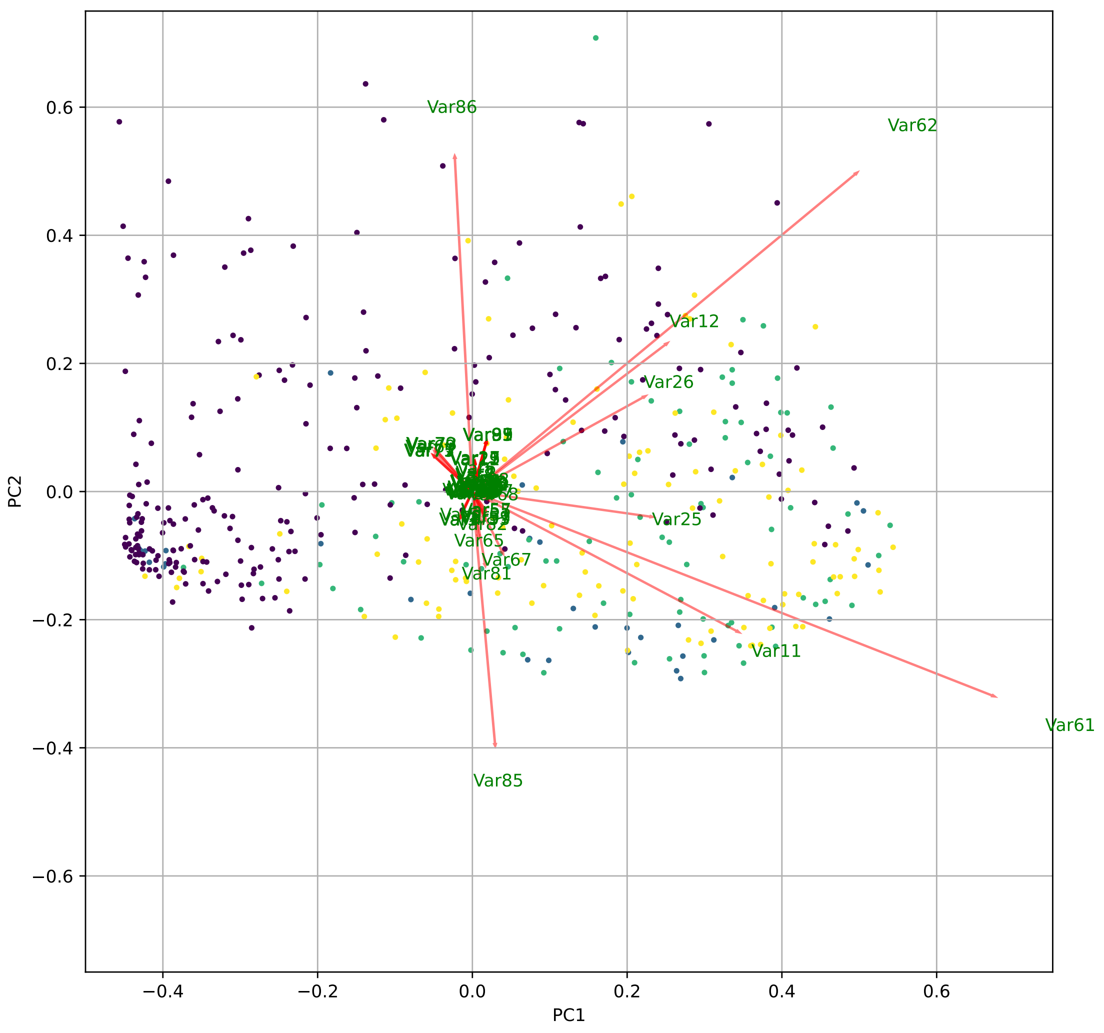
Problem 20

Task a

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt

def change_lab(x):
    if x == 'nonevent': return 0
    if x == 'Ia': return 1
    if x == 'Ib': return 2
    return 3

npf = pd.read_csv('npf_train.csv')
del npf['id']
del npf['partlybad']
del npf['date']
npf['class4'] = npf['class4'].apply(lambda x: change_lab(x))


scaled = preprocessing.scale(npf[npf.columns.difference(['class4'])])

X = npf[npf.columns.difference(['class4'])]
y = npf['class4']
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

def biplot(score,coeff,labels=None):
    xs = score[:,0]
    ys = score[:,1]
    n = coeff.shape[0]
    scalex = 1.0/(xs.max() - xs.min())
    scaley = 1.0/(ys.max() - ys.min())
    plt.scatter(xs * scalex,ys * scaley, c = y, s=5)
    for i in range(n):
        plt.arrow(0, 0, coeff[i,0], coeff[i,1],color = 'r',alpha = 0.5)
        if labels is None:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'g', ha = 'center', va
        else:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i], color = 'g', ha = 'center', va = '
    plt.xlim(-0.5,0.75)
    plt.ylim(-.75,.75)
    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))
    plt.grid()

model = PCA()
x_new = model.fit_transform(X)
```

```
plt.figure(figsize=(10,10))
biplot(x_new[:,0:2],np.transpose(model.components_[0:2, :]))
plt.show()
```

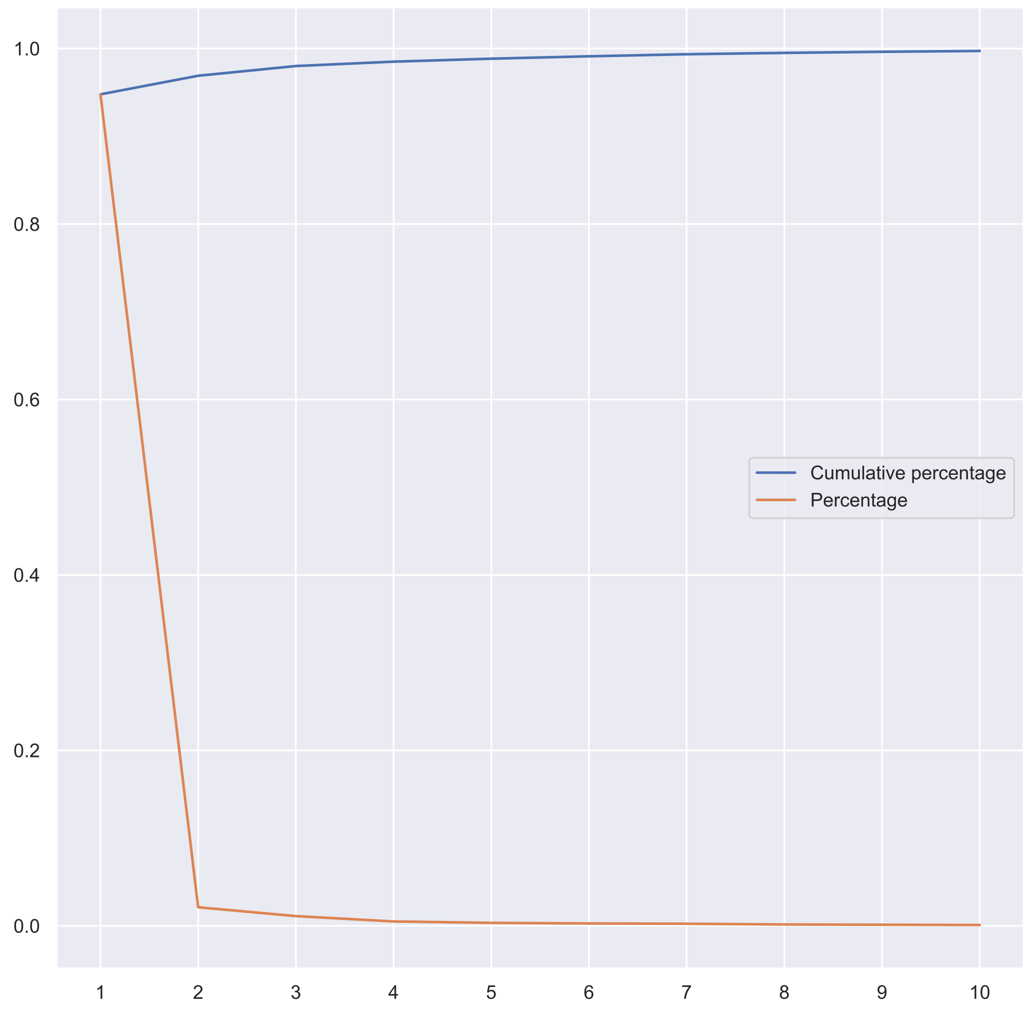


Purple points are `nonevent`, green are `Ia`, yellow are `Ib` and blue are `II`.

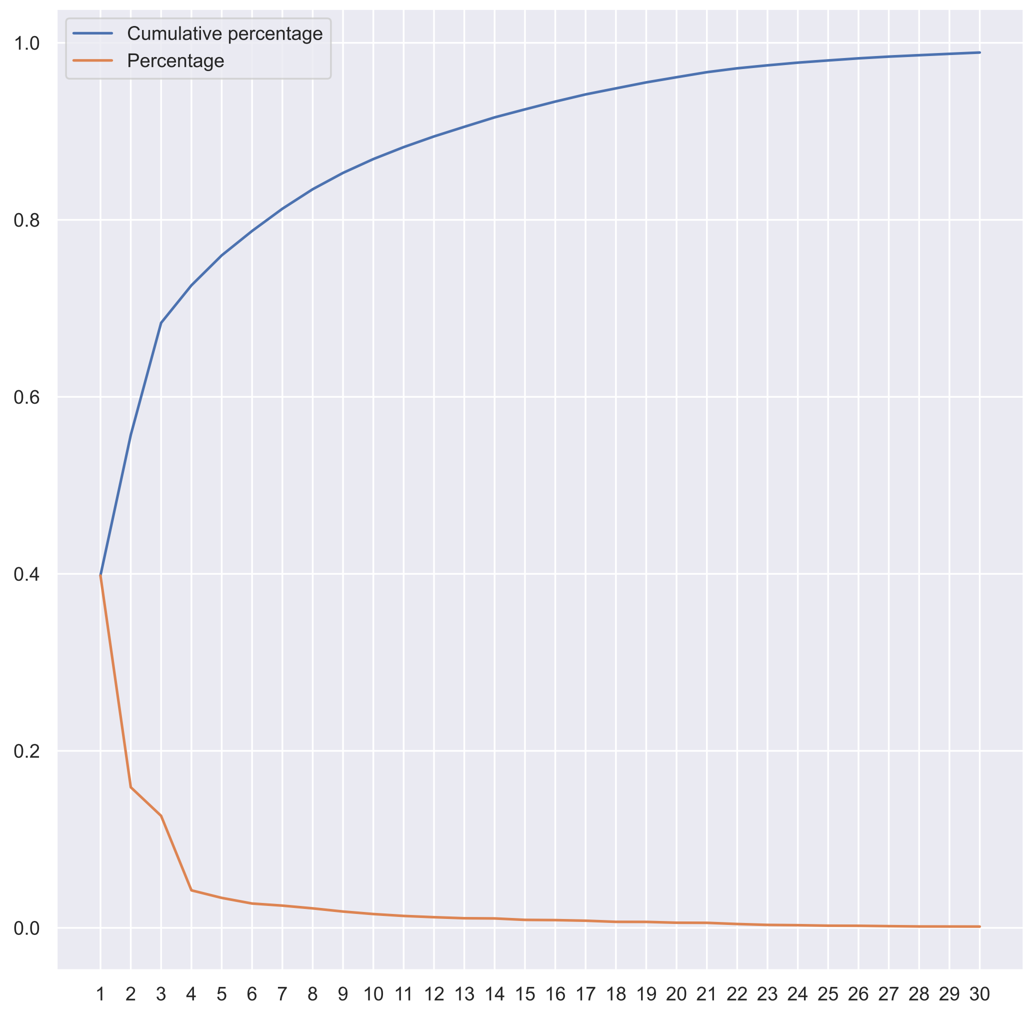
Task b

```
plt.clf()
x = []
y_down = []
y_up = []
```

```python
sum = 0

pca = model.fit(X)

for i in range(10):
    x.append(str(i+1))
    y_down.append(pca.explained_variance_ratio_[i])
    sum += pca.explained_variance_ratio_[i]
    y_up.append(sum)

sns.set(rc={'figure.figsize':(9,6)})
plt.plot(x,y_up, label='Cumulative percentage')
plt.plot(x, y_down, label='Percentage')
plt.legend()
plt.show()
```



```
plt.clf()
x = []
y_down = []
y_up = []
sum = 0

x_scaled = preprocessing.scale(X)

pca = model.fit(x_scaled)

for i in range(30):
    x.append(str(i+1))
    y_down.append(pca.explained_variance_ratio_[i])
    sum += pca.explained_variance_ratio_[i]
    y_up.append(sum)

sns.set(rc={'figure.figsize':(9,6)})
plt.plot(x,y_up, label='Cumulative percentage')
plt.plot(x, y_down, label='Percentage')
plt.legend()
plt.show()
```



In the two scenarios, the changes in the curves are as expected: when the data is not scaled, the first 2/3 features explain most of the variance, while almost all of the variance is explained by the first 30 features when the data is scaled. The high variance of the initial variables in the dataset leads to a significant amount of variance being explained by them. After scaling to a zero mean and unit variance, the variance explained by these variables decreases due to the scaling itself.

Problem 21

I learned about various techniques for unsupervised learning, including clustering, dimensionality reduction, and ensemble methods. I also learned about random forest and support vector machines (SVMs) and how they can be used for classification tasks.

There were a few concepts that I did not fully understand, such as the exact details of how PCA works. I will need to spend more time studying this topics in order to gain a deeper understanding.

Overall, the lectures were very relevant for other studies and future work in the field of machine learning. I, once again, gained a better understanding of how to apply these techniques to different datasets and how to evaluate the performance of the models that I build.