Home Exam Fys-2021

Candidate number: 38

October 2024

Problem 1

In this task we are going to classify if chemicals based on their lipophilicity, and see if they like fat or not.

The first thing we have to do is go to kaggle where the task and data set is, and download the two dataset. One for training and one for testing. Immediately we are shown that there is 230 different columns for the trainings set. Here one is the chemical ID, one is lipophilicity and the rest are features. For the test set there is exactly the same, just no lipophilicity, as that is the one we want to find. As there are 228 features, it is smart to do some data pre processing before we start with training. We can start by excluding all the features that have the same value. We can do this in python if we load the training set in with pandas. Then we can use the following python code to get rid of the features only having one unique value.

```
for i in train_set.keys():
    unique = train_set[f"{i}"].unique()
    if len(unique) <= 1:
        print(i)
        train_set.drop(columns=f"{i}")
```

We have now removed 10 features, but we still have 220 features. We can then try to implement a forward selection to cut down the amount of features. This method goes through all the features and iterative classify the lipophilicity of a validation set one by one. Then calculate the accuracy of each features to select which feature gives the best result, and chooses the one with the highest accuracy. Then it goes through the features again, but this time with the best features from the previous time. This happens until the total accuracy score becomes less than the last. And then we have the features best suited to classify lipophilicity. To implement this, we need to divide the training set into a training and validation set and choose which classifier I am going to use. As a first classifier, I am using the logistic regression. The implementation in python is

```
accuracy = 0
features = []
for i in range(220):
    print(features)
    print(f"accuracy is {accuracy}")
    acc = np.array([])

    for key in test_set.keys(): #using test set keys to not include lipophilicity
        lr = LogisticRegression(max_iter=1000000)
        X_train = train_set_forward.loc[:,features + [f"{key}"]]
        Y_train = train_set_forward[f"{last_value}"]
        x_vali = validation_set.loc[:,features + [f"{key}"]]
        y_vali = validation_set[f"{last_value}"]


        lr.fit(X_train, Y_train)
        y_pred = lr.predict(x_vali)
```

```
        acc = np.append(acc, metrics.f1_score(y_vali, y_pred))

    if acc.max() > accuracy:
        accuracy = acc.max()
        indx = np.where(acc == acc.max())[0][0]
        features.append(validation_set.columns[indx])
    else:
        break
```

I get from this a total of 19 features to be used, and the f1 score is calculated to be 61.8 % for that validation set, which is not too bad. I then perform the training on the whole set with the features i got. Then i perform the classification on the testing set, and submit the answer to kaggle. The score I received was 49.5 % which is a lot worse. This could mean that the training has overfitted the model to the exposed data. One way to fix this is to implement a cross-validation when the data is beeing trained.

I chose to implement a k-fold cross validation while it is beeing trained. This means that the model is being trained k=10 times, and producing 10 different results. Each time, only 9/10 of the training set is beeing used to train the data, this is so some of the data is not ruining the model. In python this becomes

```
def select_training_sets(k, dataframe):
    frames = []
    for _ in range(10):
        frames.append(dataframe.iloc[int(k*len(dataframe)/10) : int((k+1)*len(dataframe)/10), :])

    t = frames.pop(k)
    frames = pd.concat(frames)

    return frames, t


y_pred_all = []
for k in range(10):
    lr = LogisticRegression(max_iter=1000000)
    # X_train = train_set.iloc[int(k*len(train_set)/10) : int((k+1)*len(train_set)/10), :]
    # Y_train = train_set.iloc[int(k*len(train_set)/10) : int((k+1)*len(train_set)/10), :]

    frames, t = select_training_sets(k, train_set)

    X_train = frames.loc[:, features]
    Y_train = frames[f"{last_value}"]
    Y_act = t[f"{last_value}"]
    X_test_acc = t.loc[:, features]
    X_test = test_set.loc[:, features]

    lr.fit(X_train, Y_train)
    y_pred1 = lr.predict(X_test_acc)
    print(metrics.f1_score(Y_act, y_pred1))
    y_pred_all.append(lr.predict(X_test))

y_pred_all = np.array(y_pred_all)

y_pred = np.mean(y_pred_all, axis=0)
y_pred = np.where(y_pred >= 0.5, 1, 0)
```

After this I submitted a new answer to kaggle and got 50.5 % which is an improvement from last time, but not by much. That means it probably was not overfitting from before.

Looking at the implementation of the forward selection we only choose the standard last 20 % to be the validation set. This means that the features chosen will be best suited for this part of the training set and therefore the model will be fitted towards this part of the training set. To change this, I can implement a way to chose the validation set by random each time we go to a new feature. This way, the features chosen will be more

suited towards the whole of the training set, and not just the last 20 % of it. But for this to be most fair, we should run through it 5 times, to get different validation set for the first feature as well, and then combine all the features in the end. We could probably run through it more than 5 times, but since 19 features was chosen the first time, chances are it most likely will choose the first feature at a later point anyway. But if we try it 5 times we get to see if it makes a difference. When doing this I got 28 features to be used, and i got a 47 % on kaggle. This could be a result of having too much features making the model bad again.

I then chose to implement a random forest classifier to be able to see afterwards what features it chose to be the most important, and maybe try them on the logistic regression classifier. If I don´t do any pre processing of the data other than removing the 10 features only having one value, I get a kaggle score of 55 % when i implement the random forest classifier. I then collect the 4 most important features from this run which is MolLogP, TPSA, MinEStateIndex and VSA_EState5. I then rundown the a random forest classifier on those 4 and get a score of 57 % which is my best score so far. Now i can try to use these 4 features in my logistic regression implementation, and only use cross validation when I am training my model. When doing this I get 30 %, which is not that surprising seeing as these four features are the most important for the random forest classifier. Meaning they may not necessarily be the four best features globally. I can now try the forward selection method on the random forest classifier as well. However, it may not be any better, since the random forest classifier can be used to find the best features together, and can not necessarily select the best features to work together when iteratively choosing one by one. But we can try it, to see if there is any difference on the features and how that will impact the score. I now got 55 % which is the same as the first time, which was expected. Now it has run through the forward selection 5 times and gathered a rather large amount of features, but most of them are the same as the ones used before. Therefore the score results in the same as it did before this was done.

Since the random forest classifier worked a lot better than the logistic regression, the last method I will use will be the decision tree classifier. This has a lot in common with random forest classifier, as the random forest classifier is build up with many small decision trees. However, it could be interesting so see the difference in performance with the same features, and what features it choses with the forward selection as opposed to the random forest classifier. I start by using the 4 features that the random forest classifier said was most important. That being MolLogP, TPSA, MinEStateIndex and VSA_EState5. With these 4 features i get a kaggle score of 45 %. This was a bit lower than i expected, so i ran through it with all the features and got the 4 most important features afterwards. I then got the features MolLogP, BCUT2D_LOGPHI, PEOE_VSA13, fr_COO, which are different from the ones I got from random forest tree classifier. The only one that was the same was MolLogP, which was the most important one for both classifiers. To see if these features were to repeat them selves, I ran through the decision tree classifier, with the forward selection method, and now I got a total of 19 features when I ran it 5 times. But the four most used features were the same four as last time. However, I had now gotten a score of 50 % of kaggle, which means that we had sorted out some of the features that messed up with the decision tree classification.

In the end I also tried to use a voting classification. This is a classification that uses the results from the other three classification, and determines what is the best to classify each point by counting what each of them had classifide that point as. This is the voting hard method. The following python code is how i implemented it

```
#importing neccesary packages
import pandas as pd
import numpy as np
```

```
#loading the data sets
lr = pd.read_csv("home_exam/logistic_regression/submission.csv")
rfc = pd.read_csv("home_exam/randomForestTree/submission.csv")
dtc = pd.read_csv("home_exam/DecisionTree/submission.csv")

lr_pred = lr["lipophilicity"].to_numpy()
rfc_pred = rfc["lipophilicity"].to_numpy()
dtc_pred = dtc["lipophilicity"].to_numpy()

y_pred_all = np.array([lr_pred, rfc_pred, dtc_pred])


y_pred = np.mean(y_pred_all, axis=0)
y_pred = np.where(y_pred >= 0.5, 1, 0)



out_file = pd.DataFrame({"Id": lr["Id"], "lipophilicity": y_pred})
out_file.to_csv("home_exam/VotingClassifier/submission.csv", index=False)
```

I chose to download the results from the three other classification and read them in as csv files. With this I tried one last time one kaggle, and received a score of 55 %. This was somewhat lower than what I had thought before I submitted it. However, it was only the random forest classification that got over this, and the other two did not even get close to this. So they probably made it worse.

Problem 2

2a)

Clustering is an unsupervised learning technique that groups data without having labels to the data. Meaning that it will be used when we want to uncover natural groupings within the data. As opposed to classification which want to classify a label based on patterns learned from labeled data. Clustering can be used when studying customer trends within marketing. Here we can uncover patterns without having predefined categories, allowing for targeting marketing without constraining to the categories.

2b)

The k-mean algorithm can be divided into 5 steps. 1. choose k. Here we want to choose a appropriate k value, that fits to the selected data. The k will be the amount of clusters, and changing k will dramatically change the outcome of the algorithm. 2. initializiation. Here we want to choose the initial condition fo the algorithm, which is the centroid of each cluster. This will be done at random. 3. Find the distance. Calculate the distance between each data point and each centroid, and assign the data point to the cluster with the nearest centorid. 4. calculate mean values. Calculate the mean values for all the datapoints in each cluster. Set the new centroid to the mean values for that cluster. 5 repeat step 3 and 4 until a final condition is met. This final condition can either be a maximum amount of iterations or when the updated centroids is equal to the last one.

2c)

In figure 1 we can see the 25 first images of the frey face csv file. From this we can see that there appears to be the same person on all of the faces. Therefore it is likely that we will cluster the different facial expressions, as smiling, serious, sticking his tounge out and other.

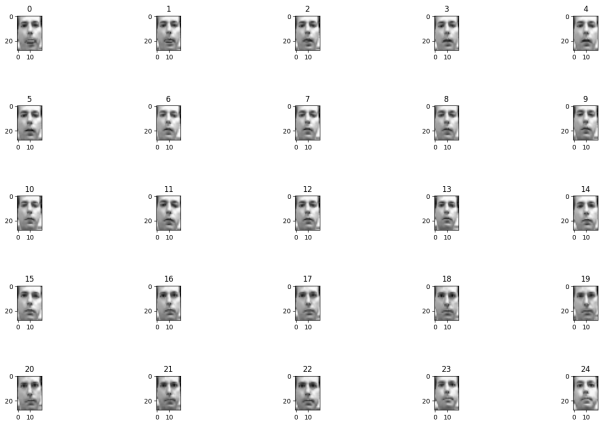

Figure 1: The first 25 images of the frey faces file

2d)

The code used for this problem can be found in appendix A. Figure 2, 3 adn 4 we can see the plots I got from the k means cluster algorithm.

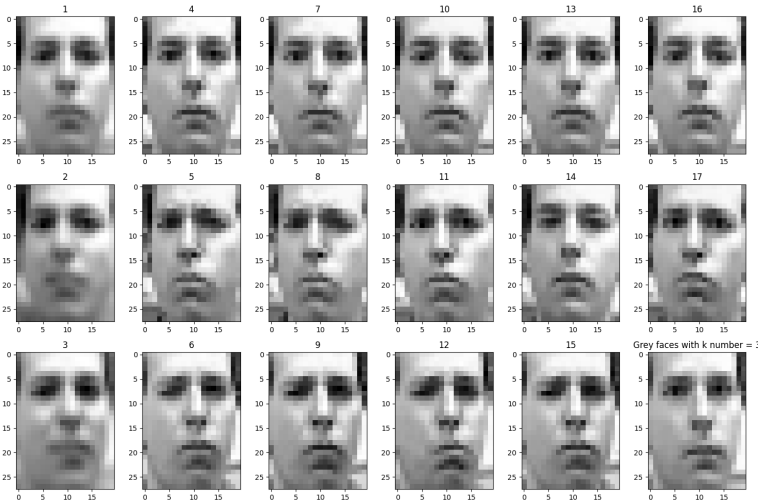

Figure 2: results of k = 3

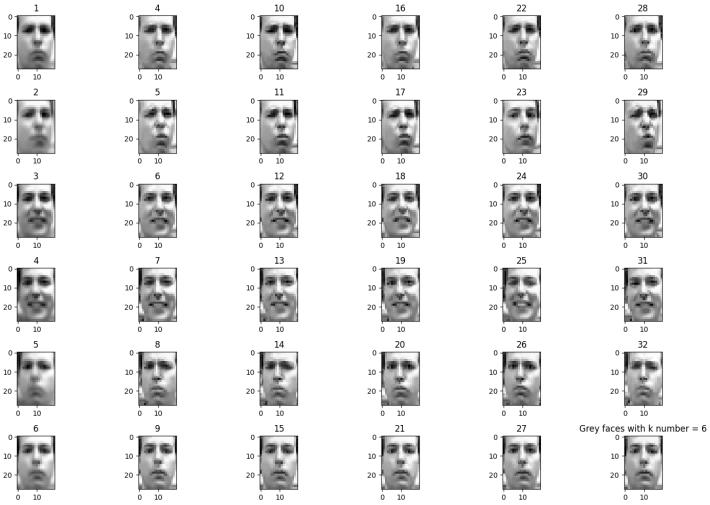

Figure 3: Result of k = 6

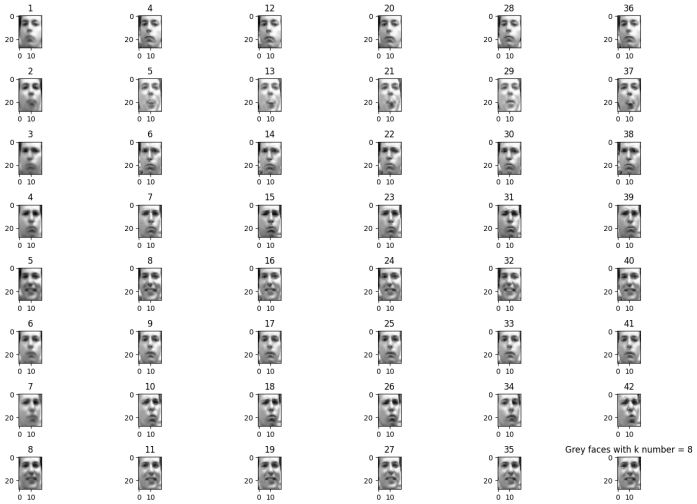

Figure 4: result of k = 8

2e

From these figures we can see that the faces in each cluster is very much alike, as we predicted on beforehand. This confirms our first belived theory that the clustering algorithm would cluster based on characteristic on the face, like smiling, tounge sticking out, tilting and others.

2f

If the faces were not centered in the middle, the algorithm might have started to cluster a bit differently. This is because this type of clustering is very sensitive against changes, and even the slightest could mean a different solution. It might have clustered more on where the head was aligned to the center, than what the head was doing regardless of the center. Like smiling, sticking out the tounge.

References

[1] Geeks for geeks, Clustering in Machine Learning, from https://www.geeksforgeeks.org/clustering-in-machine-learning/ (30.10.24)

[2] Turner, Luke, Towards Data Science, 9. apr 2022, from https://towardsdatascience.com/create-your-own-k-means-clustering-algorithm-in-python-d7d4c9077670 (30.10.24)

A Code for problem 2d

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random as rnd



frey_face = pd.read_csv("home_exam/frey-faces.csv", skiprows=4, delimiter=" ", header=None)

frey_face = frey_face.to_numpy()

width = 20
height = 28

# fig, axes = plt.subplots(5, 5, figsize=(15, 5))

# count = 0
# for y in range(len(axes[0])):
#     for x in range(len(axes[1])):
#         axes[y, x].imshow(frey_face[count].reshape((height, width)), cmap='gray')
#         axes[y, x].set_title(f"{count}")
#         count += 1

# plt.tight_layout()
# plt.show()

#defining my k

k_list = [3, 6, 8]
#Defining a function to calculate distance
def distance(center_image, now):
    s = np.sum((center_image - now)**2, axis=1)
    return np.sqrt(s)

max_iteration = 500

for k in k_list:




    #finding the centroids
    centroids = []
    for _ in range(k):
        center_ini = rnd.randint(0, len(frey_face))
        centroids.append(frey_face[center_ini, :])

    centroids = np.array(centroids)


    #doing the repetitions
```

```
for i in range(max_iteration):

    clusters = [[] for _ in range(k)]
    prevoius_distance = 100000
    for j in range(len(frey_face)):
        dis = distance(centroids, frey_face[j,:])
        cluster = np.argmin(dis) #findin the correct cluster
        clusters[cluster].append(frey_face[j, :]) #adding the data point to the correct cluster


    new_avg_centroid = np.array([np.mean(clus, axis=0) for clus in clusters])

    if np.all(centroids == new_avg_centroid):
        print("Done by convergence")
        break
    centroids = new_avg_centroid


#Finding the images
closest_images = []
count = 0
for cluster in clusters:
    centroid = centroids[count]
    distances = [np.linalg.norm(image - centroid) for image in cluster]
    closest_indices = np.argsort(distances)[:5]
    closest_images_in_cluster = [cluster[i] for i in closest_indices]
    closest_images.append(closest_images_in_cluster)
    count += 1



closest_images = np.array(closest_images)


#plotting the results
fig, axes = plt.subplots(k, 6, figsize=(15, 10))
count = 0

#plotting the three centroid images
for t in range(k):
    axes[t, 0].imshow(centroids[count].reshape((height, width)), cmap='gray')
    axes[t, 0].set_title(f"{count + 1}")
    count += 1

count = 0
for x in range(1, len(axes[1])):
    for y in range(0, k):
        # print("y = ", y)
        # print("x = ", x)
        axes[y, x].imshow(closest_images[y, x-1].reshape((height, width)), cmap='gray')
        axes[y, x].set_title(f"{count + 4}")
        count += 1
#Saving the file
plt.tight_layout()
plt.title(f"Grey faces with k number = {k}")
plt.savefig(f"home_exam/task2/grey_face_k{k}")
```