18.10.24 - 01.11.24

Candidate number: 08

Table of contents

0.1 Introduction to exam

The machine learning exam fall 2024 can be broken down into two parts. **Part one** involves classify whenever molecules like fat or not based on their lipophilicity. The task allows any function/method in the scikit-learn package introduced in the course. **Part two** focuses on implementing a K-nearest neighbors algorithm to classify the dataset frey_faces.csv. Additionally, there is also a few questions to answer.

1.0 Part one

1.1 Exercise 1

Part One of this paper presents a solution to a binary classification problem, where molecules are classified based on their lipophilicity. In the following sections, the paper will address several fundamental problems in machine learning, including data handling, model selection, cross-validation, parameter tuning, and result evaluation.

1.2 Preprocessing

The first step in preprocessing a dataset is to obtain an overview of its structure, which includes checking the matrix size, identifying missing values, and detecting "useless" features. I define useless features as those that do not provide any additional information. For instance, features with the same values across all samples would be considered useless.

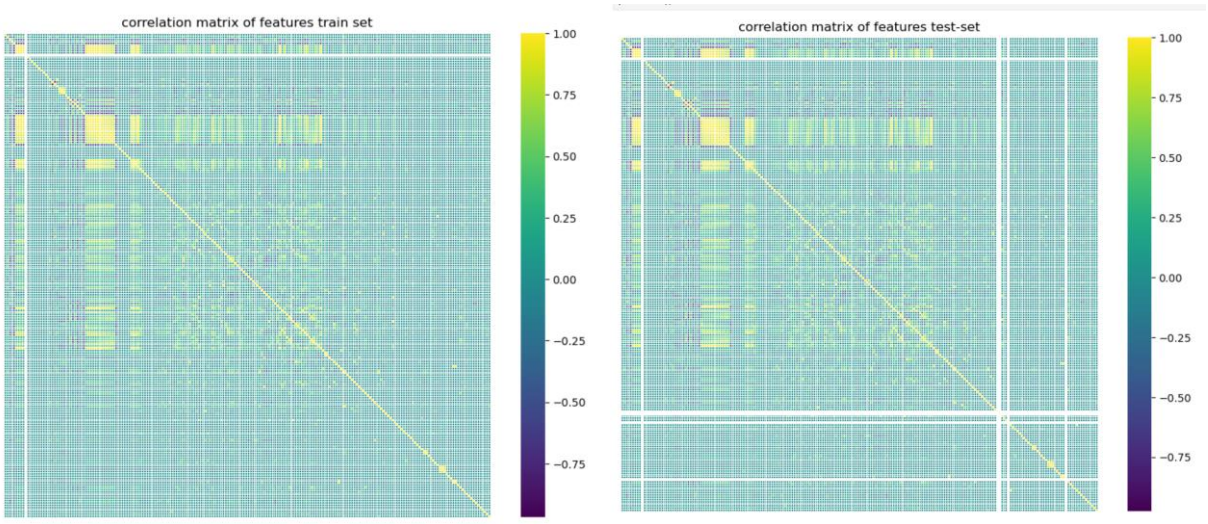
In the training set, there are 228 features and over 4,000 samples across two datasets. While there are no empty cells, many features contain a significant number of zero values. In some cases, over 90% of the samples have zero values in certain features. I strongly considered treating at least some of these zero values as missing data. However, lacking domain knowledge specific to this dataset, I ultimately decided against it.

Removing useless features

Additionally, there are many useless features that do not provide any additional information and only add complexity to the computational structure of the model(s). There was a total of 16 features removed because of this.

Correlation matrix of features

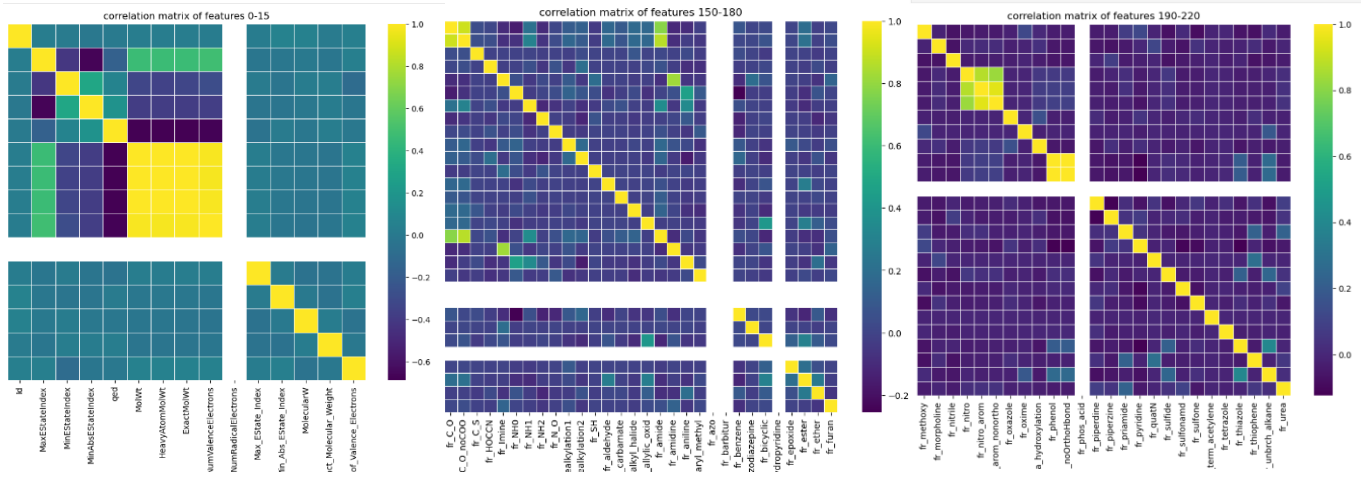
When working on large and complex datasets, its easy to get overwhelmed. Understanding the relationship between features can be difficult. For this reason, there is generated a correlation matrix for each dataset.



The bar on the right side of the correlation matrix indicates how strong the correlation is between features.

Values closer to one indicates strong, and almost a linear correlation, while a value closer to minus one show almost no correlation. Both extremes can impact the dataset negatively if not handled appropriately.

When analysing the matrices, a clear difference between the training and test datasets emerges. The white lines highlight features that show no correlation at all, suggesting that they do not provide useful information. The exercise description states: “and some of the features have been made up artificially by some very mean members of the teaching team”. I believe these features are the white lines in the test dataset. In order to further inspect these features, I zoomed in on the matrix and included feature labels:



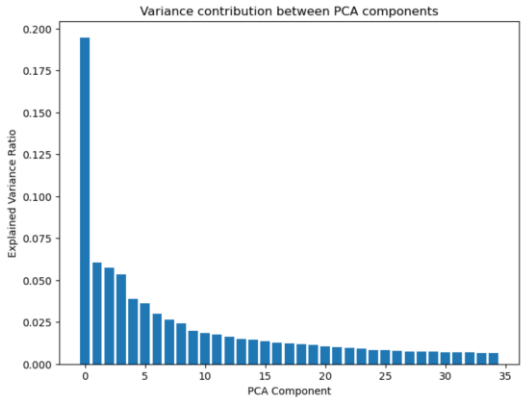
After identifying the features: *['fr_azo', 'fr_barbitur', 'fr_dihydropyridine', 'fr_phos_ester', 'fr_phos_acid','NumRadicalElectrons']*, as artificially made-up features. The next logically step was to remove them. However, this does not resolve our initial problem of having a highly correlated matrix of features.

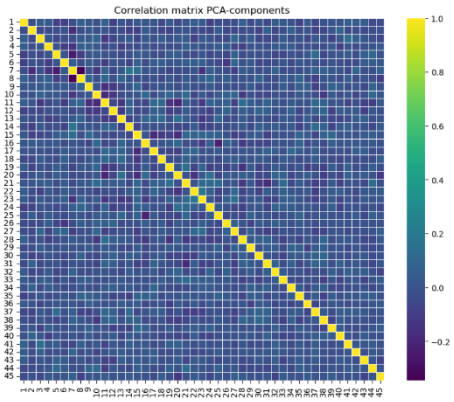
Applying PCA

Applying Principal Component Analysis (PCA) is common practice in machine learning, when working with highly correlated features. It's strength in dimension reduction, handling of noise and creation of new uncorrelated features, makes it a powerful tool. PCA improves model performance while simplifying complex datasets.

The first step is to transform the datasets using standard scale. This transformation makes sure that all the features have a mean of zero and a standard deviation of one. While maintaining the ratio of the original values, this process guarantees all features affecting the calculation the same. This prevents domination due to scaling differences.

Afterward the datasets are ready for Principal Component Analysis. When applying the PCA, there is a variable that selects how much of the variance one wish to keep. This will initially be set to 90%, this may be adjusted in the future if feature selection is not implemented.





We can see that the new features are not correlated with each other. This will hopefully result in the model being able to pick up underlying patterns and predict whether the molecule likes fat or not.

The whole preprocessing process is documented in the Jupiter Notebook file ***preprocessing.ipynb***

1.3 Model selection and reasoning

Choosing the right model is crucial when solving binary classification. Especially when working with complex datasets such as in this exercise. Since every model introduces in this course is available to use, I have chosen four different models, logistic regression, random forest, k-nearest neighbors and naive bayes classifier. I will consider how they fit the datasets. The primary goal is to maximize accuracy / f1 score. So, factors such as computational efficiency will not be considered.

Logistic regression

Logistic regression **(LR)** is a simple model that pick us linear relationships. The model picks up how each feature affect the outcome. Logistic regression often struggles with picking of relationships between features. The LR model used for the exam is a copy of the one used in mandatory assignment 1, with a few adjustments.

(code provided in folder ex1)

Random forest

Random forest **(RF)** is a complex and robust model to solve complex ML problems. The model's ability to handle non-linearity, makes it able to pick up relationships between different features. Since the forest consists of "k" amount of smaller decision trees, it's also decent in preventing overfitting.

K-nearest neighbors

k-nearest neighbors **(K-NN)** are a reliable model that does not require a statistical distribution to work, it's great to pick up underlying patterns and relationships between features on higher dimensions. The main problem with this model is its ability to adapt to noisy data and scaling. Which may result in the model struggling to adapt, **even after applying PCA**.

Naive bayes classifier

The naive bayes classifier is a simple but reliable classifier, it's ability to make its own threshold may result in it working well with this given dataset. The main problem with these models is the inability to pick up relationships between features. Since the model is based on bayes formula, the model assumes all features are independent to each other. This is often not the case when working with huge datasets.

1.4 Cross validation and parameter tuning

To evaluate the performance of the models before submitting, I used cross validation on the training set. Instead of splitting the training data into a separate training and test set. Cross validation provides a more reliable estimate of how well the models respond to new unseen data. Since it's over 3000 samples in the training set, cross validation is a suitable choice.

To increase accuracy, there was a need to tweak the parameters of the models. In the search for the optimal parameters. I implemented a function that tries different combinations and return the one with the best result.

(code provided in appendix A)

model name	Default parameters (accuracy / f1 score)	Tweaked parameters (accuracy / f1 score)
Logistic regression	69.3% / 69.2%	70% / 69.8%
Random forest	75.4% / 70.32 %	77.05% / 72%
K-nearest neighbor	77.3% / 75.6%	78.4% / 77.9%
Naive bayes classifier	68% / 69%	68% / 69.2%

As you can see, the tweaked parameters improved the models, but only around 1-2% so the change is minimal.

1.5 Further model improvements

To further improve the model. There is also a final loop in the submission file. This loops's purpose is to find the optimal number of features and what these features are. Through forward selection, the model identifies the best combination of features for each iteration. Meanwhile, the function keeps track of the best accuracy up to date. This loop was ran on the K-nearest neighbor model, because of its earlier accuracy.

Forward selection

Forward selection is a simple but effective way to select a pre-determined number of features. It starts with no features and adds one feature at the time. For each iteration, it adds the feature that yields the best results by an evaluation metric. Which in this case is a cross validation. I chose this model due to its simplicity compared to other feature selection methods.

Even though the process is simple, this loop took around 90 minutes to run.

```
for i in range(68):
    sfs = SequentialFeatureSelector(knn, n_features_to_select=i+1, direction='forward', cv=cv, n_jobs=-1)
        #forward selection
    sfs.fit(train_X, train_y)
        #train with current amount of feautures


    scores = cross_val_score(knn, sfs.transform(train_X), train_y, cv=cv, n_jobs=-1)
    mean_score = np.mean(scores)
        #avg of all 5 cross validations

    if mean_score > best_score:
        #keeps track of current best score
        best_score = mean_score
        best_amount_of_features = i+1

print(f"best score: {best_score}, with {best_amount_of_features} features")
```

Thankfully it ended up with increasing the score to 80% on the training set. I also discovered that 40 features were the optimal amount on the training set.

1.6 Results and validation

Uploading the predictions for the test set did not provide the results expected. Since the models performed in the low 80% one would expect something similar when submitting the test set. I submitted every combination of models in the table in subsection 1.4, with performances ranging from the low 40% all the way into the high 50%. The highest score I got was right under 60% with the K-nearest neighbor algorithm with the optimized parameters, and removing the forward selection process in 1.5.

The difference between performance on the training and test dataset, may suggest that the model overfitted in the training process, or that a significant difference between the datasets were not captured by the model. It could also be that the preprocessing of the datasets was not good enough.

(Final submission file is included in appendix A)

2.0 Part two

2.1 What is clustering?

Explain what clustering is, and how it is different from classification. Give an example of a problem where clustering might be the appropriate solution.

Clustering is a machine learning method that groups data together without any labels. It is an unsupervised learning method that focuses on underlying patterns and space in vectors to group together samples. Unlike classification it does not need to be trained beforehand, and its only goal is to group together data.

Example of a clustering problem:

Clustering could be used to separate houses and apartments based on its features. By gaining access to information such as size, bedrooms, location and price. Without knowing if the samples are houses and apartments, clusters could automatically separate the two classes. It does this by taking advantage of underlying patterns in the housing market.

2.2 K-mean clustering

Describe the k-means algorithm. Explain how the cluster centroids are initialized and updated. What is the stopping criterion for the algorithm? Typically, k-means is used on vectors, but how can you use it on 2D image data?

K-means algorithm:

The model's primary goal is to group data into a pre-selected number of clusters, often notated as **"k"**. Each sample is represented as a vector, where each feature is displayed as a separate dimension. The algorithm creates centroids with some given conditions. It then moves the centroid toward the average position of all the closest samples. Here is a breakdown of how the algorithm manipulates the centroids into clusters of data:

How centroids are initialized:

There is primarily two ways to create centroids:

1) **Random**

Centroids are placed randomly within the data space. This method is simple but risky, as randomly chosen centroids may end up too close to each other or too far from the actual data clusters. This could lead to poor clustering results, as certain clusters might overlap or remain underrepresented. However, if the random centroids are placed near the true cluster centers, this approach can work well.

2) **Using samples**

Centroids can also be created by using the vector position of samples in the dataset. This ensures that the centroids are close to the data points we wish to cluster. There is still a chance that we pick samples that is close to each other, therefore it's important to look at the samples when using samples for centroids. This could be hard to do in high dimensional vector space.

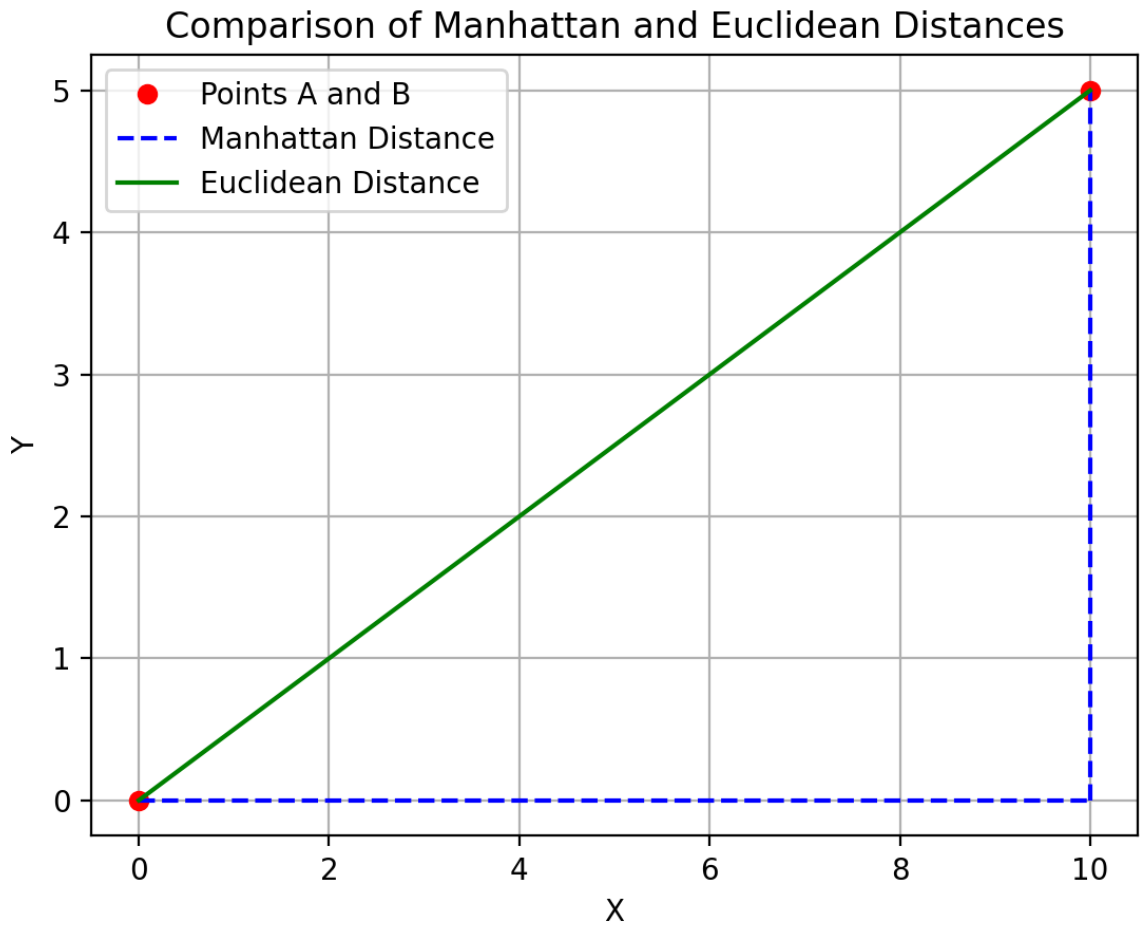
Each method has their use cases depending on the goal of the algorithm and the dataset. Therefore, it is important for the developer to considers the best option.

Updating centroids:

1) Each vector is assigned to the closest centroid
2) Centroid is moved toward the average position of all the closest centroids.

3) These steps are repeated until a stopping criterion is met.

There are multiple ways to determine the distance between vectors. The most common is Euclidean distance and Manhattan distance. Under is a plot I made to visualize the difference:



Stopping criterions:

1) Max iterations: The model has run for a predefined number of iterations.
2) Minimal change: There is no longer significant change in centroid position from last iteration.

Using k-means on 2D images:

By representing each pixel as a feature, we can use K-means algorithm to cluster together images. We can use information such as color or intensity of each feature to make groups of the pictures.

Steps:

1) Flatten the images into a vector of pixels, and give each pixel numerical values determined by their color/intensity
2) Apply k-mean clustering to the vectors
3) Reconstruct the images

2.3 Display of dataset

From the images you see, do you have any intuition on what clusters the k-means algorithm will yield?

Display of 5 random pictures from the dataset

Given that the angle, lighting, and placement of the faces are consistent across the dataset, I predict that the k-means algorithm will primarily group images based on facial expressions. With relatively few pixels representing each image, capturing finer facial details is challenging. Therefore, the algorithm is likely to focus on more prominent differences, such as variations in facial expression.

2.4 Implementation of the K-means algorithm

The k-means algorithm was implemented using the principals in **2.2**, in **summary**:

1) Initialization

The centroids are initially created as random samples within the dataset. I chose this approach because of the dataset. Each sample has 28x20 features (560), so with creating a random centroid, there is a chance the vector would land outside of the data's range. Leading to poor clustering. Using samples is a safer approach when its impossible to visualize the vector space.

2) Distance metric

In this implementation of the k-means algorithm, the user can choose between Manhattan and Euclidean distance. When testing the program, I did not notice any difference between the methods. Given that the computational cost for the Manhattan method is cheaper, I chose to use it.

3) Stopping criterion

The model runs either until a maximum number of iterations is reached or until the centroids no longer change significantly between iterations. Specifically, the algorithm stops when the change in centroid positions is smaller than a set tolerance. Due to computational precision issues with very small changes, I set the tolerance to 10^{-8}.

4) Data handling

The dataset is already preprocessed into a 1D array of numerical values. After the clustering process, the pictures are reconstrued using the pyplot import from matplotlib. Chatgpt helped me with the reconstruction.

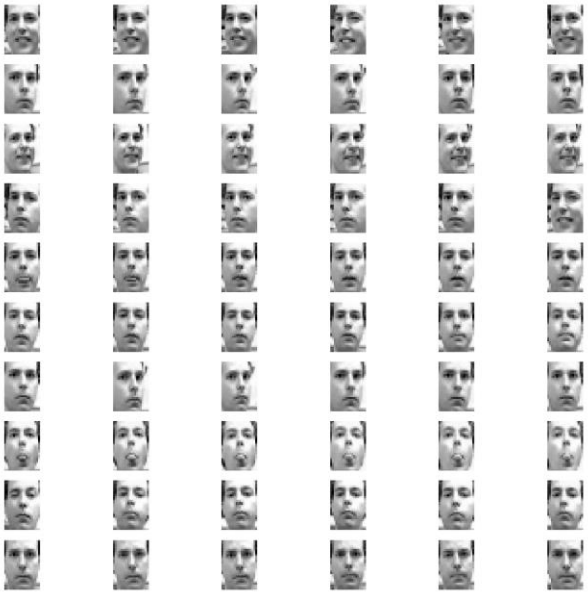
*(The full snippet of the code is included in **Appendix B)***

2.5 The models result

In this implementation, the user has the option to choose the number of clusters (k), Here is the result of different k values:

K= 3

K=6



K=10

Based on the visualization, what do the individual clusters represent? Does what you observe align with your initial intuition?

Based on the visualizations, the model effectively clustered pictures of the same person, despite accounting for image flips, facial expressions and different angles. This outcome surprised me, as I initially believed that facial expressions would be the dominating factor. The result suggests that face shape, positing of eyes and facial features are the dominating factors. This also proves that k-means clustering is not only a simple model, but also an effective one.

What do you think would happen if the faces in the dataset were not perfectly centred in the image but randomly (slightly) shifted to the left or the right? Would you get the same results? why?

If the pictures were not perfectly centered, I think it would negatively affect the model. Simple algorithms like K-means are sensitive to positioning because each pixel is treated as a feature. When the pixels shift, they would end up being compared with different features. This means prominent features like face shape, eye positioning, and facial details wouldn't be matched correctly anymore. As a result, the algorithm's ability to detect underlying patterns would significantly weaken, and we would not get the same results.

3.0 Conclusion

In this exam I mainly handled two main tasks:

Implementing a binary classification to predict whenever a molecule likes fat or not and clustering a dataset of images using a self-implemented K-means algorithm. For the classification task, I followed several steps including, preprocessing, model selection and optimizing parameters. Even though the accuracy on the test set was disappointing. This exercise has taught me the importance of testing the model on unseen data, to ensure and test the quality of the predictions the model yields.

In the cluster task I managed to classify faces using the K-means model. I was pleasantly surprised how effective the simple algorithm was. The model was able to cluster the same individuals together, despite the change in facial expression and flipped images.

4.1 appendix A

Python script used for optimizing parameters:

```python
def testing_parameters_lr(lr_list,max_iter_list,treshold_list):
    """
    tests all combination of given parameters to optimize the accuracy of
    the logistic regression model.
    """

    results = []
        #array to save results

    for lr in lr_list:
        for max_iter in max_iter_list:
                for treshold in treshold_list:

                    lr_model = LogisticRegression(lr, max_iter,treshold)
                    cv_scores = cross_val_score(lr_model, X, y, cv=5, scoring='accuracy')
                    results.append((lr, max_iter,treshold, cv_scores.mean()))

    results = np.array(results, dtype=object)
        #convert into np array to use np.argmax

    scores = results[:, 3].astype(float)
        #get the accuracy results
    best_index = np.argmax(scores)
    best_params = results[best_index]

    return best_params


def testing_parameters_rf(criterion_list,max_depth_list,
                          max_features_list,min_sample_split_list):
    """
    tests all combination of given parameters to optimize the accuracy of
    random forest model.
    """
    results = []

    for criterion in criterion_list:
        for max_depth in max_depth_list:
            for max_features in max_features_list:
                for min_sample in min_sample_split_list:
                    rf_model = RandomForestClassifier(criterion=criterion,
                                                      max_depth=max_depth,
                                                      max_features=max_features,
                                                      min_samples_split=min_sample)


                    cv_scores = cross_val_score(rf_model, X, y, cv=5, scoring='accuracy')
                    results.append((criterion, max_depth, max_features,min_sample, cv_scores.mean()))

    results = np.array(results, dtype=object)
        #convert into np array to use np.argmax

    scores = results[:, 4].astype(float)
        #get the accuracy results
    best_index = np.argmax(scores)
    best_params = results[best_index]

    return best_params


def testing_parameters_knn(n_neighbors_list,weight_list,metric_list):
    """
    tests all combination of given parameters to optimize the accuracy of
    knn model.
    """

    results = []

    for n_neighbors in n_neighbors_list:
        for weight in weight_list:
            for metric in metric_list:
                knn_model = KNeighborsClassifier(n_neighbors=n_neighbors,
                                                 weights=weight,
                                                 metric=metric)

                cv_scores = cross_val_score(knn_model, X, y, cv=5, scoring='accuracy')
                results.append((n_neighbors, weight, metric, cv_scores.mean()))

    results = np.array(results, dtype=object)
        #convert into np array to use np.argmax

    scores = results[:, 3].astype(float)
        #get the accuracy results
    best_index = np.argmax(scores)
    best_params = results[best_index]
```

Combination of parameters tested:

```
lr = [0.01,0.05,0.1]
max_iter = [1000,5000]
treshold = [0.3,0.4,0.5,0.6,0.7]

print(testing_parameters_lr(lr,max_iter,treshold))
    #[0.1 1000 0.3 np.float64(0.7002976190476191)]

criterion = ['gini', 'entropy', 'log_loss']
max_depth = [None,10,20,3]
max_features = ['sqrt', 'log2']
min_sample_split = [5,10,100]
print(testing_parameters_rf(criterion,max_depth,max_features,min_sample_split))
    #['log_loss' None 'sqrt' 5 np.float64(0.7705357142857143)]


n_neighbors = [3,5,10]
weights = ['uniform','distance']
metric = ['minkowski','euclidean','manhattan']
print(testing_parameters_knn(n_neighbors,weights,metric))
    #[5 'distance' 'minkowski' np.float64(0.7848214285714284)]

y = np.array(y)
class_counts = np.bincount(y)
prior = class_counts / len(y)


gnb_model = GaussianNB(priors=prior)


cv_scores = cross_val_score(gnb_model, X, y, cv=5, scoring='accuracy')


print(f"Gaussian Naive Bayes Cross-Validated Accuracy: {cv_scores.mean()}")
    #accuracy 0.68
```

Python script for submission

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.model_selection import StratifiedKFold, cross_val_score
import pandas as pd
import numpy as np

train_X =  pd.read_csv('X_val_train.csv')
train_y = pd.read_csv('label_train.csv')
train_y = train_y.values.ravel()
    #transform into 1d array
test_x = pd.read_csv('test_preprocessed.csv')
test_id = pd.read_csv('test.csv')
test_id = test_id['Id']

"""
finding optimal amount of features on trainingset:
"""

knn = KNeighborsClassifier(n_neighbors=10,weights='distance',metric='manhattan')
cv = StratifiedKFold(n_splits=5)
best_score = 0
best_amount_of_features = None

for i in range(68):
    sfs = SequentialFeatureSelector(knn, n_features_to_select=i+1, direction='forward', cv=cv, n_jobs=-1)
        #forward selection
    sfs.fit(train_X, train_y)
        #train with current amount of feautures


    scores = cross_val_score(knn, sfs.transform(train_X), train_y, cv=cv, n_jobs=-1)
    mean_score = np.mean(scores)
        #avg of all 5 cross validations

    if mean_score > best_score:
        #keeps track of current best score
        best_score = mean_score
        best_amount_of_features = i+1

print(f"best score: {best_score}, with {best_amount_of_features} features")
    #0.8014880952380953 40

"""
model prediction
"""

knn.fit(train_X,train_y)
predictions = knn.predict(test_x)
submission = pd.DataFrame({'Id': test_id, 'lipophilicity': predictions})
submission.to_csv('submission.csv', index=False)
```

4.2 Appendix B

K-means algorithm implementation

```python
class Kmean:

    def __init__(self,data:np.array,measure,k:int,tolerance=10e-3):

        self.data = data
        self.k = k
        self.tolerance = tolerance
        self.measure = measure
        self.cluster_assignments = {}

        """
        creation of centroids using random samples, uses samples because of data
        complexity.
        """
        self.num_samples = data.shape[0]
        random_row = np.random.choice(self.num_samples, size=k, replace=False)
        self.centroids = data[random_row, :]

    def euclidean(self,cord_1,cord_2):

        return np.sqrt(np.sum((np.array(cord_1) - np.array(cord_2))**2))

    def manhatten(self,cord_1,cord_2):

        return np.sum(np.abs(np.array(cord_1) - np.array(cord_2)))


    def fit(self,max_iter):
        """
        fit function follows steps given in 2.2
        """

        for _ in range(max_iter):
            argmin_list = self._distance()
            print(_)

            self._update_centroids(argmin_list)
                #update all datapoint into closest centroid

            if self._check():
                #check if the centorids have moved less than a given tolerance
                self._store_cluster_assignments(argmin_list)
                break


        return self._store_cluster_assignments(argmin_list)


    def _distance(self)-> np.array:
        """
        calulate distance between all the centroids and each datapoint
        return the index of closest centroid for each sample in a np list
        """

        num_samples = self.data.shape[0]
        num_centroids = self.centroids.shape[0]
```

```
    distances = np.zeros((num_samples, num_centroids))
        #makes an empty matrix (samples x centroids)


    for i in range(num_samples):
        for j in range(num_centroids):
            distances[i, j] = self.measure(self.data[i], self.centroids[j])
                #calulate distance, between centroid and sample

    return np.argmin(distances,axis=1)


def _update_centroids(self,argmin_list: np.array):
    """
    update all datapoint into closest centroid,also saves the prevoius centroids.
    """
    self.previous_centroids = self.centroids.copy()


    for i in range(self.k):
        points_in_cluster = self.data[argmin_list == i]
        self.centroids[i] = np.mean(points_in_cluster, axis=0)


def _check(self):
    """
    This function checks if there is no signicant changes between updating
    of the centroids.
    """
    if self.measure(self.previous_centroids,self.centroids) < self.tolerance:
        return True

    return False

def _store_cluster_assignments(self, argmin_list):
    """
    save all the datapoints, only used when the function ends
    """


    for i in range(self.k):
        self.cluster_assignments[i] = []
            #empty list for each centroid

    for i in range(self.num_samples):
        centroid = argmin_list[i]
            # Get the index of the closest centroid for data point
        self.cluster_assignments[centroid].append(self.data[i])
            #store the data into the centroid

    return self.cluster_assignments
```

Reconstruction of the images:

```
figure, axises = plt.subplots(k, 6, figsize=(8, 8))

"""
chatgpt made the display loop:
https://chatgpt.com/share/671583ab-a288-800d-9671-c556a342f36f
"""
for i in range(k):
    for j in range(6):
            # Reshape the image from 1D (560,) to 2D (28, 20)
        image = np.reshape(kmeans.cluster_assignments[i][j], (28, 20))
        axises[i, j].imshow(image, cmap='gray')  # Plot the image
        axises[i, j].axis('off')
            # removes axis values for each picture.

plt.show()
```

5.0 Sources

ChatGPT. (2024). *ChatGPT* [Large Language Model]. OpenAI. https://chatgpt.com/share/671583ab-a288-800d-9671-c556a342f36f

ChatGPT. (2024). *ChatGPT* [Large Language Model]. OpenAI. https://chatgpt.com/share/671a843c-b380-800d-8d4a-e1a040492986

NumPy. (n.d.). *NumPy*. https://numpy.org/

Stack Overflow. (2020, December 15). *What should we do with highly correlated features?* https://stackoverflow.com/questions/65302136/