ÇUKUROVA UNIVERSITY

ENGINEERING AND ARCHITECTURE FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

GRADUATION THESIS

BREAST CANCER CLASSIFICATION PROJECT

By

2016555029 - Ali Doğan

Advisor

Associate Professor – Zekeriya Tüfekci

June 2021

ADANA

Summary

This project has been studied with machine learning and analysis methods with a balanced rate of data on the subject in order to facilitate the detection of breast cancer. First of all, after determining an important issue in our daily life, I found a suitable data set and made analyzes.

In the project, there are detection estimations with minimum error rate with 4 different analyzes. Recently, studies have been made on this issue in order to facilitate early diagnosis in increasing cancer cases. The project is written in Phyton language and in the Spyder development environment. It is a fully working project and 99% success has been achieved as a result of the analysis. The project was successfully completed with 400 lines of code.

The analyzes used in the project are non-compartmental analysis, principal component analysis, the k-nearest neighbors and the best k-nearest neighbor analysis. Different systematics of these analyzes have achieved different results. Preliminary preparation and many processes have been processed in detail in the process of processing the data. All details are covered step by step in the contents section.

In the project where machine learning was used in detail, training and prediction operations were carried out many times. In this process, many error messages and help from online sources were received, all of them were collected in bibliographies. I would like to thank all the work that contributed to the project.

CONTENT

USING ABBREVIATIONS AND ICONS

M: Malignant

B = Benign

AI: Artificial Intelligence

ML: Machine Learning

SE: Standard Error

DB: Database

KNN: K-Nearest Neighbors

LOF: Local Outlier Factor

EDA: Exploratory Data Analysis

PCA: Principal Component Analysis

NCA: Neighborhood Components Analysis

MIT: Massachusetts Institute of Technology

IDE: Integrated Development Environment

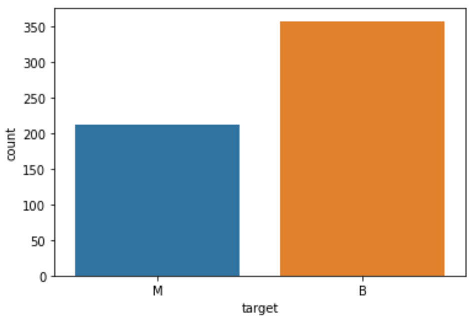
BCCP: Breast Cancer Classification Project

BPKNN: Best Parameters of K-Nearest Neighbors

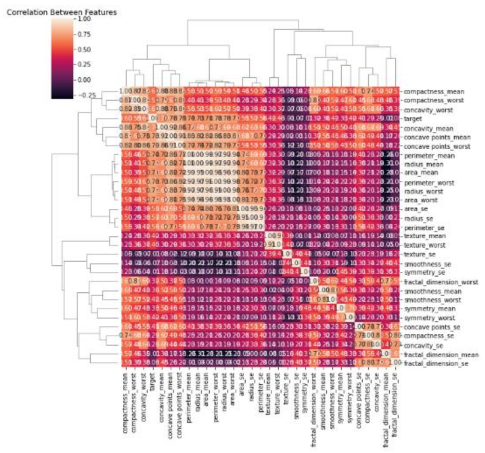
pandas NumPy seaborn

matplotlib scikit learn

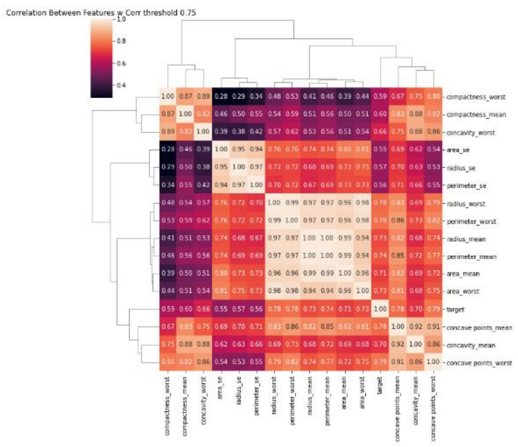
CHART LIST



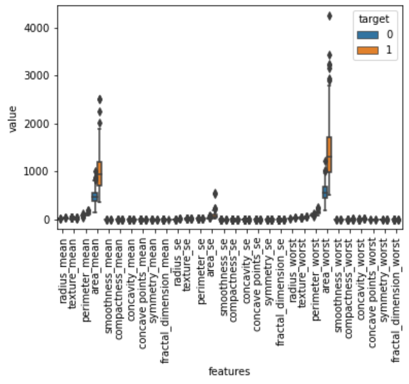
Char 1.1



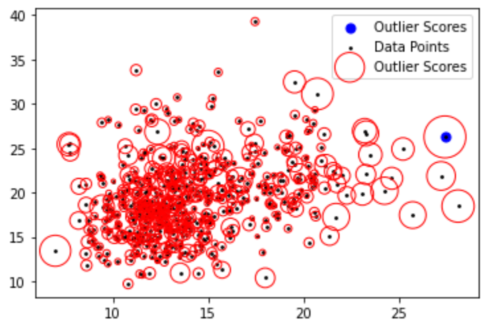
Char 1.2



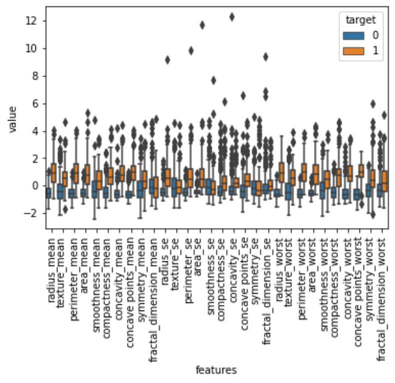
Char 1.3



Char 1.4

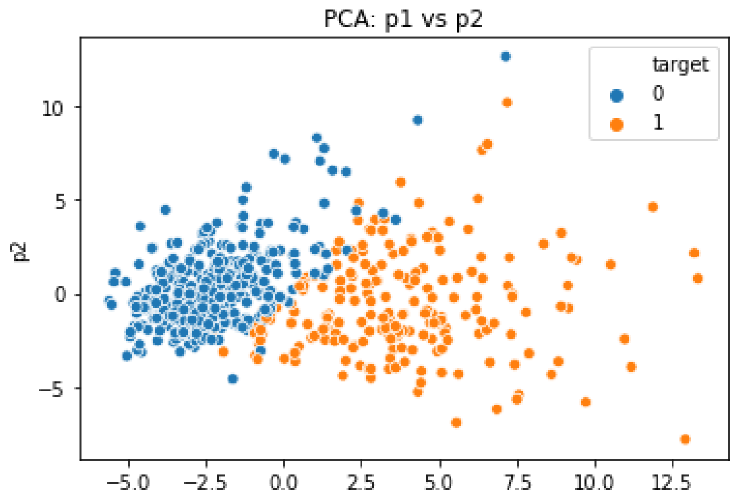


Char 1.5

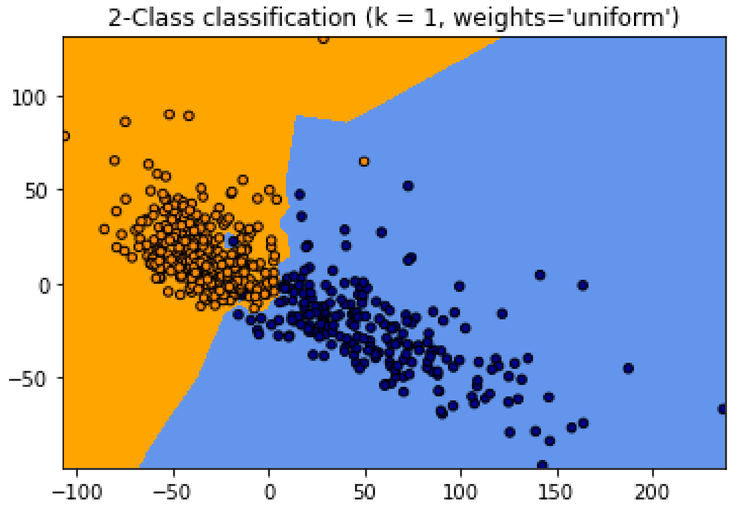


Char 1.6

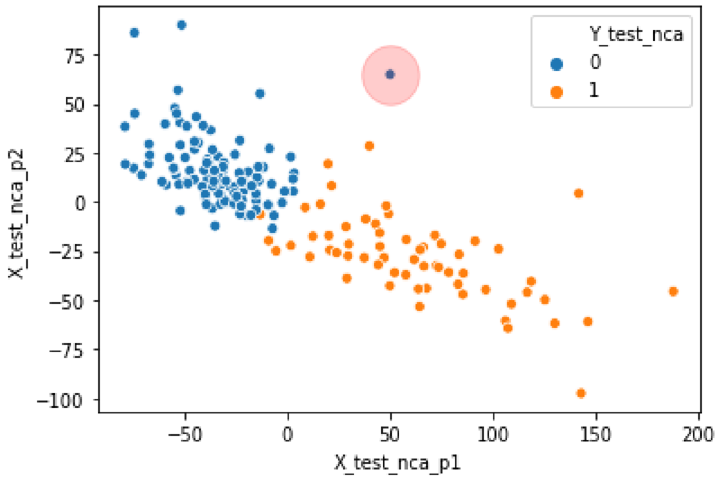
III



Char 1.6

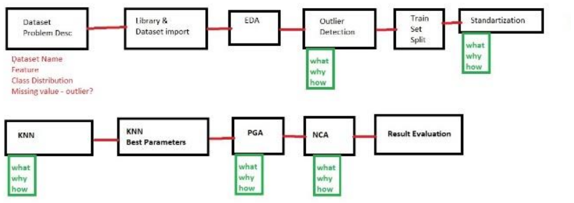


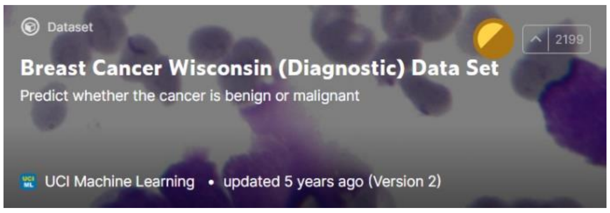
Char 1.7



Char 1.8

IMAGE LISTS





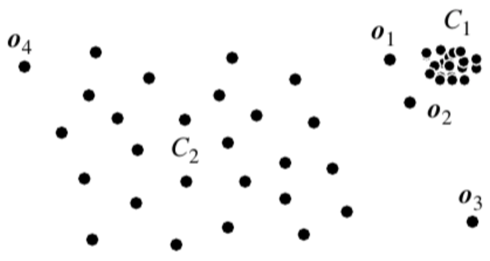
Breast Cancer Wisconsin (Diagnostic) Data Set

Predict whether the cancer is benign or malignant

UCI Machine Learning • updated 5 years ago (Version 2)

Image 1.1

Image 1.2



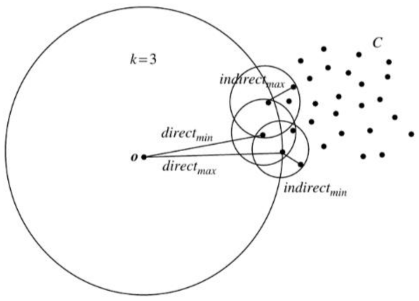
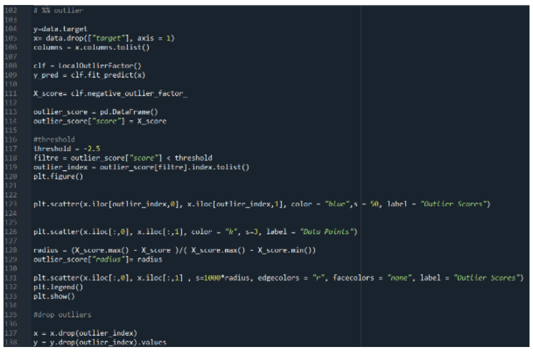


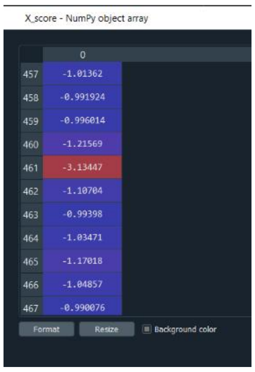
Image 1.3

Image 1.4



```python
# %% outlier

y=data.target
x= data.drop(["target"], axis = 1)
columns = x.columns.tolist()

clf = LocalOutlierFactor()
y_pred = clf.fit_predict(x)

X_score= clf.negative_outlier_factor_

outlier_score = pd.DataFrame()
outlier_score["score"] = X_score

#threshold
threshold = -2.5
filtre = outlier_score["score"] < threshold
outlier_index = outlier_score[filtre].index.tolist()
plt.figure()


plt.scatter(x.iloc[outlier_index,0], x.iloc[outlier_index,1], color = "blue",s = 50, label = "Outlier Scores")


plt.scatter(x.iloc[:,0], x.iloc[:,1], color = "k", s=3, label = "Data Points")

radius = (X_score.max() - X_score )/( X_score.max() - X_score.min())
outlier_score["radius"]= radius

plt.scatter(x.iloc[:,0], x.iloc[:,1] , s=1000*radius, edgecolors = "r", facecolors = "none", label = "Outlier Scores")
plt.legend()
plt.show()

#drop outliers

x = x.drop(outlier_index)
y = y.drop(outlier_index).values
```

Image 1.5

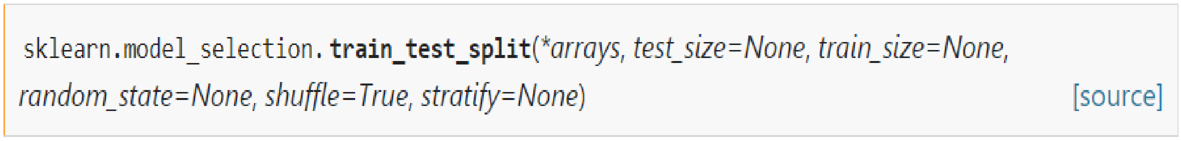


X_score - NumPy object array

	0
457	-1.01362
458	-0.991924
459	-0.996014
460	-1.21569
461	-3.13447
462	-1.10704
463	-0.99398
464	-1.03471
465	-1.17018
466	-1.04857
467	-0.990076

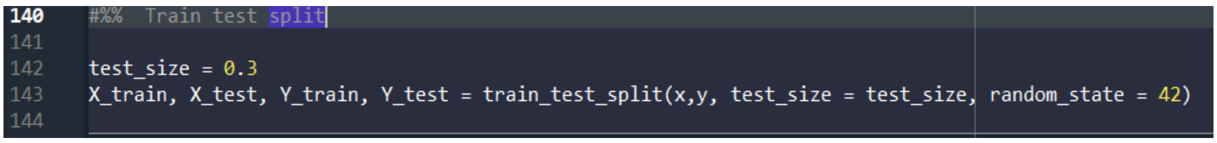
Image 1.6

V



sklearn.model_selection.**train_test_split**(**arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None*) [source]

Image 1.7



```python
#%%  Train test split

test_size = 0.3
X_train, X_test, Y_train, Y_test = train_test_split(x,y, test_size = test_size, random_state = 42)
```

Image 1.8

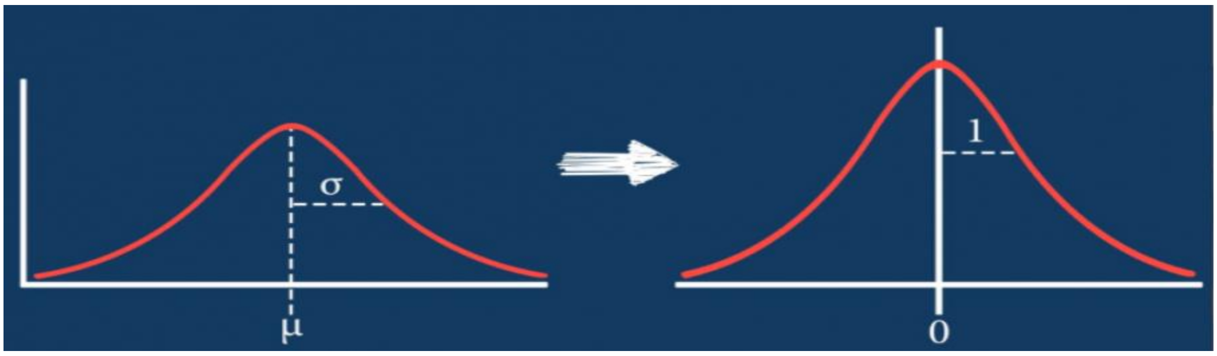
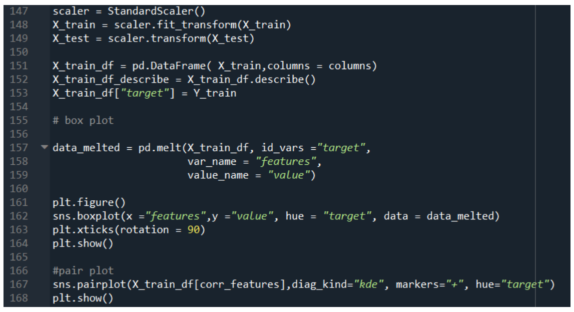


Image 1.9



```python
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train_df = pd.DataFrame( X_train,columns = columns)
X_train_df_describe = X_train_df.describe()
X_train_df["target"] = Y_train

# box plot

data_melted = pd.melt(X_train_df, id_vars ="target",
                      var_name = "features",
                      value_name = "value")

plt.figure()
sns.boxplot(x ="features",y ="value", hue = "target", data = data_melted)
plt.xticks(rotation = 90)
plt.show()

#pair plot
sns.pairplot(X_train_df[corr_features],diag_kind="kde", markers="+", hue="target")
plt.show()
```

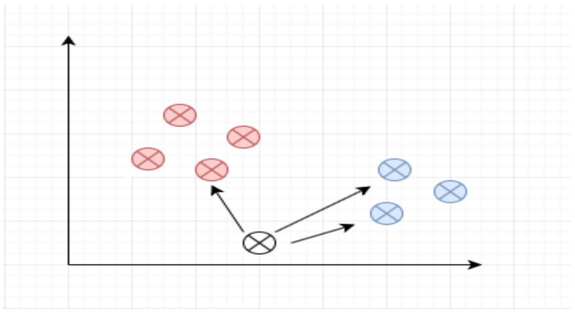
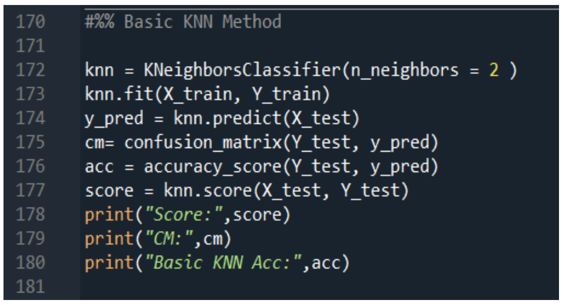


Image 2.0

Image 2.1



```python
#%% Basic KNN Method

knn = KNeighborsClassifier(n_neighbors = 2 )
knn.fit(X_train, Y_train)
y_pred = knn.predict(X_test)
cm= confusion_matrix(Y_test, y_pred)
acc = accuracy_score(Y_test, y_pred)
score = knn.score(X_test, Y_test)
print("Score:",score)
print("CM:",cm)
print("Basic KNN Acc:",acc)
```

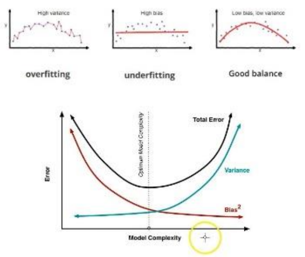
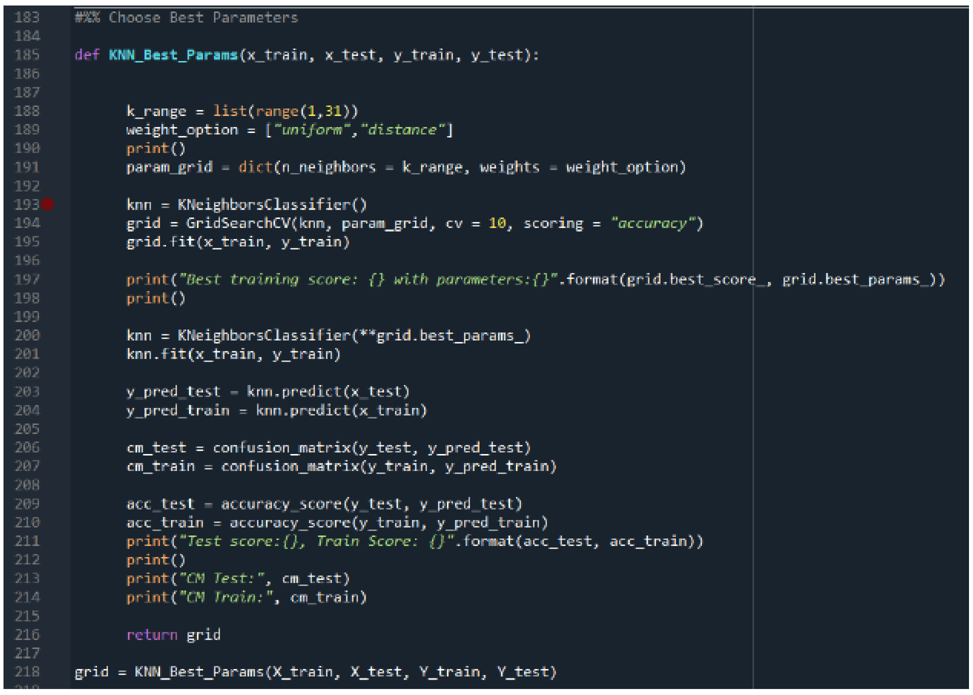


Image 2.2

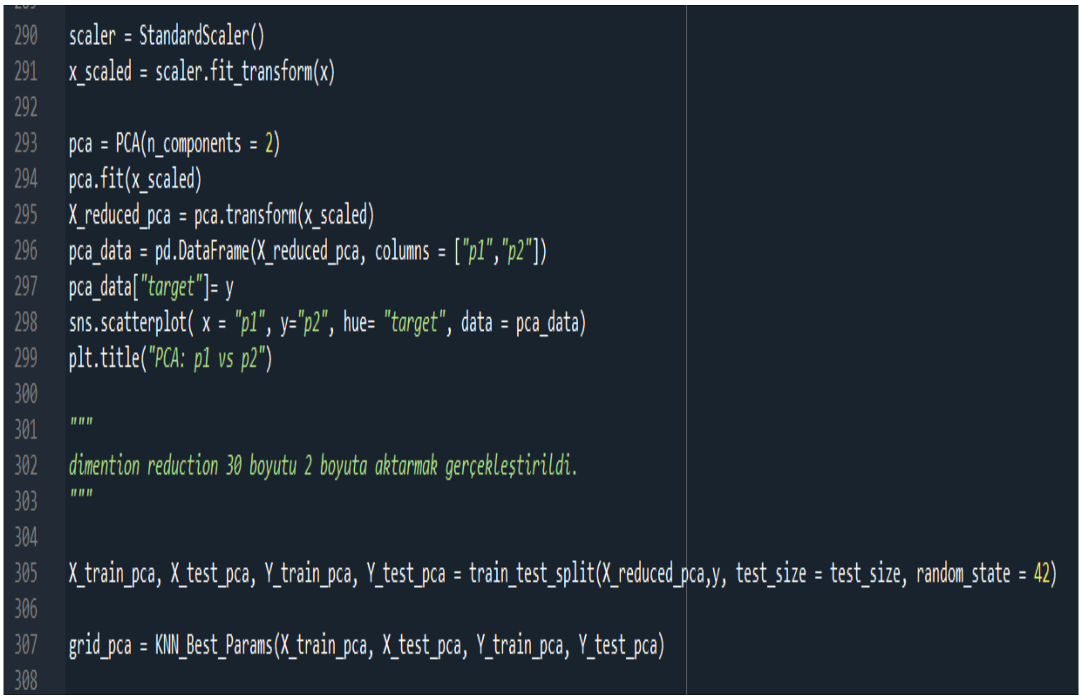
Image 2.3

VI



```python
#%% Choose Best Parameters

def KNN_Best_Params(x_train, x_test, y_train, y_test):


    k_range = list(range(1,31))
    weight_option = ["uniform","distance"]
    print()
    param_grid = dict(n_neighbors = k_range, weights = weight_option)

    knn = KNeighborsClassifier()
    grid = GridSearchCV(knn, param_grid, cv = 10, scoring = "accuracy")
    grid.fit(x_train, y_train)

    print("Best training score: {} with parameters:{}".format(grid.best_score_, grid.best_params_))
    print()

    knn = KNeighborsClassifier(**grid.best_params_)
    knn.fit(x_train, y_train)

    y_pred_test = knn.predict(x_test)
    y_pred_train = knn.predict(x_train)

    cm_test = confusion_matrix(y_test, y_pred_test)
    cm_train = confusion_matrix(y_train, y_pred_train)

    acc_test = accuracy_score(y_test, y_pred_test)
    acc_train = accuracy_score(y_train, y_pred_train)
    print("Test score:{}, Train Score: {}".format(acc_test, acc_train))
    print()
    print("CM Test:", cm_test)
    print("CM Train:", cm_train)

    return grid

grid = KNN_Best_Params(X_train, X_test, Y_train, Y_test)
```

Image 2.4



```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

pca = PCA(n_components = 2)
pca.fit(x_scaled)
X_reduced_pca = pca.transform(x_scaled)
pca_data = pd.DataFrame(X_reduced_pca, columns = ["p1","p2"])
pca_data["target"]= y
sns.scatterplot( x = "p1", y="p2", hue= "target", data = pca_data)
plt.title("PCA: p1 vs p2")

"""
dimention reduction 30 boyutu 2 boyuta aktarmak gerçekleştirildi.
"""


X_train_pca, X_test_pca, Y_train_pca, Y_test_pca = train_test_split(X_reduced_pca,y, test_size = test_size, random_state = 42)

grid_pca = KNN_Best_Params(X_train_pca, X_test_pca, Y_train_pca, Y_test_pca)

```

Image 2.5

1- INTRODUCTION

This area has been created for general information to talk about the technologies used in this project. Python has been chosen for the software language. Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Same time; machine learning methods were also actively used in this project. Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Spyder was used as integrated development environment in the project. Spyder is an open-source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Spyder integrates with a number of prominent packages in the scientific Python stack, including NumPy, SciPy, Matplotlib, pandas, IPython, SymPy and Cython, as well as other open-source software. It is released under the MIT license.

Initially created and developed by Pierre Raybaut in 2009, since 2012 Spyder has been maintained and continuously improved by a team of scientific Python developers and the community.

1- BREAST CANCER CLASSIFICATION PROJECT

Breast cancer is cancer that develops from breast tissue. Signs of breast cancer may include a lump in the breast, a change in breast shape, dimpling of the skin, fluid coming from the nipple, a newly inverted nipple, or a red or scaly patch of skin. In those with distant spread of the disease, there may be bone pain, swollen lymph nodes, shortness of breath, or yellow skin.

Risk factors for developing breast cancer include being female, obesity, a lack of physical exercise, alcoholism, hormone replacement therapy during menopause, ionizing radiation, an early age at first menstruation, having children late in life or not at all, older age, having a prior history of breast cancer, and a family history of breast cancer.

The aim of this project is to do the detection phase with machine learning and analysis methods in a dataset containing cancer and non-cancer patient data without human intervention and to see what results will be in the next case in the light of the data we have.

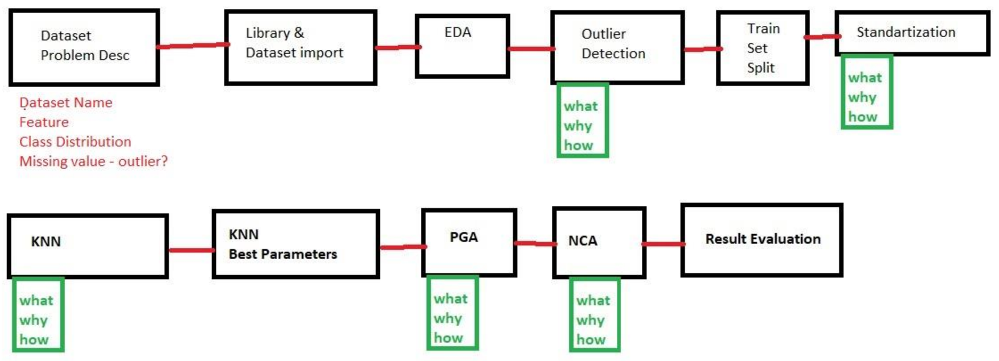


Image1.1

In the project, the first step is completed by understanding the data, defining the problem, importing the necessary libraries, preparing it, and finally standardization, which are the steps shown in the Image1.1. After that, after the standardization, the analysis starts and continues as the results of the analysis. In this progression we will examine what and how each analysis is, and the success rates at different values.

2.1.1- Dataset and Problem Description

Dataset

Breast Cancer Wisconsin (Diagnostic) Data Set

Predict whether the cancer is benign or malignant

UCI Machine Learning • updated 5 years ago (Version 2)

Image1.2

The dataset name I use is Breast Cancer Wisconsin (Diagnostic) Data Set. There is a cancer cell in this dataset, we will guess whether it is benign or malignant. Features are computed from a digitized image of a fine needle aspirate of a breast mass. They describe characteristics of the cell nuclei present in the image.

A balance of data as class distribution. A dataset with no missing value. A dataset containing 32 attributes. Ten properties with true value were calculated for each cell nucleus. These are: radius (mean of distances from center to points on the perimeter), texture (standard deviation of gray-scale values) perimeter, area, smoothness (local variation in radius lengths), compactness (perimeter ^ 2 / area - 1.0), concavity (severity of concave portions of the contour), concave points (number of concave portions of the contour), symmetry, fractal dimension ("coastline approximation" - 1) The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

In this study, feature extraction was made for better results, so after 10 main features, there were 3 new productions for each feature. All feature values are recoded with four significant digits. I have 312 benign data and 212 malignant data. If we rate this, we see that there is not an unbalanced data, such as 63 to 37 percent. Gaussian distributions are more. In addition, there is positive security in some data groups. If we come to our problem, this problem is to reduce the existing workload by transferring this workload to a machine when doctors can classify cancer cells as benign or malignant with 10 different data. If we do this with minimum errors, it can save time and workload and make our doctors more efficient. In the increasing world demands, especially during the epidemic period, such a procedure will significantly reduce their workload. In addition, such a systematic solution is an additional opportunity to minimize deaths due to misdiagnosis and reduce human error.

2.1.2 Libraries and Dataset Upload

First of all, I needed four main libraries to perform EDA, that is to analyze data, to visualize data, these are Pandas, NumPy, Seaborn and Matplotlib. Seaborn and Matplotlib are used for visualization. NumPy takes care of vectorization. Pandas is the library that enables me to analyze data frames.

Pandas: It is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. Its name is a play on the phrase "Python data analysis" itself. Wes McKinney started building what would become pandas at AQR Capital while he was a researcher there from 2007 to 2010. It is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

NumPy: It is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

It targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays; using these requires rewriting some code, mostly inner loops, using NumPy.

Seaborn: There is just something extraordinary about a well-designed visualization. The colors stand out, the layers blend nicely together, the contours flow throughout, and the overall package not only has a nice aesthetic quality, but it provides meaningful insights to us as well.

This is quite important in data science where we often work with a lot of messy data. Having the ability to visualize it is critical for a data scientist. Our stakeholders or clients will more often than not rely on visual cues rather than the intricacies of a machine learning model.

There are plenty of excellent Python visualization libraries available, including the built-in matplotlib. But seaborn stands out for me. It combines aesthetic appeal seamlessly with technical insights.

Matplotlib: “matplotlib.pyplot” is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure, creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels. In “matplotlib.pyplot” various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes.

The library I use for machine learning is Sklearn library. I also added my standardization and model selection methods. I will also apply the process of splitting the data into two as training and test data sets in Sklearn. I used metrics to evaluate the results. After that, I used confusion matrix and accuracy. In this way, we will be able to see where we made a mistake. We will use the neighbors. This is our basic method of analysis. We also used sklearn methods for composition operation.

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and database scan, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. Sklearn is used to build machine learning models. It should not be used for reading the data, manipulating and summarizing it.

Scikit-learn is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.

Scikit-learn integrates well with many other Python libraries, such as Matplotlib and plotly for plotting, NumPy for array vectorization, Pandas dataframes, SciPy, and many more.

Breast cancer is a disease in which cells in the breast grow out of control. There are different kinds of breast cancer. The kind of breast cancer depends on which cells in the breast turn into cancer.

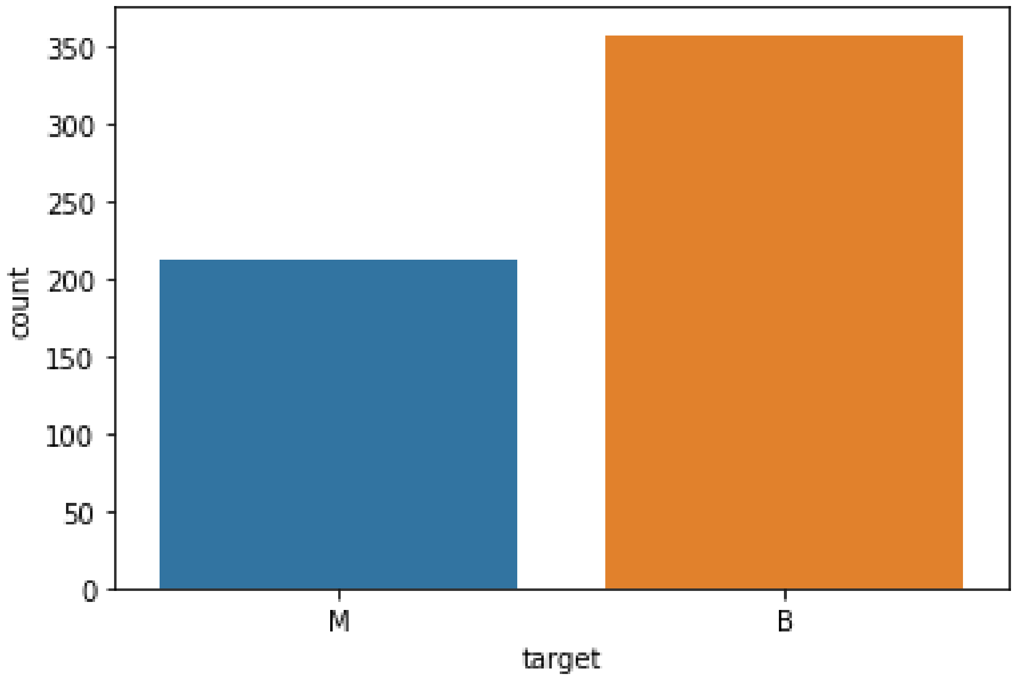
Breast cancer can begin in different parts of the breast. A breast is made up of three main parts: lobules, ducts, and connective tissue. The lobules are the glands that produce milk. The ducts are tubes that carry milk to the nipple. The connective tissue (which consists of fibrous and fatty tissue) surrounds and holds everything together. Most breast cancers begin in the ducts or lobules.

Both graphical and vectorial and large libraries were needed to measure and display such cancer data. For this reason, these libraries were carefully found and selected.

Finally, I called it with the database import process, this is a csv file, that is a data set separated by commas. I changed some columns names and applied drop operation to unnecessary columns.

2.2 Exploratory Data Analysis

Let's visualize how much data we have with graphs for easy understanding and the balance between benign or malignant.

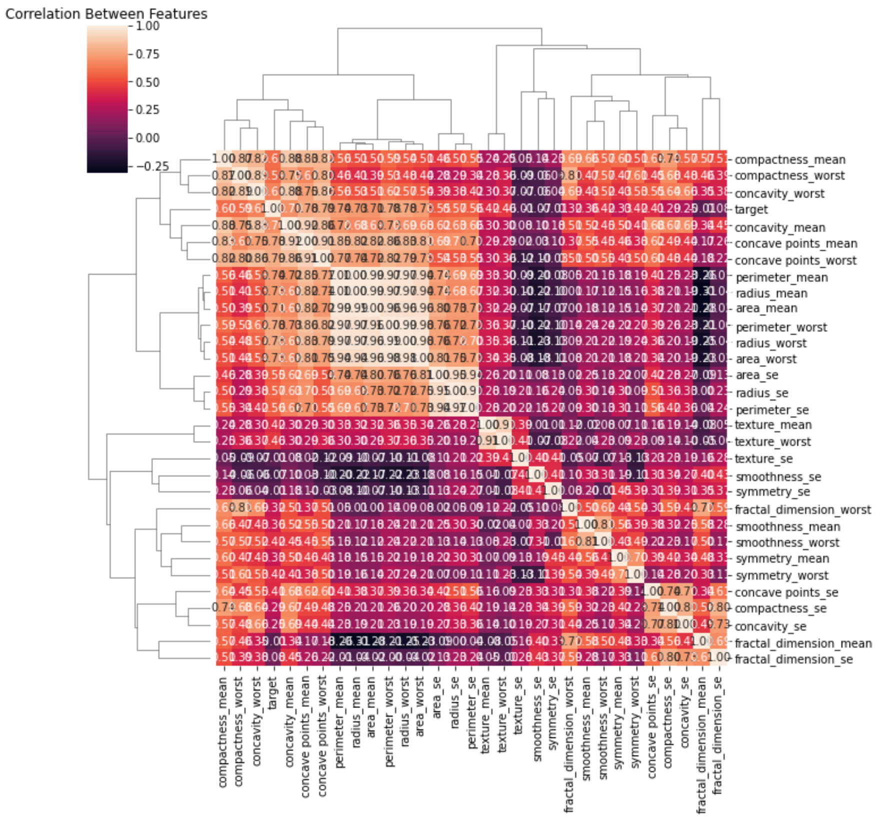


Char 1.1

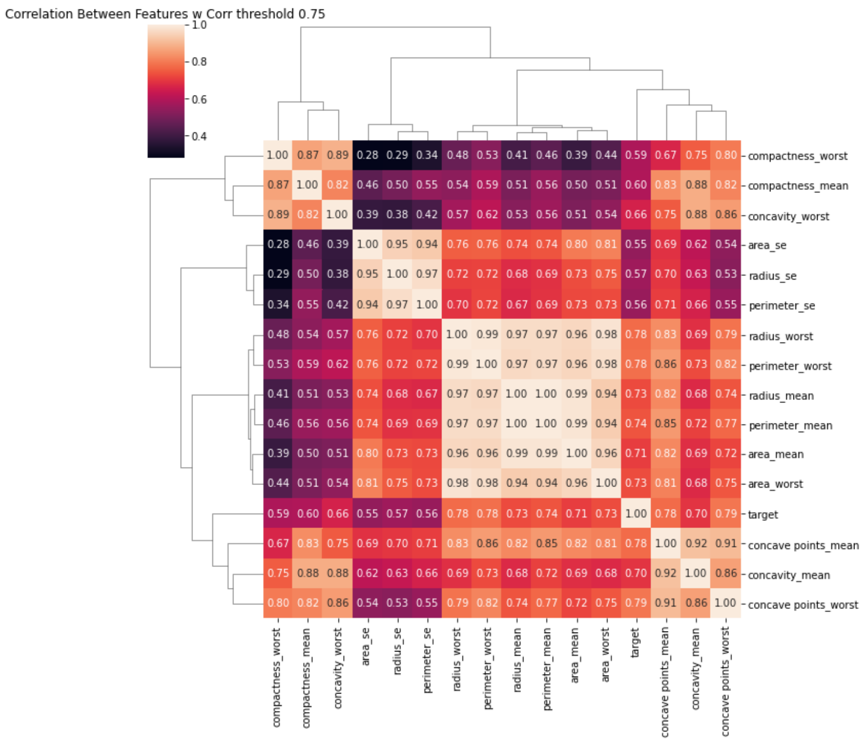
We can see in Char 1.1, we have only numerical data, so we must first examine its correlation matrix. A correlation matrix is simply a table which displays the correlation coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values in a table. It is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data. A correlation matrix consists of rows and columns that show the variables. Each cell in a table contains the correlation coefficient.

I have a dot plot describing the relationship between 30 features, in summary, if the relationship between the two features I have is one, these features are directly proportional to one hundred percent. If this relationship is -1, these two are inversely proportional. If 0, they do not have a proportional relationship.

When we apply this process, we mentioned to the dataset we have, we will get the result in Char 1.2 on the bottom page. In order to see the results more clearly, I search for a clearer correlation by lowering the threshold to 0.75. You can see in Char 1.3 for this.

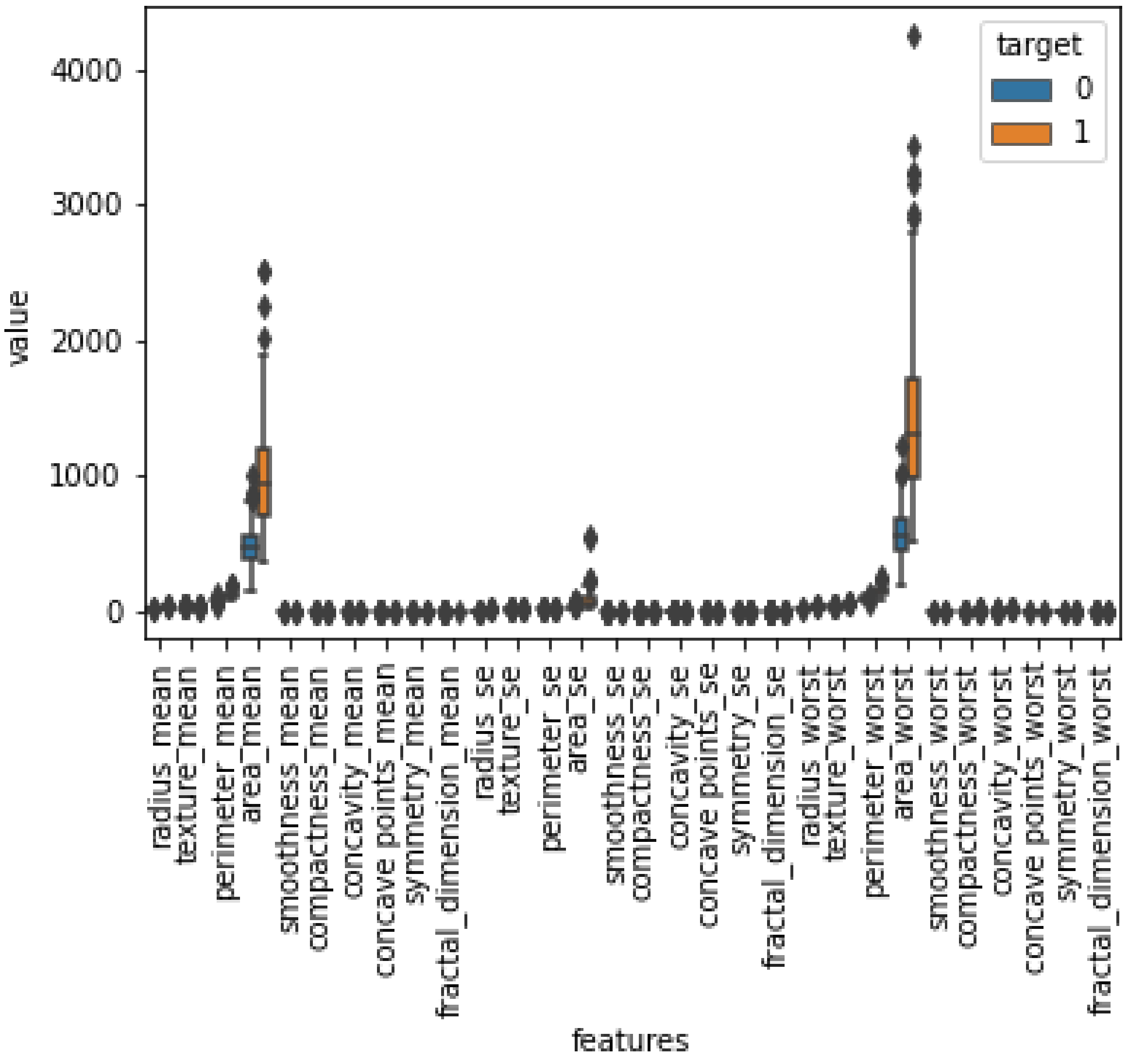


Char 1.2



Char 1.3

In this project, I realized that it would not be appropriate to draw a certain relational rule. I will re-examine how the data will yield when I visualize it with a box plot in Char 1.4. When I separated by target, I realized that I needed standardization or normalization to make sense of the data. this is because there are very high numbers in the scale.



Char 1.4

In order to standardize a data, I first need to remove the outlier ones, then pre-prepare the train test split operations and do the final standardization. So at this stage I will continue with outlier analysis.

2.3.1 Outlier Detection: Local Outlier Factor Method

Outlier analysis is the process of identifying outliers, or abnormal observations, in a dataset. Also known as outlier detection, it's an important step in data analysis, as it removes erroneous or inaccurate observations which might otherwise skew conclusions. There are a wide range of techniques and tools used in outlier analysis. However, it's often very easy to spot outlying data points. As a result, there's really no excuse not to perform outlier analysis on any and all datasets. Unlike other data analysis processes, outlier analysis only really has one benefit: it improves the quality of the dataset being subject to analysis. Of course, this in turn brings benefits. With a higher-quality dataset, analysts can expect to draw more accurate conclusions.A distance the threshold that can be defined as a reasonable neighborhood of the object. For each object that we can find a reasonable number of neighbors of an object. Density-based outlier detection method investigates the density of an object and that of its neighbors. After that, an object is identified as an outlier if its density is relatively much lower than that of its neighbors.

Many real-world data sets demonstrate a more complex structure, where objects may be considered outliers with respect to their local neighborhoods, rather than with respect to the global data distribution.

In this study, we will use the density based outlier detection method. Distance-based outlier detection method consults the neighborhood of an object, which is defined by a given radius. An object is then considered an outlier if its neighborhood does not have enough other points.

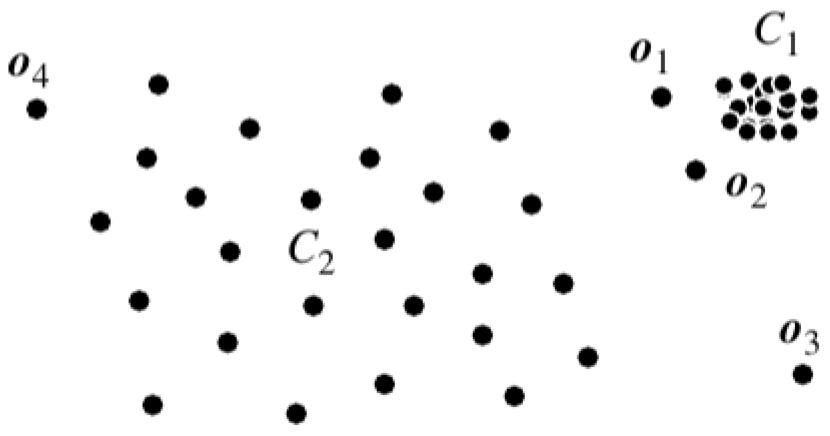


Image 1.3

Consider the example above, distance-based methods are able to detect o3, but as for o1 and o2 it is not as evident.

The idea of density-based is that we need to compare the density around an object with the density around its local neighbors. The basic assumption of density-based outlier detection methods is that the density around a nonoutlier object is similar to the density around its neighbors, while the density around an outlier object is significantly different from the density around its neighbors. dist_k(o) -is a distance between object o and k-nearest neighbors. The k-distance neighborhood of o contains all objects of the distance to o is not greater than dist_k(o) the k-th distance of o:

$$N_k(o) = [o' | o' \in D, dist(o, o') \leq dist_k(o)]$$

We can use the average distance from the objects in Nk(o) to o as the measure of the local density of o. If o has very close neighbors o’ such that dist(o,o’) is very small, the statistical fluctuations of the distance measure can be undesirably high. To overcome this problem, we can switch to the following reachability distance measure by adding a smoothing effect.

$$reachdist_k(o, o') = max[dist_k(o), dist(o, o')]$$

k is a user-specified parameter that controls the smoothing effect. Essentially, k specifies the minimum neighborhood to be examined to determine the local density of an object.Reachability distance is not symmetric.

Local reachability density of an object o is:

$$lrd_k(o) = \frac{\|N_k(o)\|}{\sum_{o' \in N_k(o)} reachdist_k(o, o')}$$

We calculate the local reachability density for an object and compare it with that of its neighbors to quantify the degree to which the object is considered an outlier.

$$LOF_k(\boldsymbol{o}) = \frac{\sum_{\boldsymbol{o'} \in N_k(\boldsymbol{o})} \frac{lrd_k(\boldsymbol{o'})}{lrd_k(\boldsymbol{o})}}{\|N_k(\boldsymbol{o})\|} = \sum_{\boldsymbol{o'} \in N_k(\boldsymbol{o})} lrd_k(\boldsymbol{o'}) \cdot \sum_{\boldsymbol{o'} \in N_k(\boldsymbol{o})} reachdist_k(\boldsymbol{o'} \leftarrow \boldsymbol{o}).$$

The local outlier factor is the average of the ratio of the local reachability density of o and those of o’s k-nearest neighbors. The lower local reachability density of o and the higher the local reachability densities of the k-nearest neighbors of o, the higher the LOF value is. This exactly captures a local outlier of which the local density is relatively low compared to the local densities of its k-nearest neighbors.

The local outlier factor is the average of the ratio of the local reachability density of o and those of o’s k-nearest neighbors. The lower local reachability density of o and the higher the local reachability densities of the k-nearest neighbors of o, the higher the LOF value is. This exactly captures a local outlier of which the local density is relatively low compared to the local densities of its k-nearest neighbors.

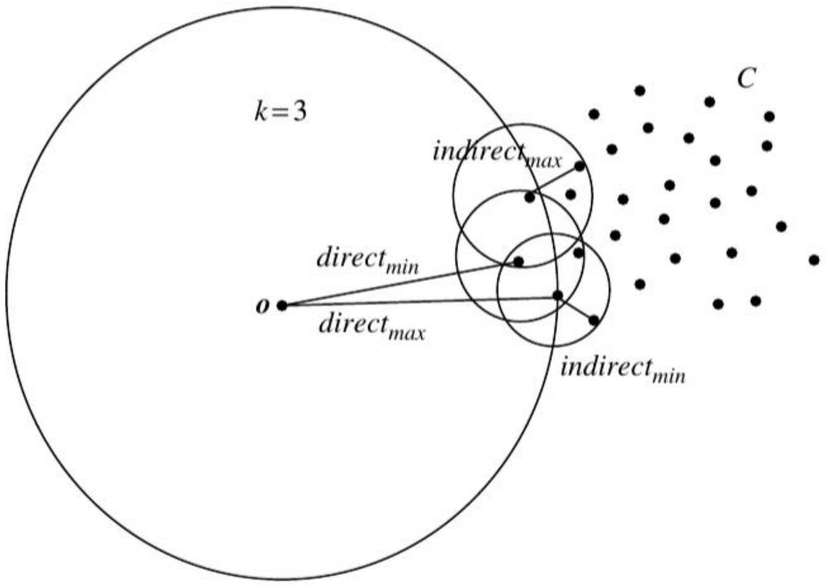


Image 1.4

LOF algorithm

Based on the clickstream event frequency pattern in data.csv, we will apply LOF algorithm to calculate LOF for each point with the following initial settings:

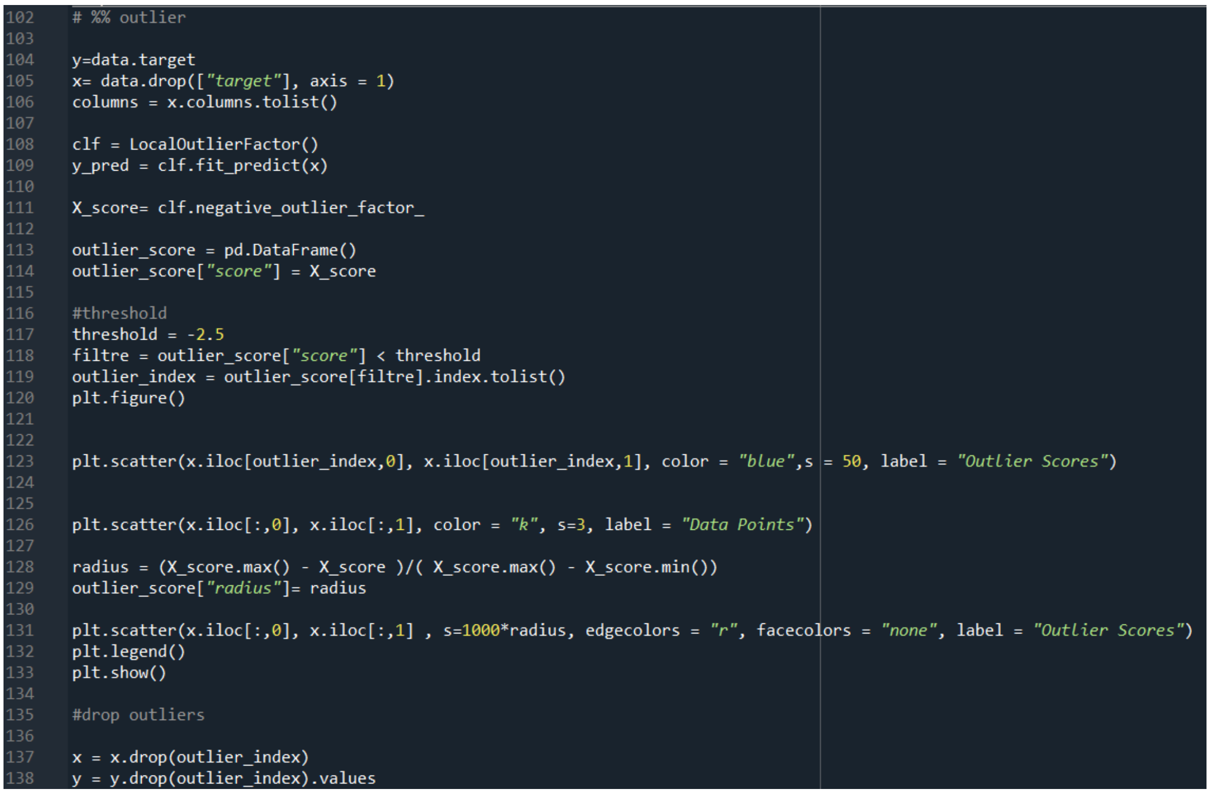
k = 2 and use Manhattan distance.

k = 3 and use Euclidean distance.

2.3.2 Detection and Removal of Outliers

Outliers are data points that are far from other data points. In other words, they’re unusual values in a dataset. Outliers are problematic for many statistical analyses because they can cause tests to either miss significant findings or distort real results. Unfortunately, there are no strict statistical rules for definitively identifying outliers. Finding outliers depends on subject-area knowledge and an understanding of the data collection process. While there is no solid mathematical definition, there are guidelines and statistical tests you can use to find outlier candidates.

In this part, I explained what outliers are and why they are problematic, and present various methods for finding them. Additionally, I close this post by comparing the different techniques for identifying outliers and share my preferred approach.



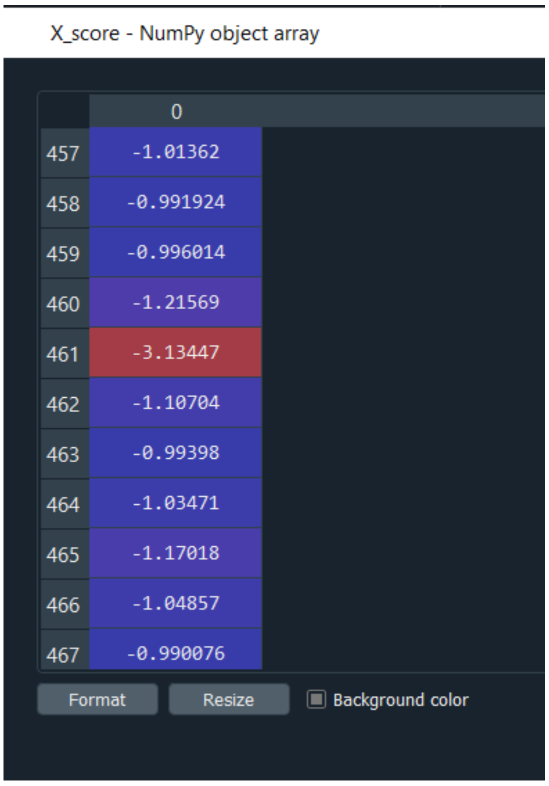
```
# %% outlier

y=data.target
x= data.drop(["target"], axis = 1)
columns = x.columns.tolist()

clf = LocalOutlierFactor()
y_pred = clf.fit_predict(x)

X_score= clf.negative_outlier_factor_

outlier_score = pd.DataFrame()
outlier_score["score"] = X_score

#threshold
threshold = -2.5
filtre = outlier_score["score"] < threshold
outlier_index = outlier_score[filtre].index.tolist()
plt.figure()


plt.scatter(x.iloc[outlier_index,0], x.iloc[outlier_index,1], color = "blue",s = 50, label = "Outlier Scores")


plt.scatter(x.iloc[:,0], x.iloc[:,1], color = "k", s=3, label = "Data Points")

radius = (X_score.max() - X_score )/( X_score.max() - X_score.min())
outlier_score["radius"]= radius

plt.scatter(x.iloc[:,0], x.iloc[:,1] , s=1000*radius, edgecolors = "r", facecolors = "none", label = "Outlier Scores")
plt.legend()
plt.show()

#drop outliers

x = x.drop(outlier_index)
y = y.drop(outlier_index).values
```

Image 1.5

I used the local outlier factor method from sklearn to visually express Image 1.5 and see the results while applying the algorithm that we specified in the upper step in the outlier analysis. The number of neighbors I can get is 20 by default. However, this will return something called a negative outlier factor to us and the reverse of the local outlier factor will be returned to me. Multiplying by -1, we get the result.

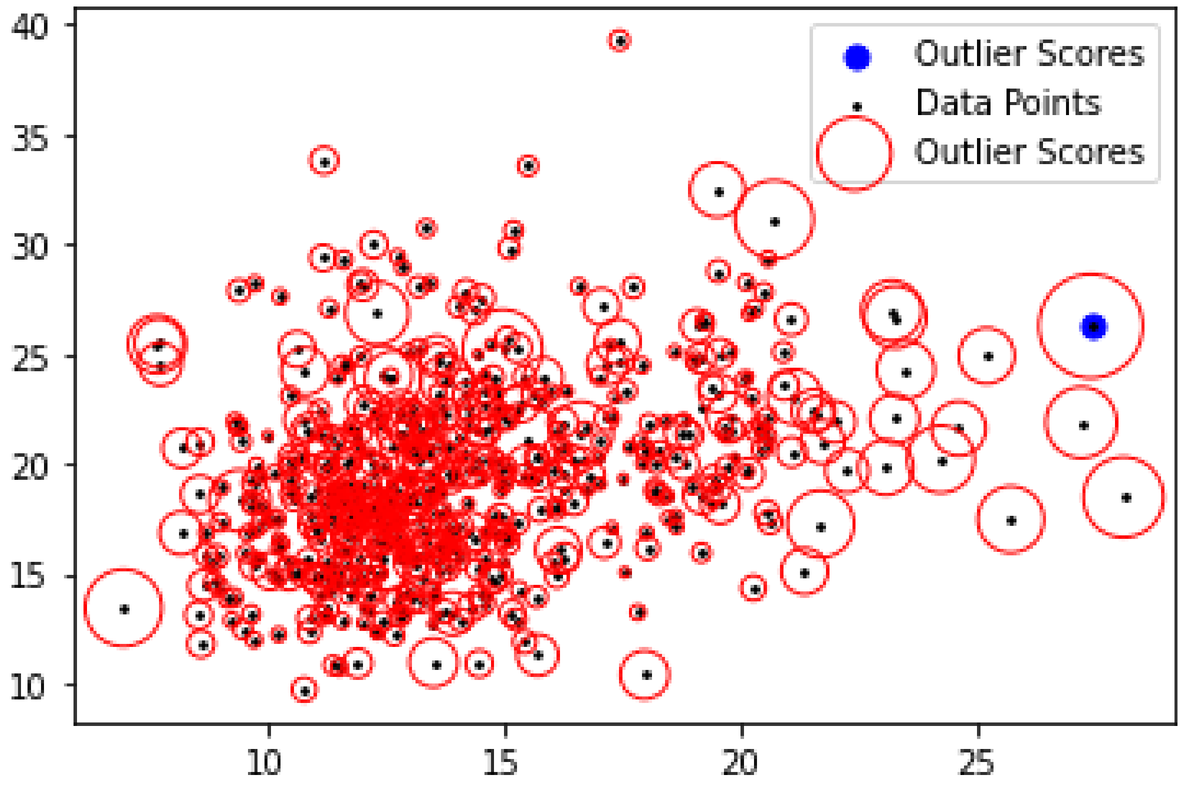
In order for us to be able to detect it as an outlier, the X score must be greater than -2.5. In order not to lose data, I determine this value myself according to the X score. It shown on Image 1.6.

As it can be seen in the Char 1.5, we have completed our outlier analysis with the threshold value we have chosen as a result of the analysis, and here we have found such a solution in order to be able to train in more accurate results by removing the unrelated and more irrelevant data group from the system. Outliers are a simple concept.They are values that are notably different from other data points, and they can cause problems in statistical procedures.To demonstrate how much a single outlier can affect the results, let’s examine the properties of an project dataset.


X_score - NumPy object array

	0
457	-1.01362
458	-0.991924
459	-0.996014
460	-1.21569
461	-3.13447
462	-1.10704
463	-0.99398
464	-1.03471
465	-1.17018
466	-1.04857
467	-0.990076

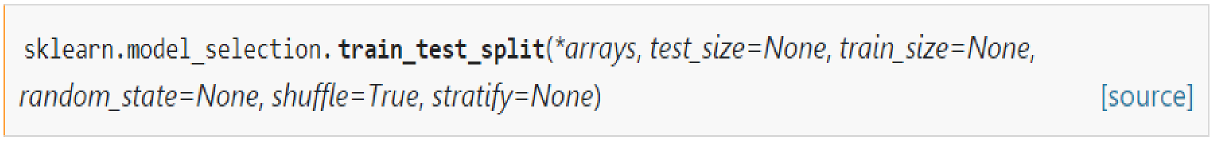
Image 1.6



Char 1.5

2.4. Separation of Data Set as Training and Test Data

After providing the necessary outlier cleaning, I started working for the next step, test and train. First of all, let's examine the concepts of train-test split respectively. The train-test split is a technique for evaluating the performance of a machine learning algorithm which shown Image 1.7. It can be used for classification or regression problems and can be used for any supervised learning algorithm.



```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None,
random_state=None, shuffle=True, stratify=None)                          [source]
```

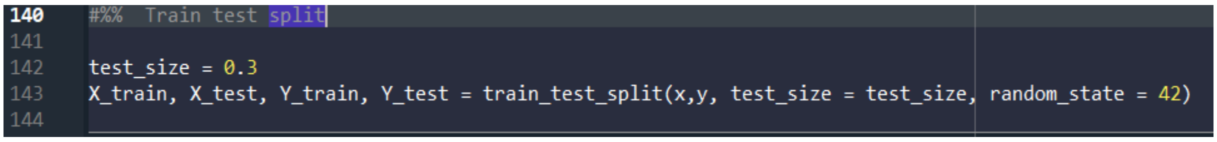
Image 1.7

Split arrays or matrices into random train and test subsets. Quick utility that wraps input validation and next(ShuffleSplit().split(X, y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

Train Dataset: Used to fit the machine learning model.

Test Dataset: Used to evaluate the fit machine learning model.



```
#%%  Train test split

test_size = 0.3
X_train, X_test, Y_train, Y_test = train_test_split(x,y, test_size = test_size, random_state = 42)

```

Image 1.8

The objective is to estimate the performance of the machine learning model on new data; data not used to train the model. Image 1.8 is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values. The train-test procedure is appropriate when there is a sufficiently large dataset available.

While performing this operation, I have certain parameters, let's call them x and y. Determining the size of the test, we chose it as 0.3. We create two trains and tests for X and Y. Since shuffle is done, in this sklearn method we will ensure that the default is not mixed by choosing true false.

2.5. Standardization

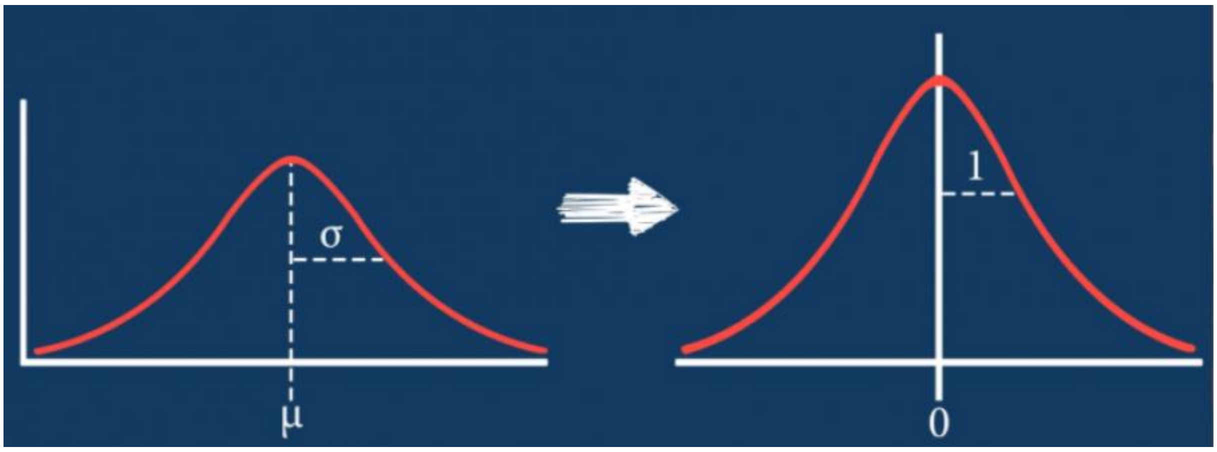


Image 1.9

Standardization is an important technique that is mostly performed as a pre-processing step before many Machine Learning models, to standardize the range of features of input data set like Image1.9.

Some ML developers tend to standardize their data blindly before “every” Machine Learning model without taking the effort to understand why it must be used, or even if it’s needed or not. So, the goal of our standardization part is to explain how, why and when to standardize data.

Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} (x_i)$$

and standard deviation

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2}$$

Standardization comes into picture when features of input data set have large differences between their ranges, or simply when they are measured in different measurement units (e.g., Pounds, Meters, Miles … etc).

These differences in the ranges of initial features causes trouble to many machine learning models. For example, for the models that are based on distance computation, if one of the features has a broad range of values, the distance will be governed by this particular feature.

If we ask why we need to standardize our ML model answers is: Variables that are measured at different scales do not contribute equally to the model fitting & model learned function and might end up creating a bias.

Thus, to deal with this potential problem feature-wise standardized (μ=0, σ=1) is usually used prior to model fitting. To do that using scikit-learn, we first need to construct an input array X containing the features and samples with X.shape being[number_of_samples, number_of_features] .

The main idea is to normalize/standardize i.e. μ = 0 and σ = 1 your features/variables/columns of X, individually, before applying any machine learning model. Thus, StandardScaler() will normalize the features i.e. each column of X, individually so that eachcolumn/feature/variable Imagewill have μ = 0 and σ = 1.

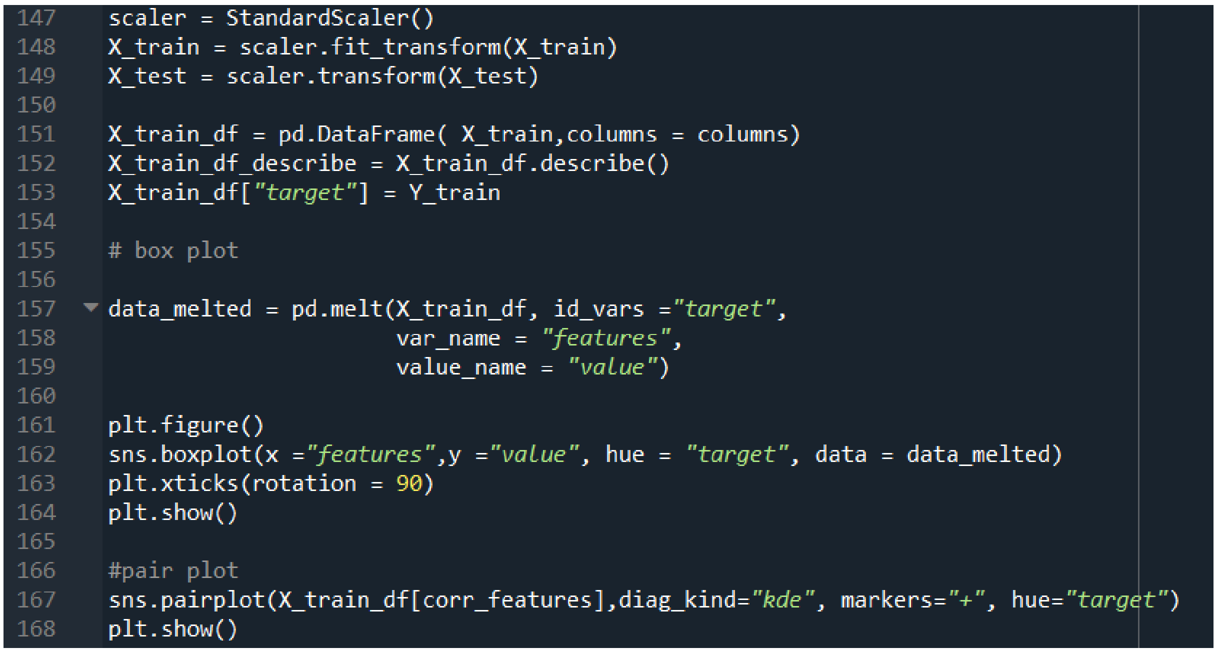
Z-score is one of the most popular methods to standardize data, and can be done by subtracting the mean and dividing by the standard deviation for each value of each feature.

$$z = \frac{value - mean}{standard\ deviation}$$

k-nearest neighbors is a distance based classifier that classifies new observations based on similarity measures (e.g., distance metrics) with labeled observations of the training set. Standardization makes all variables to contribute equally to the similarity measures

In Principal Component Analysis, features with high variances/wide ranges, get more weight than those with low variance, and consequently, they end up illegitimately dominating the First Principal Components (Components with maximum variance). I used the word “Illegitimately” here, because the reason these features have high variances compared to the other ones is just because they were measured in different scales. Standardization can prevent this, by giving same weightages to all features.

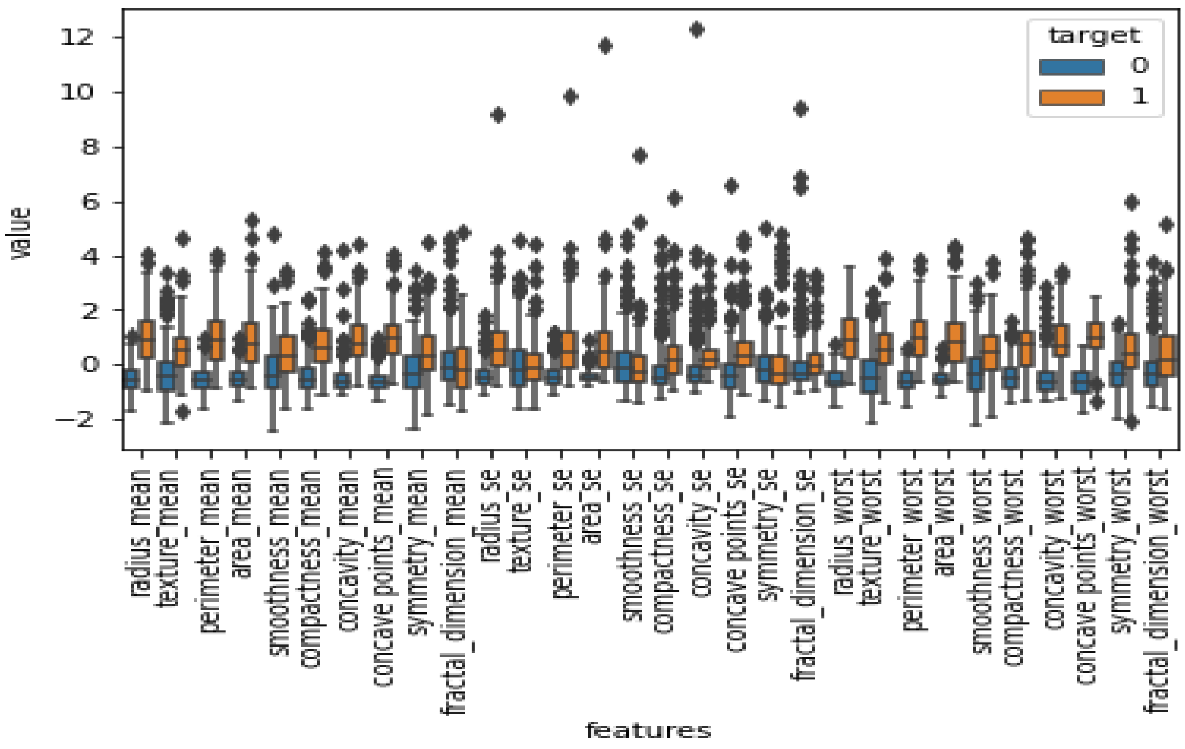
In my dataset, the data value gap was too much, ignoring this could ignore some features such as smootless in the examination. To balance them, I standardized my gaussian distribution codes with the following lines of code Image 2.0.



```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train_df = pd.DataFrame( X_train,columns = columns)
X_train_df_describe = X_train_df.describe()
X_train_df["target"] = Y_train

# box plot

data_melted = pd.melt(X_train_df, id_vars ="target",
                      var_name = "features",
                      value_name = "value")

plt.figure()
sns.boxplot(x ="features",y ="value", hue = "target", data = data_melted)
plt.xticks(rotation = 90)
plt.show()

#pair plot
sns.pairplot(X_train_df[corr_features],diag_kind="kde", markers="+", hue="target")
plt.show()
```

Image 2.0

As it can be seen clearly in Char 1.6, we have transformed my values into more usable standard forms, such as 0 and 1, and turned them into a more understandable structure than our previous chard.



Char 1.6

2.6.1 What is K-Nearest Neighbors (KNN)?

The K-NN (K-Nearest Neighbor) algorithm is one of the simplest and most used classification algorithms. K-NN is a non-parametric, lazy learning algorithm. If we try to understand the concept of lazy, unlike eager learning, lazy learning does not have a training phase. It does not learn the training data, but instead “memorizes” the training dataset. When we want to make a prediction, it looks for nearest neighbors in the whole dataset.

K value is determined in the operation of the algorithm. This K value means the number of elements to look at. When a value arrives, the distance between the incoming value is calculated by taking the nearest K number of elements. The Euclidean function is generally used in distance calculation. As an alternative to the Euclidean function, Manhattan, Minkowski and Hamming functions can also be used. After the distance is calculated, it is sorted and the value is assigned to the appropriate class.

The k-nearest neighbor (KNN) algorithm is one of the supervised learning algorithms that is easy to implement. Although it is used in solving both classification and regression problems, it is mostly used in solving classification problems in industry.

KNN algorithms were proposed by T. M. Cover and P. E. Hart in 1967. The algorithm is used by making use of the data in a sample set with certain classes. The distance of the new data to be added to the sample data set is calculated according to the existing data, and its k number of close neighbors are checked. Three types of distance functions are generally used for distance calculations: “Euclidean” distance, “Manhattan” distance, "Minkowski" distance.

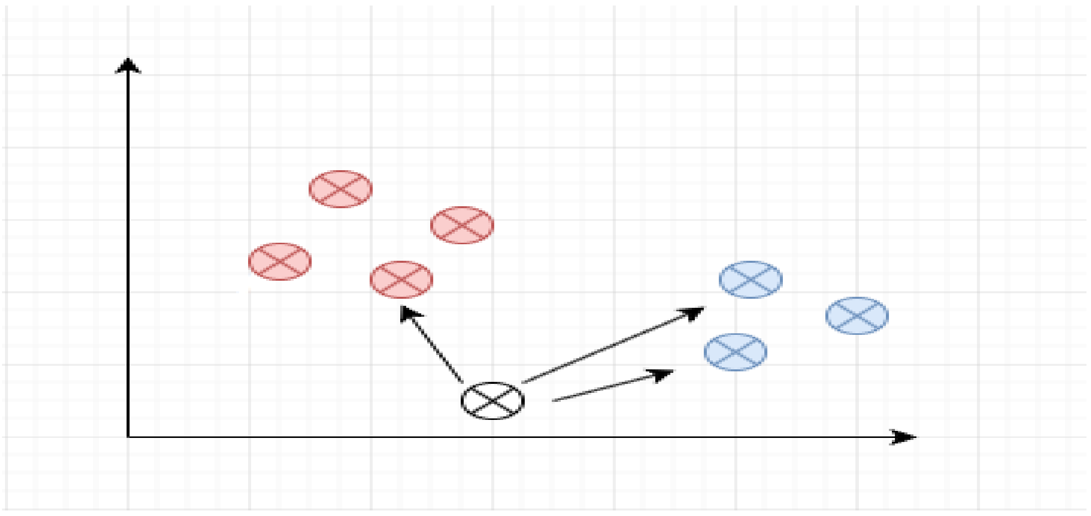


Image 2.1

KNN; It is one of the most popular machine learning algorithms due to its resistance to old, simple and noisy training data. But it also has a disadvantage. For example, it needs a lot of memory space when used for large data, since it stores all states when calculating distances.

The steps of the KNN algorithm:

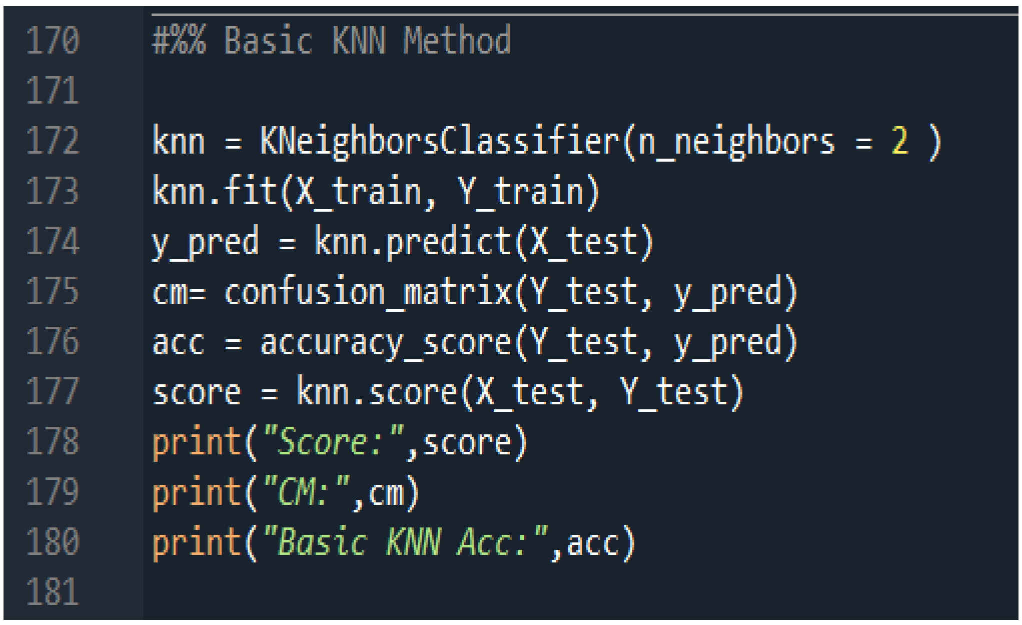
-First, the parameter k is determined. This parameter is the number of nearest neighbors to a given point. For example: Let k = 2. In this case, classification will be made according to the 2 closest neighbors.

-The distance of the new data, which will be added to the sample data set, according to the existing data is calculated one by one. With the help of the corresponding distance functions.

-The k nearest neighbors of the related distances are considered. It is assigned to the class of k neighbors or neighbors according to the attribute values.

-The selected class is considered to be the class of the observation value expected to be estimated. In other words, the new data is labeled.

In this study, the "scikit-learn" module of the python programming language was used. “scikit-learn” is a machine learning module built from Numpy, SciPy and Matplotlib modules. Scikit-learn provides simple and efficient tools for data analysis such as classification, regression, clustering, size reduction, model selection, and data preprocessing. My code is shown in Image 2.2.



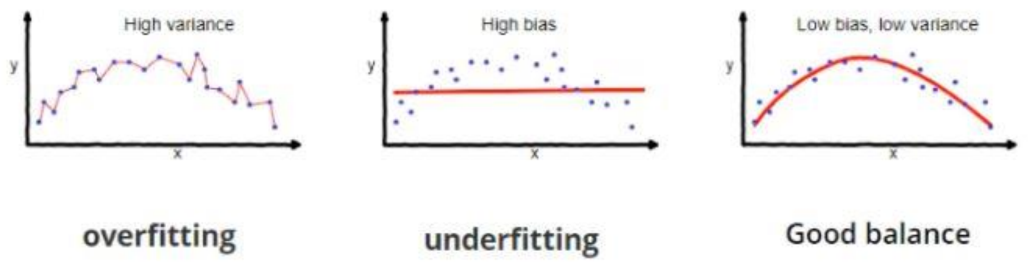
```
#%% Basic KNN Method

knn = KNeighborsClassifier(n_neighbors = 2 )
knn.fit(X_train, Y_train)
y_pred = knn.predict(X_test)
cm= confusion_matrix(Y_test, y_pred)
acc = accuracy_score(Y_test, y_pred)
score = knn.score(X_test, Y_test)
print("Score:",score)
print("CM:",cm)
print("Basic KNN Acc:",acc)

```

Image 2.2

2.6.2 Finding the Best Parameters of KNN

In the previous stage, we achieved a 95 percent success rate after the KNN procedure. I explained this success rate in detail in the results section in the next section. At this stage, if we give the best value to the neighborhood, it will work more efficiently and we will see if we can achieve results that do not exceed the one percent error rate we want. For this, we have to check whether our KNN model is underfit or overfit. If the data we have is overfit, the data will memorize the values. That means high variance. When the new data set arrives, the system decreases the success of correct conclusion due to memorization. With underfitting, we fit a model and this model cannot perform the fitting operation and it still has unsuccessful results, so we used a good balance model.

The desired thing should be low variance and low bias. We tried to reach the values we wanted by playing with the parameters. It is shown on Image 2.3:



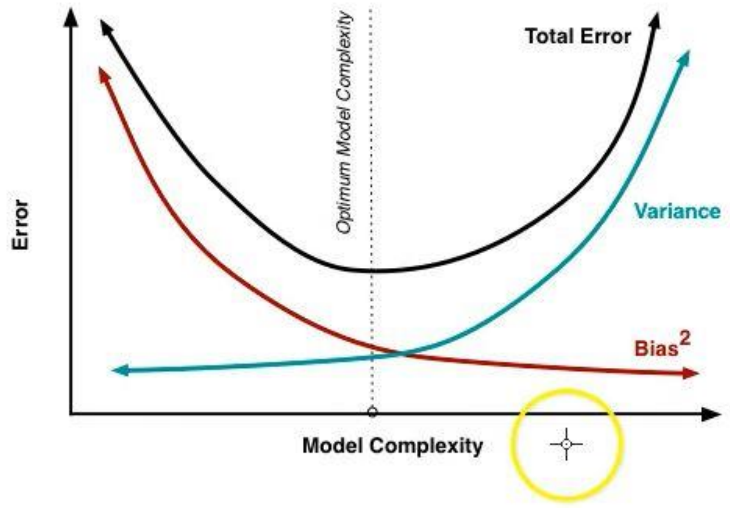
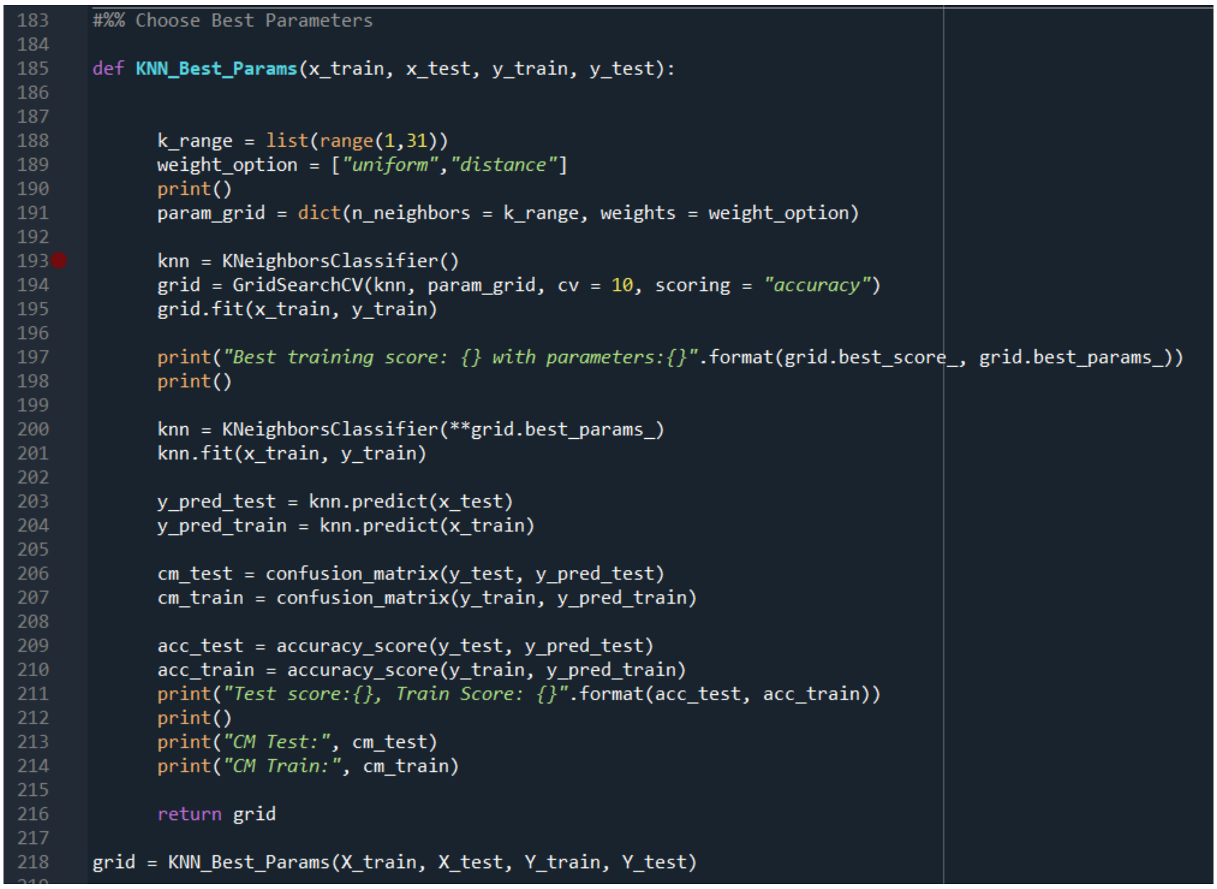


Image 2.3



```
#%% Choose Best Parameters

def KNN_Best_Params(x_train, x_test, y_train, y_test):


    k_range = list(range(1,31))
    weight_option = ["uniform","distance"]
    print()
    param_grid = dict(n_neighbors = k_range, weights = weight_option)

    knn = KNeighborsClassifier()
    grid = GridSearchCV(knn, param_grid, cv = 10, scoring = "accuracy")
    grid.fit(x_train, y_train)

    print("Best training score: {} with parameters:{}".format(grid.best_score_, grid.best_params_))
    print()

    knn = KNeighborsClassifier(**grid.best_params_)
    knn.fit(x_train, y_train)

    y_pred_test = knn.predict(x_test)
    y_pred_train = knn.predict(x_train)

    cm_test = confusion_matrix(y_test, y_pred_test)
    cm_train = confusion_matrix(y_train, y_pred_train)

    acc_test = accuracy_score(y_test, y_pred_test)
    acc_train = accuracy_score(y_train, y_pred_train)
    print("Test score:{}, Train Score: {}".format(acc_test, acc_train))
    print()
    print("CM Test:", cm_test)
    print("CM Train:", cm_train)

    return grid

grid = KNN_Best_Params(X_train, X_test, Y_train, Y_test)
```

Image 2.4

K value indicates the count of the nearest neighbors. We have to compute distances between test points and trained labels points. Updating distance metrics with every iteration is computationally expensive, and that’s why KNN is a lazy learning algorithm.

If we ask how, we can find best parameters: There are no pre-defined statistical methods to find the most favorable value of K. Initialize a random K value and start computing. Choosing a small value of K leads to unstable decision boundaries. The substantial K value is better for classification as it leads to smoothening the decision boundaries.

Derive a plot between error rate and K denoting values in a defined range. Then choose the K value as having a minimum error rate.

In Image 2.4 way, after writing a code, I was able to increase the KNN effect by a maximum of 94 percent with the best parameters I needed. While this wasn’t better than our previous work, it was a loss of at least 99 more than we wanted so in the next step I will try to achieve better results with PCA analysis.

2.7.1 What is Principal Component Analysis (PCA)?

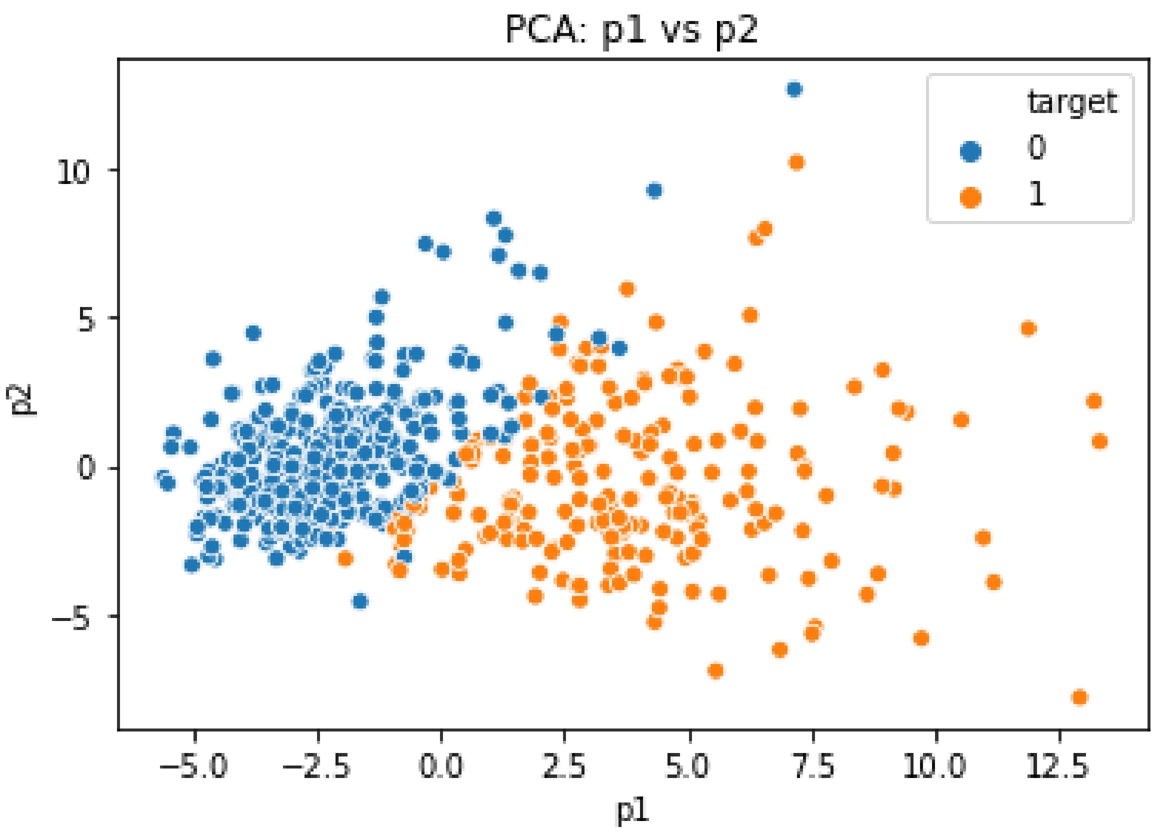
The principal components of a collection of points in a real coordinate space are a sequence of {\displaystyle p}p unit vectors, where the {\displaystyle i}i-th vector is the direction of a line that best fits the data while being orthogonal to the first {\displaystyle i-1}i-1 vectors. Here, a best-fitting line is defined as one that minimizes the average squared distance from the points to the line. These directions constitute an orthonormal basis in which different individual dimensions of the data are linearly uncorrelated. Principal component analysis (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.

PCA is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data. The {\displaystyle i}i-th principal component can be taken as a direction orthogonal to the first {\displaystyle i-1}i-1 principal components that maximizes the variance of the projected data.

From either objective, it can be shown that the principal components are eigenvectors of the data's covariance matrix. Thus, the principal components are often computed by eigen decomposition of the data covariance matrix or singular value decomposition of the data matrix. PCA is the simplest of the true eigenvector-based multivariate analyses and is closely related to factor analysis. Factor analysis typically incorporates more domain specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix. PCA is also related to canonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe the cross-covariance between two datasets while PCA defines a new orthogonal coordinate system that optimally describes variance in a single dataset. Robust and L1-norm-based variants of standard PCA have also been proposed.

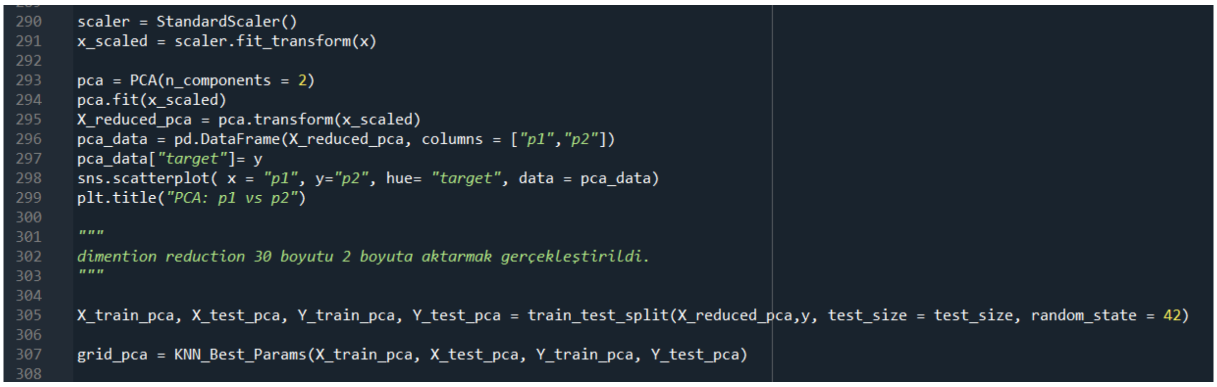
PCA reduces data size by keeping as much data as possible. It does this by reducing dimensions. We will use what kind of result we will get by reducing the number of features. At the same time, since there are correlated features, we will do PCA analysis to eliminate them. The reason we do this is that we will also reduce dimensions to visualize our work. We have done the reduction from thirty dimensions to 2 dimensions. And we'll look at how that results. I begin this by finding eigen vectors.

2.7.2 Principal Component Analysis Application



Char 1.6

While doing this, I actually find the averages of the axes and subtract them from them, so I move this data to the center. I continue with finding the covariance of the data coming to the point (0,0). In fact, it means their relationship with each other. We're going to get a two-by-two matrix from here. I create the components by taking the necessary codes from the page documentation to find the covariance matrix.



```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

pca = PCA(n_components = 2)
pca.fit(x_scaled)
X_reduced_pca = pca.transform(x_scaled)
pca_data = pd.DataFrame(X_reduced_pca, columns = ["p1","p2"])
pca_data["target"]= y
sns.scatterplot( x = "p1", y="p2", hue= "target", data = pca_data)
plt.title("PCA: p1 vs p2")

"""
dimention reduction 30 boyutu 2 boyuta aktarmak gerçekleştirildi.
"""

X_train_pca, X_test_pca, Y_train_pca, Y_test_pca = train_test_split(X_reduced_pca,y, test_size = test_size, random_state = 42)

grid_pca = KNN_Best_Params(X_train_pca, X_test_pca, Y_train_pca, Y_test_pca)
```

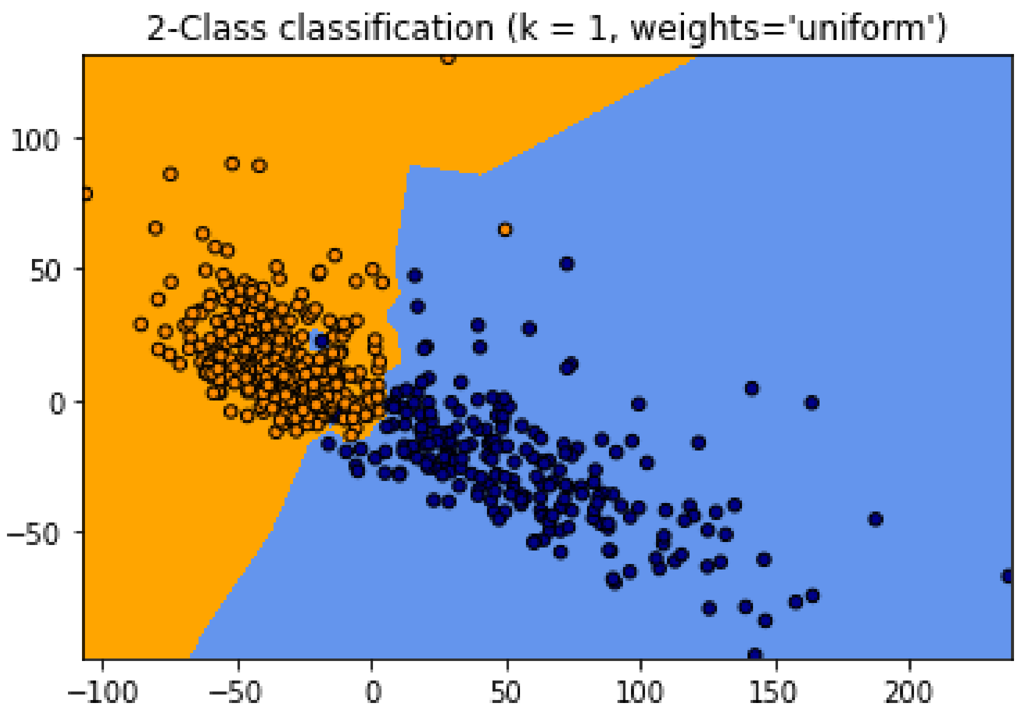
Image 2.5

2.8 Neighborhood Components Analysis and Application (NCA)

Neighbourhood components analysis is a supervised learning method for classifying multivariate data into distinct classes according to a given distance metric over the data. Functionally, it serves the same purposes as the K-nearest neighbors algorithm, and makes direct use of a related concept termed stochastic nearest neighbours.

Neighbourhood components analysis aims at "learning" a distance metric by finding a linear transformation of input data such that the average leave-one-out (LOO) classification performance is maximized in the transformed space. The key insight to the algorithm is that a matrix {\displaystyle A}A corresponding to the transformation can be found by defining a differentiable objective function for {\displaystyle A}A, followed by use of an iterative solver such as conjugate gradient descent. One of the benefits of this algorithm is that the number of classes {\displaystyle k}k can be determined as a function of {\displaystyle A}A, up to a scalar constant. This use of the algorithm therefore addresses the issue of model selection.

Learning by itself by finding linear transformations on NCA input data. This process gave us the most successful result and I explained it in detail in the conculusion section.



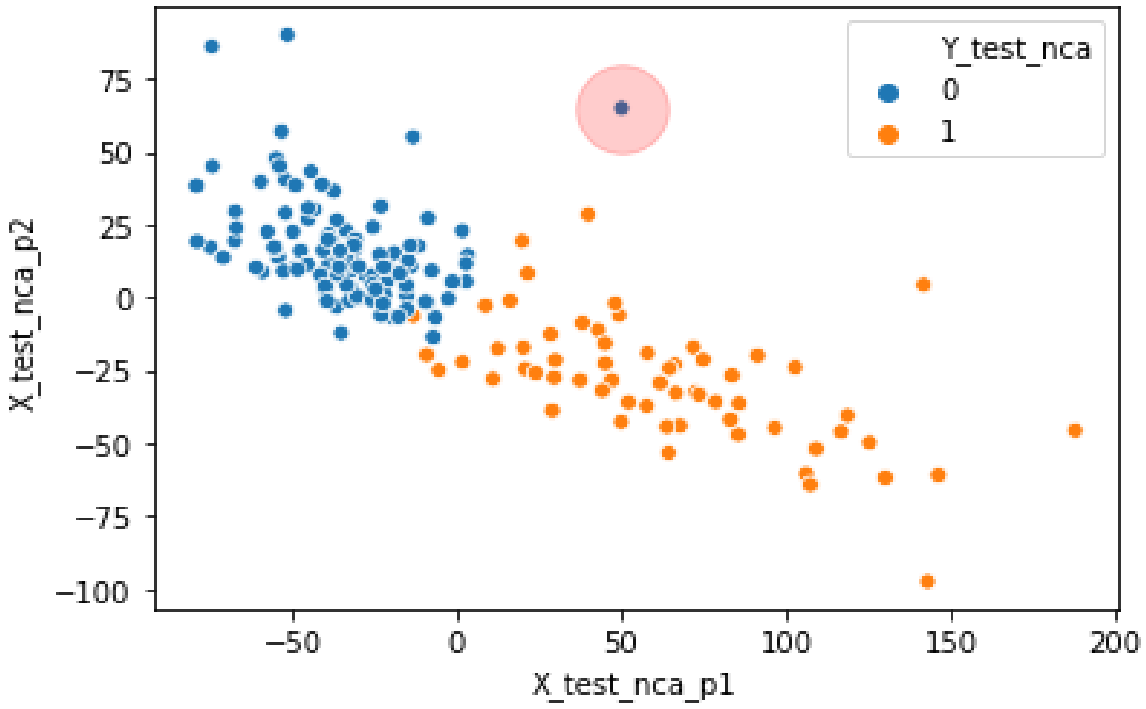
Char 1.7

3- CONCLUSION AND DISCUSSION

At this stage, we will evaluate all the analyzes we have done in detail. First of all, we found different percentages of success with four different analyzes. While doing this, we first tried to work with the KNN algorithm. This was a promising 95 percent for our first experience, but on the other hand, it was insufficient for us. When we determined the correct variable values with KNNBP, we wanted to prevent overfitting in KNN. Although we reduced overfitting, we actually achieved a worse result than the previous analysis, with a 94 percent success rate as a result of this step.

In this study, which has 30 features, we tried PCA analysis to transfer and visualize data on the binary plane. This showed us sharply, with a result of 92 percent, that reducing the features actually takes us away from the correct result. Then, when we tried the NCA analysis, we achieved a 99 percent success rate when we tried the neighbor relations and the grouping trend by calculating the distance between neighbors with vector lengths. Until this stage, we understood our data and its returns correctly with different analyzes and achieved a high success in breast cancer detection.

It was determined that only 1 data was analyzed incorrectly in the NCA visualization graphics, which achieved 99 percent success. And I showed you this with Char 1.8.



Char 1.8

RESOURCES

“breast-cancer-wisconsin-data” Accessed March 15, 2021.

http://www.kaggle.com.

“Neighbourhood_components_analysis” Accessed March 23, 2021.

https://en.wikipedia.org/

“standart-test-sonuc-analizi-hazirligi” Accessed March 27, 2021.

http://www.ncaworld.org/

“Neighbourhood_components_analysis” Accessed April 3, 2021.

https://scikit-learn.org/

“How to find the optimal value of K in KNN? Accessed April 8, 2021.

https://towardsdatascience.com/

“GridSearchCV Basics” Accessed April 13, 2021.

https://medium.com/

“stable/tutorials/introductory/pyplot” Accessed April 17, 2021.

https://matplotlib.org

“Standardizasyon” Accessed April 23, 2021.

https://veribilimcisi.com/

“5 Ways to Find Outliers in Your Data” Accessed May 10, 2021.

https://statisticsbyjim.com

“Pandas Library” Accessed May 17, 2021.

https://pandas.pydata.org

“Data Visualization” Accessed May 23, 2021.

https://www.analyticsvidhya.com

“numpy_random_seaborn” Accessed May 28, 2021.

https://www.w3schools.com

THANK

I would like to thank Associate Professor Doctor Zekeriya Tüfekci for accepting me as a project student at first, allowing me to work on the project I wanted, answering my questions about the subject, and helping me improve myself with his lectures as I approach my graduation today.

Then I would like to thank K. P. Bennett and O. L. Mangasarian who prepared the database that allowed me to start doing this study while doing all my research. I would also like to thank all the bibliography owners and bibliography site founders that I used as I got stuck in the process of my project.

In addition, I would like to thank all the academicians of Çukurova University, the computer engineering department, my mother who supported me in my thesis writing, and my friends who were with me.

RESUME

Personal Information

Name Surname	:	Ali Doğan
Birth place and date	:	Mersin/Türkiye – 09/03/1998

Education Status

High School	:	Toros Science High School
Bachelor Education	:	Çukurova University, Computer Engineering
Foreign Languages	:	Turkish(Native), English, Spanish,

Work Experience

Internships	:	Badem Bilgi Sistemleri, Binovist
Projects	:	AHBS Family Doctor System, Travelers App
Working Languages	:	Phyton, Java, Microsoft SQL, Firebase

Contact

Address	:	Osmaniye Neigbourhood 81006 Street Doğan Apartment No: 10 Flat 4 Mersin / Toroslar
Phone Number	:	+90 553 820 4757
Email	:	Doganaliofficial@gmail.com

Foreign Language Knowledge

English	:	Writing: good	Speaking: good
Turkish	:	Writing: perfect	Speaking: perfect
Spanish	:	Writing: low	Speaking: low