End of course summative assessment:

Machine Learning Approaches for Breast Cancer Detection: A Comparative Study

By Kai Bowers

Student Number: 230319794
Course: CSM010-2023-APR

2086 Words

Table of Contents

Chapter 1.1 : Introduction : Research motivation

With approximately 72,000 annual cases, breast cancer is by far the most common cancer in women. [1]

It is complicated and can be caused by many factors, like genes, environment and lifestyle. Machine learning can help with studying breast cancer, as it can analyze patterns and connections that humans might miss. AI can help us learn more about the disease, diagnose it accurately, and make better treatment decisions.
One important thing in breast cancer is finding it early. When we find breast cancer early, it's easier to treat and can save lives. AI and machine learning can look at pictures of the breast, like mammograms or ultrasounds, and find signs that might mean cancer is there.
AI and machine learning are also helpful in predicting what might happen with the disease. Doctors want to know how fast the cancer might grow or if it will come back after treatment. Using AI, scientists can analyze a lot of information, like genes and clinical data, to make predictions. This helps doctors make better decisions about treatment and tell patients what to expect.

In this paper, I am going to explore the dataset given by UI Irvine titled “Breast Cancer Wisconsin, advanced models for cancer detection". I will focus on utilizing the Breast Cancer Wisconsin Diagnostic dataset [2] to train multiple machine learning models capable of detecting breast cancer. The dataset provides us with 30 features which were computed from three digitized images of a fine needle aspirate (FNA) of a breast mass. The classification of each data point (row) will be a diagnoses of eighter M (malignant) or B (benign). There are 569 rows in the dataset.
The ten features of a single image are : radius , texture, perimeter , area , smoothness , compactness , concavity , concave points , symmetry, fractal dimension. We are given all ten features for each of the three images. There is also a row for a ID and the diagnoses.

Chapter 1.2 : Introduction : Local setup

I created all code locally using VS Code, Python 3 and Windows 11.
To run, simply clone the repo, install python and the required PIP packages.

All following screenshots of the jupyter notebooks were created by me and are visible in the repository. [3]

Chapter 2.1 : The data : Preparing the dataset

The raw data is given to us in the "wdbc.data"-file. In the jupyter notebook "Data pre-processing" I prepared the data for my machine learning algorithms. First, I created a copy of the file and added the feature labels as the first row of the CSV dataset. The features are available at the source of the dataset. [2]

```
# The goal of this Notebook is the process the wdbc.data into a wdbc.csv
```

```
# First, Import Libraries
```

```
import shutil
import pandas as pd
```

```
# Copy The Original Raw Dataset to "wdbc.csv"
```

```
shutil.copyfile("wdbc.data", "wdbc.csv")
```

```
'wdbc.csv'
```

```
# These Features are defined at https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic
```

```
Features = "ID,Diagnosis,radius1,texture1,perimeter1,area1,smoothness1,compactness1,concavity1,concave_points1,symmetry1,fractal_
```

```
# Add The Features as the first row of the CSV
```

```
with open('wdbc.csv', 'r') as original: data = original.read()
with open('wdbc.csv', 'w') as modified: modified.write(Features + "\n" + data)
```

I then removed the ID collum and normalized the data. There was no need for the ID field as it didn't provide additional information to the models. The source assured me that all values were filled and none were missing, which I double checked and confirmed to be correct.

```
In [10]: # Remove the ID Collum
```

```
In [11]: df = pd.read_csv('wdbc.csv')
         first_column = df.columns[0]
         df = df.drop([first_column], axis=1)
         df.to_csv('wdbc.csv', index=False)
```

```
In [12]: # Normalize each collum
```

```
In [13]: import pandas as pd
         from sklearn.preprocessing import MinMaxScaler

         data = pd.read_csv('wdbc.csv')
         cols_to_normalize = data.columns[1:]
         scaler = MinMaxScaler()
         data[cols_to_normalize] = scaler.fit_transform(data[cols_to_normalize])
         data.to_csv('wdbc.csv', index=False)
```

```
In [ ]: # Check if every value in the csv is set
```

```
In [15]: df = pd.read_csv("wdbc.csv")
         is_empty = df.isnull().values.any()
         if is_empty:
             print("There are missing values in the CSV file.")
         else:
             print("All values in the CSV file are set.")
```

```
All values in the CSV file are set.
```

Running this Jupyter Notebook creates the wdbc.csv which will be used in the next tasks.

Chapter 2.2 : The data : Exploring the dataset

Before starting the machine learning process, I wanted to understand the dataset.
To do this I created the "Data understanding.ipynb" notebook in which I ran multiple tests that would help me decide which type of learning algorithms to use for my task.

```
# Imports

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns #Install
from scipy.stats import chi2_contingency, ttest_ind
```

```
# Print the Head of the CSV

data = pd.read_csv('wdbc.csv')
print(data.head())
```

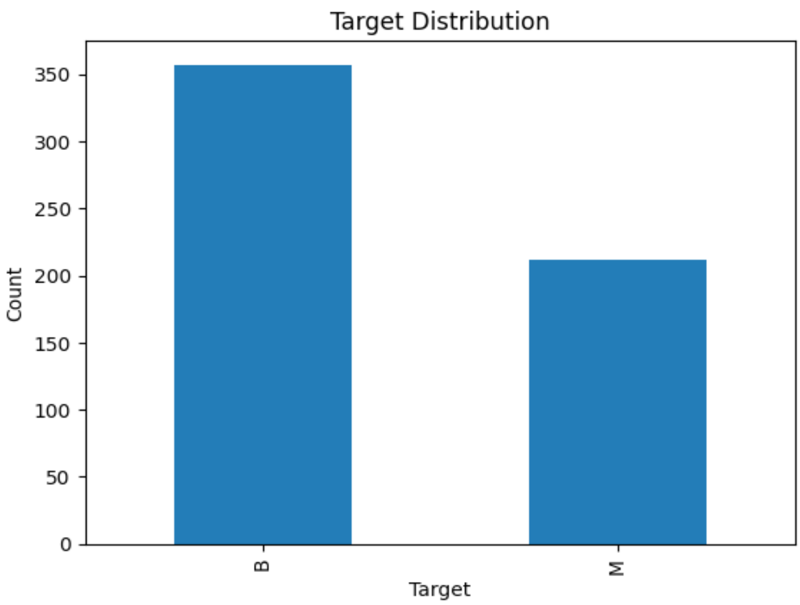
```
  Diagnosis   radius1  texture1  perimeter1     area1  smoothness1  \
0         M  0.521037  0.022658    0.545989  0.363733     0.593753
1         M  0.643144  0.272574    0.615783  0.501591     0.289880
2         M  0.601496  0.390260    0.595743  0.449417     0.514309
3         M  0.210090  0.360839    0.233501  0.102906     0.811321
4         M  0.629893  0.156578    0.630986  0.489290     0.430351

   compactness1  concavity1  concave_points1  symmetry1  ...   radius3  \
0      0.792037    0.703140         0.731113   0.686364  ...  0.620776
1      0.181768    0.203608         0.348757   0.379798  ...  0.606901
2      0.431017    0.462512         0.635686   0.509596  ...  0.556386
3      0.811361    0.565604         0.522863   0.776263  ...  0.248310
4      0.347893    0.463918         0.518390   0.378283  ...  0.519744

   texture3  perimeter3     area3  smoothness3  compactness3  concavity3  \
0  0.141525    0.668310  0.450698     0.601136      0.619292    0.568610
1  0.303571    0.539818  0.435214     0.347553      0.154563    0.192971
2  0.360075    0.508442  0.374508     0.483590      0.385375    0.359744
3  0.385928    0.241347  0.094008     0.915472      0.814012    0.548642
4  0.123934    0.506948  0.341575     0.437364      0.172415    0.319489

   concave_points3  symmetry3  fractal_dimension3
0         0.912027   0.598462            0.418864
1         0.639175   0.233590            0.222878
2         0.835052   0.403706            0.213433
3         0.884880   1.000000            0.773711
4         0.558419   0.157500            0.142595

[5 rows x 31 columns]
```

Calling print(data.head()) allows us to view the first five lines of the normalized data.

```
# Summary statistics of the data
print(data.describe())
```

```
         radius1    texture1  perimeter1       area1  smoothness1  \
count 569.000000  569.000000  569.000000  569.000000   569.000000
mean    0.338222    0.323965    0.332935    0.216920     0.394785
std     0.166787    0.145453    0.167915    0.149274     0.126967
min     0.000000    0.000000    0.000000    0.000000     0.000000
25%     0.223342    0.218465    0.216847    0.117413     0.304595
50%     0.302381    0.308759    0.293345    0.172895     0.390358
75%     0.416442    0.408860    0.416765    0.271135     0.475490
max     1.000000    1.000000    1.000000    1.000000     1.000000

       compactness1  concavity1  concave_points1  symmetry1  \
count    569.000000  569.000000       569.000000 569.000000
mean       0.260601    0.208058         0.243137   0.379605
std        0.161992    0.186785         0.192857   0.138456
min        0.000000    0.000000         0.000000   0.000000
25%        0.139685    0.069260         0.100944   0.282323
50%        0.224679    0.144189         0.166501   0.369697
75%        0.340531    0.306232         0.367793   0.453030
max        1.000000    1.000000         1.000000   1.000000

       fractal_dimension1  ...     radius3    texture3  perimeter3  \
count          569.000000  ...  569.000000  569.000000  569.000000
mean             0.270379  ...    0.296663    0.363998    0.283138
std              0.148702  ...    0.171940    0.163813    0.167352
min              0.000000  ...    0.000000    0.000000    0.000000
25%              0.163016  ...    0.180719    0.241471    0.167837
50%              0.243892  ...    0.250445    0.356876    0.235320
75%              0.340354  ...    0.386339    0.471748    0.373475
max              1.000000  ...    1.000000    1.000000    1.000000
```

In [5]:
```
# Visualize the target distribution

data['Diagnosis'].value_counts().plot(kind='bar')
plt.title('Target Distribution')
plt.xlabel('Target')
plt.ylabel('Count')
plt.show()
```



It is important to note that the amount of targets resulting in "B" was almost double the amount of targets resulting in "M". This made me want to analyze both outputs using t-tests and chi-square tests.

```
In [9]: # t-tests

numerical_cols = data.select_dtypes(include='number').columns
for col in numerical_cols:
    target_M = data[data['Diagnosis'] == 'M'][col]
    target_B = data[data['Diagnosis'] == 'B'][col]
    t_stat, p_value = ttest_ind(target_M, target_B)
    print(f'T-test for column {col}:')
    print(f'T-statistic: {t_stat}')
    print(f'P-value: {p_value}')
    print('---')
```

```
T-test for column radius1:
T-statistic: 25.43582161005704
P-value: 8.465940572264348e-96
---
T-test for column texture1:
T-statistic: 10.867201081464334
P-value: 4.058636047898335e-25
---
T-test for column perimeter1:
T-statistic: 26.405212979192687
P-value: 8.436251036172328e-101
---
T-test for column area1:
T-statistic: 23.93868723569098
P-value: 4.734564310307834e-88
---
T-test for column smoothness1:
T-statistic: 9.14609880814903
P-value: 1.051850359203269e-18
```

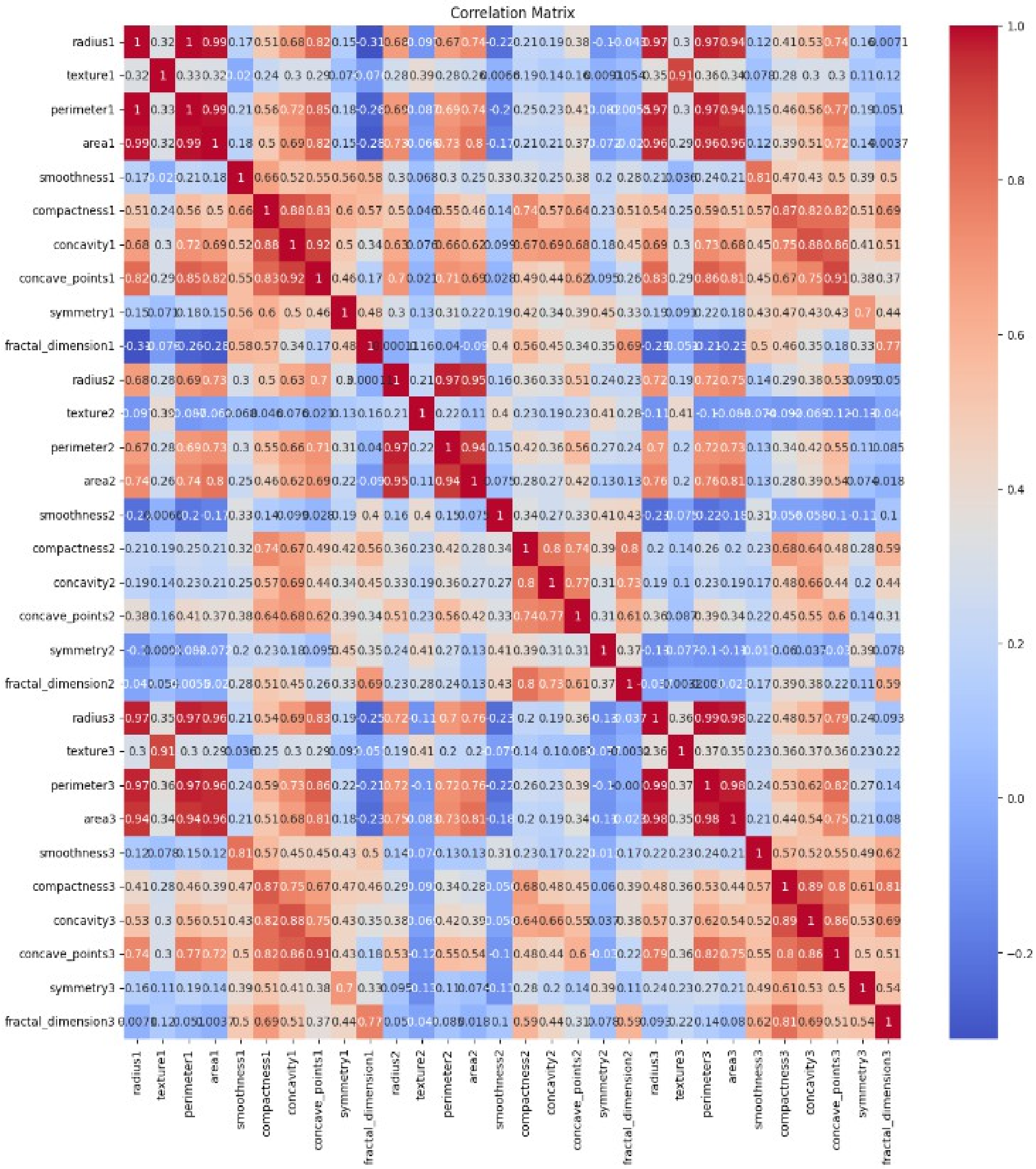
```
In [10]: # chi-square test

categorical_cols = data.select_dtypes(include='object').columns
for col in categorical_cols:
    contingency_table = pd.crosstab(data[col], data['Diagnosis'])
    chi2, p_value, _, _ = chi2_contingency(contingency_table)
    print(f'Chi-square test for column {col}:')
    print(f'Chi2 statistic: {chi2}')
    print(f'P-value: {p_value}')
    print('---')
```

```
Chi-square test for column Diagnosis:
Chi2 statistic: 564.7302404341926
P-value: 7.86394182828703e-125
---
```

The T-tests determine how much of a significant difference there are between B & M. The Chi-square tests assess the independence between the features and how they relate to the target value.

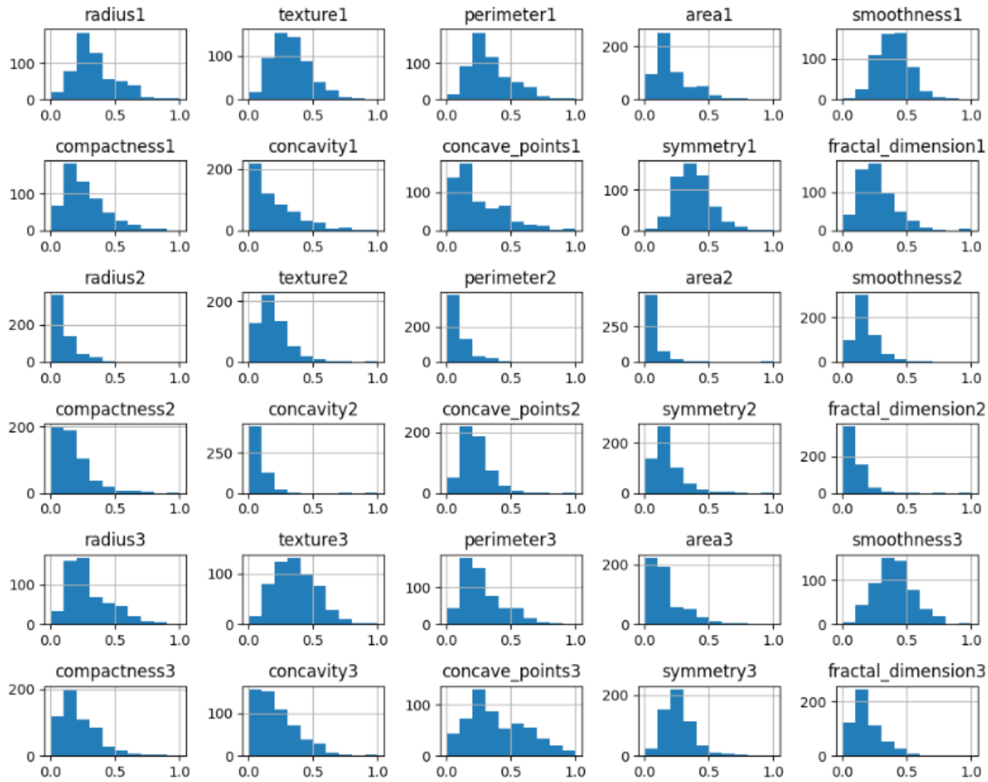
Above we can see that the difference is very significant in the "radius1" and "perimiter1" fields, and less significant in other fields.

```
In [7]: # Create a Heatmap of the Correlation Matr

correlation_matrix = data.corr(numeric_only = True)
plt.figure(figsize=(15, 15))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

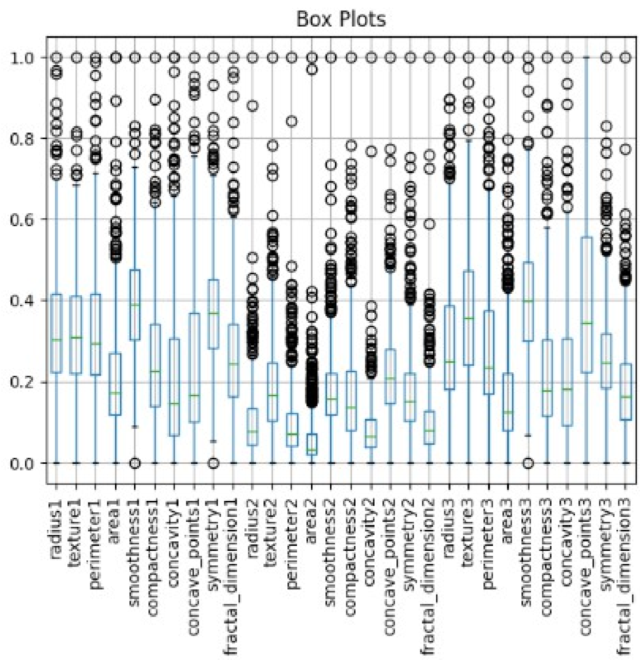


I also created a Heatmap that shows a correlation matrix. Here we can see how similiar certain features are. Radius3 and Radius1 for example are very similiar, while smoothness2 and fractal_dimension3 are very different. Later in the feature selection stage we will see less values that are similar to each other appearing less.

In [6]:
```python
# Visualize the distribution of float values

data.drop('Diagnosis', axis=1).hist(figsize=(10, 8))
plt.tight_layout()
plt.show()
```



In [8]:
```python
# Generate box plots

data.drop('Diagnosis', axis=1).boxplot()
plt.title('Box Plots')
plt.xticks(rotation=90)
plt.show()
```



The box plot and distribution plot provide a visual representation of the distribution of data.

Chapter 3 : Building ML algorithms

I will present the three basic learning algoirthms I created below.
In all of my models I used a test size of 0.2 and a maxium iteration count of 1000.

When presented with the opportunity to set a random state, I consciously selected the number 42. By doing so, I aimed to achieve improved reproducibility of the algorithm, which is important for consistent results.

Chapter 3.1 : Supervised model

For my first model, I chose to use a logistic regression approach. Logistic Regression is a commonly used and straightforward supervised learning algorithm, especially useful for binary classification tasks like my breast cancer detection algorithm. By training the logistic regression model with the labeled data I had, my aim was to find straightforward connections between the features and the target variable.

In [1]:
```
# Supervised Learning: Logistic Regression
```

In [2]:
```
# Imports

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

In [3]:
```
data = pd.read_csv('wdbc.csv')

X = data.iloc[:, 1:]
y = data.iloc[:, 0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           B       0.97      1.00      0.99        71
           M       1.00      0.95      0.98        43

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

The supervised model performed exceptionally well, achieving an immediate impressive accuracy rating of 0.98. This outcome demonstrates the effectiveness and efficiency of the model in accurately predicting the target variable based on the labeled data.

Chapter 3.2 : Unsupervised model

For my subsequent model, I opted for an unsupervised K-means clustering approach. Typically, this method is used when dealing with missing data or in scenarios where labeled information is unavailable. However, in this case, I chose to compare its outcomes to those of the first model.

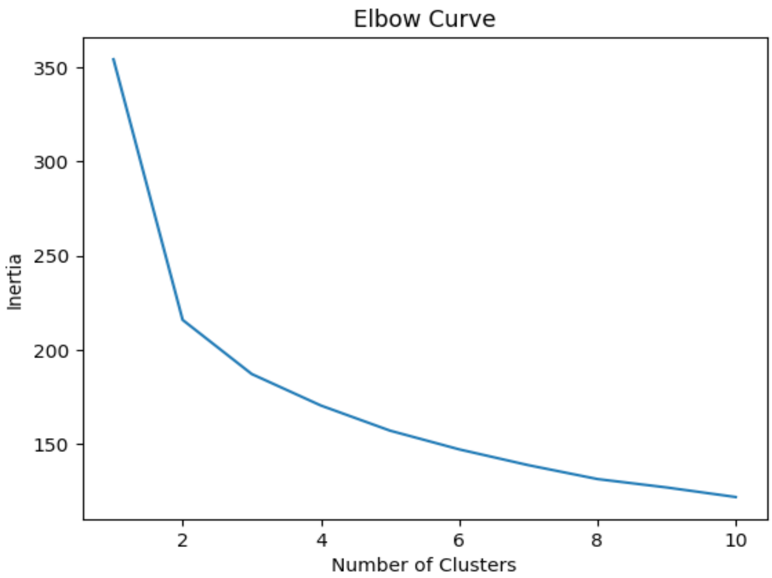
While it is not be directly possible to directly compare the outputs of the unsupervised K-means clustering approach with the supervised model, I still can gain insights by combining these models with feature selection techniques and comparing their behaviour. By incorporating feature selection, we can assess the impact on performance and potentially gain a better understanding of the strengths and weaknesses of each model in relation to the dataset at hand.

Chapter 6 will provide an in-depth examination of feature selection techniques.

```
In [1]: # Unsupervised Learning: K-means Clustering
```

```
In [2]: import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans

        data = pd.read_csv('wdbc.csv')

        X = data.iloc[:, 1:]

        inertia = []
        for k in range(1, 11):
            kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
            kmeans.fit(X)
            inertia.append(kmeans.inertia_)

        plt.plot(range(1, 11), inertia)
        plt.title('Elbow Curve')
        plt.xlabel('Number of Clusters')
        plt.ylabel('Inertia')
        plt.show()

        kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
        kmeans.fit(X)

        data['Cluster'] = kmeans.labels_

        print(data[['Diagnosis', 'Cluster']])
```

When we visualize the elbow graph we can see the Intertia decrease for each additinal cluster. This indicates that the clusters become more tightly packed, resulting in reduced distances between the data points and their respective cluster centers. Understanding the graph helps us create a balance between using meaningful structure within the data and avoiding unnecessary complexity.




```
    Diagnosis  Cluster
0           M        1
1           M        1
2           M        1
3           M        2
4           M        1
..        ...      ...
564         M        1
565         M        1
566         M        1
567         M        1
568         B        0

[569 rows x 2 columns]
```

The above picture of the elbow graph illustrates the impressive performance of the model.

Chapter 3.3 : Semi-supervised model

For my third learning model, I decided to use a Semi-Supervised label propagation model.

I first loaded and prepared the data. The 'X' variable contains the features of the dataset (all columns except the first one), while 'y' contains the corresponding labels (the first column).
The 'train_test_split' function is used to split the data into labeled and unlabeled portions. 'X_labeled' and 'y_labeled' represent the labeled data, while 'X_unlabeled' and 'y_unlabeled' represent the unlabeled data. Here, 80% of the data is used for the unlabeled portion. I then created a LabelPropagation model. The model was then trained on the labeled data and produced the classification report below.

```
In [1]: #  Semi-Supervised Learning : Label Propagation
```

```
In [2]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.semi_supervised import LabelPropagation
from sklearn.metrics import classification_report

data = pd.read_csv('wdbc.csv')

X = data.iloc[:, 1:]
y = data.iloc[:, 0]

X_labeled, X_unlabeled, y_labeled, y_unlabeled = train_test_split(X, y, test_size=0.8, random_state=42)

model = LabelPropagation()
model.fit(X_labeled, y_labeled)

y_pred = model.predict(X_unlabeled)

print(classification_report(y_unlabeled, y_pred))
```

```
              precision    recall  f1-score   support

           B       0.97      0.98      0.98       290
           M       0.96      0.95      0.96       166

    accuracy                           0.97       456
   macro avg       0.97      0.97      0.97       456
weighted avg       0.97      0.97      0.97       456
```

Here we can see a total accuraccy of 0.97. The results are almost as good as in the first algorithm.

Chapter 4 : Constructing and Selecting Features

After creating the three machine learning models, I explored what effect different feature selection methods had on these machine learning algorithms.

I choose five different feature selection algorithms : SelectKBest_chi2, SelectKBest_f_classif, SelectKBest_mutual_info_classif, SelectFromModel_RandomForest and SelectFromModel_LinearSVC. I then compared them with each of the models.

I then created five variations of each of the feature selection algorithms, with a different number of features to select : 5, 10, 15, 20 and 25.

adding the three models from chapter five (which do not use feature selection), this resultes in 78 (5 * 3 * 5 + 3) differently trained models.

Doing this helped compare the models, the feature selection algorithms and their parameters.

To achive this, I used the original post-processed "wdbc.csv"-file and created the required 25 new CSV files uisng my "Compare Feature Selection" script below. These 25 new csv files would then be used by each model.

Saving the CSV files has proven to be highly advantageous in terms of optimizing both RAM and CPU usage. By loading the files individually as needed, I have successfully reduced the strain on the system's memory. Additionally, this approach eliminates the need to rerun feature selection algorithms multiple times with identical parameters, resulting in significant CPU savings.

```
In [33]: import pandas as pd
from sklearn.feature_selection import SelectKBest, chi2, f_classif, mutual_info_classif, SelectFromModel
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC

# LOAD CSV
data = pd.read_csv('wdbc.csv')

X = data.iloc[:, 1:]
y = data.iloc[:, 0]

# FEATURE COUNT
ks = [5, 10, 15, 20, 25]

for k in ks:

    # LIST OF METHODS
    methods = [
        ('SelectKBest_chi2', SelectKBest(chi2, k=k)),
        ('SelectKBest_f_classif', SelectKBest(f_classif, k=k)),
        ('SelectKBest_mutual_info_classif', SelectKBest(mutual_info_classif, k=k)),
        ('SelectFromModel_RandomForest', SelectFromModel(RandomForestClassifier(), max_features=k)),
        ('SelectFromModel_LinearSVC', SelectFromModel(LinearSVC(), max_features=k)),
    ]

    # APPLY METHOD
    for name, selector in methods:

        X_selected = selector.fit_transform(X, y)
        selected_indices = selector.get_support(indices=True)
        selected_features = X.columns[selected_indices]

        # VIEW SELECTED FEATURES

        print(f"Selected Features ({k}) for {name}:")
        fts = ""
        for feature in selected_features:
            if len(fts) > 0:
                fts += ", "
            fts += (feature)
        print(fts)
        print()

        selected_data = pd.DataFrame(X_selected, columns=selected_features)
        selected_data_with_target = pd.concat([selected_data, y], axis=1)

        # SAVE FILE

        path = f'wdbc_{name}_{k}_.csv'
        selected_data_with_target.to_csv(path, index=False)

        df = pd.read_csv(path)
        last_column = df.iloc[:, -1]
        df = df.iloc[:, :-1]
        df.insert(0, 'Diagnosis', last_column)
        df.to_csv(path, index=False)
```

All created CSV Files can be viewed in the Github repo under the naming convention "wdbc_{x}_{y}_.csv'" where "x" represents the name of the feature selection algorithm and "y" represents the number of selected features.

"Methods" contains all the feature selection algorithms and "ks" contains all the amounts of features to be selected. This can be edited to create even more variantions.

The features selected by each algorithm are shown in the "Selected Features.txt" file located in the coursework repository.

Chapter 5 : Evaluating models and analysing the results

At this point I created the "All algorithms - full comparison with feature selection.ipynb" script to run the final above described test.

First I prepared the learning algorithms from Chapter 5 and made them easily accessible in my final evaluation algorithm below. Here I tested both the perfomance of the models with different feature selection settings.

In [1]:
```
def check_accuracy_supervisedLearning(path):

    # ALGO A

    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import classification_report
    from sklearn.metrics import precision_score

    data = pd.read_csv(path)

    X = data.iloc[:, 1:]
    y = data.iloc[:, 0]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    precision = precision_score(y_test, y_pred, average='weighted')

    return precision
```

In [2]:
```
def check_accuracy_unsupervisedLearning(path):

    # ALGO B

    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.cluster import KMeans
    from sklearn.metrics import silhouette_score

    data = pd.read_csv(path)

    X = data.iloc[:, 1:]

    silhouette_scores = []

    for k in range(2, 11):
        kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
        kmeans.fit(X)
        labels = kmeans.labels_
        silhouette_scores.append(silhouette_score(X, labels))

    best_k = silhouette_scores.index(max(silhouette_scores)) + 2

    kmeans = KMeans(n_clusters=best_k, n_init=10, random_state=42)
    kmeans.fit(X)
    labels = kmeans.labels_

    precision = silhouette_score(X, labels)
    return precision
```

```python
def check_accuracy_SemiSupervisedLearning(path):

    # ALGO C

    from sklearn.metrics import classification_report, precision_score
    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn import datasets
    from sklearn.semi_supervised import LabelPropagation
    from sklearn.metrics import classification_report

    data = pd.read_csv(path)

    X = data.iloc[:, 1:]
    y = data.iloc[:, 0]

    X_labeled, X_unlabeled, y_labeled, y_unlabeled = train_test_split(X, y, test_size=0.2, random_state=42)

    model = LabelPropagation()
    model.fit(X_labeled, y_labeled)

    y_pred = model.predict(X_unlabeled)

    precision = precision_score(y_unlabeled, y_pred, average='macro')

    return precision
```

I then ran the following code to check each algorithm, with each feature selection CSV and each their selected features :

In [5]:

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


FeatureSelectionPath = [

    "wdbc_SelectFromModel_LinearSVC_",
    "wdbc_SelectFromModel_RandomForest_",
    "wdbc_SelectKBest_chi2_",
    "wdbc_SelectKBest_f_classif_",
    "wdbc_SelectKBest_mutual_info_classif_"

]

FeatureSelectionLabels = [
    "SVC",
    "Forest",
    "KB-Chi2",
    "KB-Clsf",
    "KB-MI-Clsf",
]

FeatureSelectionCount = [5, 10, 15, 20, 25]

for i in range(0, 3):

    df = pd.DataFrame(0, index=FeatureSelectionCount, columns=FeatureSelectionLabels)

    indexlabel = 0

    for FeatureSelectionLabel in FeatureSelectionLabels:
        for FeatureSelectionCountCurrent in FeatureSelectionCount:

            path = FeatureSelectionPath[indexlabel] + str(FeatureSelectionCountCurrent) + "_.csv"

            val = 0;

            if i == 0:
                val = check_accuracy_supervisedLearning(path)
            if i == 1:
                val = check_accuracy_unsupervisedLearning(path)
            if i == 2:
                val = check_accuracy_SemiSupervisedLearning(path)

            df.loc[FeatureSelectionCountCurrent, FeatureSelectionLabel] = round(val, 4)

        indexlabel += 1

    fig, ax = plt.subplots(figsize=(8, 2))

    table = ax.table(cellText=df.values,
                     colLabels=df.columns,
                     rowLabels=df.index,
                     cellLoc='center',
                     loc='center',
                     cellColours=plt.cm.Greens(np.zeros_like(df.values))
                     )

    table.auto_set_font_size(False)
    table.set_fontsize(12)
    table.scale(1.2, 1.2)

    ax.axis('off')

    ax.set_title(["Supervised learning", "Unsupervised learning", "Semi-supervised learning"][i])
    plt.show()
```

Running the script produced the following tables :

Supervised learning

	SVC	Forest	KB-Chi2	KB-Clsf	KB-MI-Clsf
5	0.9569	0.9475	0.9475	0.9475	0.9475
10	0.9668	0.9569	0.9569	0.9569	0.9668
15	0.9652	0.9668	0.9569	0.9569	0.9569
20	0.9652	0.9668	0.9748	0.9748	0.9652
25	0.9652	0.9569	0.9748	0.9748	0.9748

Unsupervised learning

	SVC	Forest	KB-Chi2	KB-Clsf	KB-MI-Clsf
5	0.5014	0.5883	0.5825	0.5761	0.5883
10	0.4923	0.5742	0.5571	0.5571	0.5706
15	0.4402	0.5718	0.5293	0.5293	0.5293
20	0.4402	0.5718	0.4529	0.4529	0.4643
25	0.4402	0.5571	0.4186	0.4186	0.4186

Semi-supervised learning

	SVC	Forest	KB-Chi2	KB-Clsf	KB-MI-Clsf
5	0.9733	0.9538	0.9538	0.9538	0.9538
10	0.9671	0.9605	0.9605	0.9605	0.9605
15	0.9605	0.9605	0.9605	0.9605	0.9605
20	0.9605	0.9605	0.9554	0.9554	0.9673
25	0.9605	0.9605	0.9554	0.9554	0.9554

These tables display the output of each of the learning algorithms, with both the feature selection (X-Axis) and the number of selected features (Y-Axis).

In the supervised and semi-supervised tables I displayed the precision scores, while the unsupervised learning table is based on the silhouette score.

viewing the tables and comparing them with the outputs given in chapter 5 (the outputs without any feature selection), we can clearly see that the supervised learning model, without any feature selection works best:

	precision	recall	f1-score	support
B	0.97	1.00	0.99	71
M	1.00	0.95	0.98	43
accuracy			0.98	114
macro avg	0.99	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

This was expected because the unsupervised and semi-supervised models tend to work better than the supervised model in cases where there is a lack of data (for example: lack of labels or noise). In this case, the dataset I worked with had plenty of data, good features, and was well-prepared.

When looking at the feature selection, it's interesting to see how well the model performs with just five features instead of using the full 30 features.

In the case of the SVC feature selection, the supervised model continues to have an impressive accuracy of 0.95. This minor decrease in accuracy compared to the full model demonstrates the efficiency of the feature selection process. Despite a significant reduction in the number of features, the model maintains a high level of performance, displaying the importance of selecting the most influential features for optimal results.

In larger datasets aiming to address this problem, using feature selection will reduce complexity and training time. SVC works well when using a small number of features, but if more than 15 features are used, any of the KB-* feature selection systems work better.

Unfortunately, the RandomForest classifier doesn't perform well enough with a small number of features to beat SVC, and it doesn't work well enough with a large number of features to beat any of the KB-* algorithms. It even seems to get worse after 20 features, both in supervised and unsupervised learning.

Chapter 6 : Conclusion

The course and this task allowed me to understand the power that is machine learning. It is incredible to see how fast algorithms can create a deep understanding of raw data, and apply it to solve problems like these.

Im certain machine learning will become one of the most prevalent tools in medicine. The potential of this technology to save lives is incredible.

Chapter 7 : Sources

1. *https://www.krebsdaten.de/Krebs/EN/Content/Cancer_sites/Breast_cancer/breast_cancer_node.html*

2. *https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic*

3. *https://github.com/University-of-London/csm010-aml-coursework-KaiBowers99*