Breast Cancer Classification using Machine Learning Methods

1. Introduction

Tumors are masses of mutated and dysfunctional cells that may cause pain and disfigurement, invade organs and, potentially, spread throughout the body. However, not all tumors are aggressive or malignant (cancerous). Malignant tumors represent a greater threat than benign tumors. They are more likely to spread, whereas benign tumors often do not invade and remain localized. When someone is diagnosed with breast cancer, usually the healthcare provider would first have to determine if it is benign or malignant.

In this project, we aim to provide a supervised machine learning model that aid the classification of breast cancer tumors based on various features of breast mass lesions provided in the Breast Cancer Wisconsin (Diagnostic) Data Set [1] available from Kaggle. First, we begin by formulating the problem. Following this, we introduce two machine learning methods to address the issue. Finally, we compare and analyze the results to determine the optimal choice between the two methods. Reader can find the project code in the appendix section of this report.

2. Problem formulation

The dataset provides 30 features in total, including radius, texture, perimeter, area, smoothness, compactness, concavity, symmetry, and fractal dimension, computed from each cell nucleus. For each feature, these values are provided: the mean, standard deviation, and worst (biggest) value. As a result, 10 traits are measured in triplicate, giving rise to a total of 30 features. The features are continuous numerical values computed from a digitized image of a fine needle aspirate (FNA) of a breast mass, describing characteristics of the cell nuclei present in the image. The dataset was originally collected and compiled by Dr. William H. Wolberg at the University of Wisconsin Hospitals in Madison, USA.

The structure of the dataset includes 569 instances of breast masses, with no missing value and each of which represents a breast mass lesion. Out of all includes:

- 212 are labeled as malignant (1), indicating cancerous tumors.
- 357 are labeled as benign (0), indicating non-cancerous tumors.

3. Methods

3.1. Data preparation and feature selection

There are a total of 569 data points, each with 30 traits, 26 of which are chosen as features after feature selection and feature engineering process, and 1 label. As there are numerous variables and we do not have any medical background to determine the most important ones, features are selected using Principal Component Analysis (PCA) to reduce the dimension of our feature space. Although this will slightly reduce the accuracy of the model, PCA increases simplicity significantly, helps eliminating irrelevant features, extracting important features and relationships between them and as a result, avoid overfitting.

First, we transform diagnosis into binary value 0 (benign) and 1 (malignant) and use StandardScaler to transform other features to ensure that all features have a mean of 0 and and standard deviation of 1, which is important to perform PCA. As PCA involves finding eigenvectors and eigenvalues, it can be influenced by the scale of features. Therefore, we apply standardization to ensure that all features are treated equally in the process. A heatmap of the correlation matrix for the columns in the dataset is then generated, which allows us to quickly visualize the strength and direction of relationships between pairs of columns and understand the data's patterns. In addition, libraries including Pandas and Scikit-learn are used for data preparation, data splitting and feature engineering.

Based on the result of the heat map and PCA, four features ‘symmetry_se', 'fractal_dimension_mean', 'texture_se', 'smoothness_se' are not used as features as there is no correlation with the label we are trying to predict. Apart from these four, other features have strong positive correlation with the label.

3.2. Logistic Regression

Given the binary classification nature of our problem, Logistic Regression stands out as a natural and efficient choice. It effectively capitalizes on the high positive correlation between the features and labels, making it a robust selection. Another notable advantage of Logistic Regression is its regularization capabilities. In this project, we will utilize L1 regularization to force potentially redundant features to a coefficient of zero, thereby addressing the issue of overfitting.

In this project we will employ logistic loss to train our model. Logistics loss works naturally with binary classification tasks and is readily available in the scikit-learn library, making it easy to use in our project.

3.3. Decision Trees

Decision tree is chosen for its capability to capture non-linear relationships between features that were not captured by Logistic Regression. Moreover, by setting appropriate splitting criteria and tree depth, we can emphasize the reduction of false positives, which is especially important in the healthcare sector.

For the model’s loss function, decision trees use impurity to guide their splitting and decision-making process. One impurity measure that goes naturally with decision tree classifier is entropy. Entropy measures impurity, in the context of decision trees and their loss functions means disorder or uncertainty corresponds to a group of data points within a node of the tree. A node with low impurity means that the data points it contains are predominantly from a single class, making classification decisions straightforward. Measuring entropy can help to minimize impurity and maximize information gain. These are especially suitable for our particular data set where the label is relatively balanced.

3.4. Validation

We split the data into training and test sets using scikit-learn package. It splits the features (**x**) and the target (**y**) into training and test sets, with 80% of the data used for training (**X_train** and **y_train**) and 20% for testing (**X_test** and **y_test**). PCA is then applied to the training data (**X_train**) to reduce the dimensionality of the feature space.

Given the ratio of feature and data point, we test and train data with different sessions using k-fold cross validation. This extension of the single split into training set and validation set effectively addressed the problem of limited dataset and the risk of outliers having more impact on the result.

The dataset is first divided evenly into k = 5 subsets. Then, the loop of 5 repetitions of training and validation are executed with each repetition uses a different fold as the validation set and the remaining folds as a training set. Precision and accuracy values for our two different models Decision Tree and Logistic Regression are accumulated and updated during cross-validation. The final output is the average of the validation errors obtained from the 5 repetitions.

4. Results

The table below presents average training and validation accuracy and average training and validation precision for each model with and without PCA (rounded to four decimals). Accuracy refers to how often the model is correct overall. Precision refers to how often the model is correct when predicting the target class, i.e. benign or malignant.

		Average Training Accuracy	Average Training Precision	Average Validation Accuracy	Average Validation Precision	Test Accuracy	Test Precision
With PCA	Decision Tree Classifier	0.9934	0.9925	0.9385	0.9238	0.9123	0.8837
	Logistic Regression	0.9769	0.9733	0.9626	0.9482	0.9825	0.9767
Without PCA	Decision Tree Classifier	0.9973	1.0	0.9253	0.9020	0.9386	0.9091
	Logistic Regression	0.9846	0.9853	0.9670	0.9760	0.9737	0.9545

Decision Tree model achieved slightly higher training accuracy and precision, indicating that it is capable of fitting the training data almost perfectly. However, it performs much worse with test and validation, which indicates that there is overfitting. In addition, the test precision of Decision Tree is the lowest, indicating that the model produces false positive predictions, which is especially dangerous in medical sector as it is stressful, expensive, and dangerous to patients and put burden on healthcare facilities.

The higher precision suggests that Logistic Regression is better at making accurate positive predictions. It also has a more consistent performance, meaning that it can generalize to unseen data better than Decision Tree. Therefore, the final chosen method is Logistic Regression with test error of 2.63% (1 - Test Accuracy), as a linear hypothesis is apparently better for this particular application and dataset.

5. Conclusion

The report reflects the application and result of two machine learning model Logistic Regression and Decision Tree, with Principal Component Analysis (PCA) transformation and k-fold cross-validation, on the problem of breast cancer tumors classification based on features of breast mass lesions provided in the Breast Cancer Wisconsin (Diagnostic) Data Set [1].

With and without PCA, both models performed well in terms of accuracy and precision on the validation set. However, Logistic Regression consistently outperforms Decision Tree. It is worth-noticing here that PCA slightly decreases the performance of the models. Although PCA would avoid overfitting and greatly improve simplicity, this might not be the best for such a limited dataset, where information loss would significantly impact the result.

The final method, Logistic Regression shows a strong generalization and consistent performance with test error of 2.63%. This is a very good model, however we believe there is room for improvement.

As small dataset is considered in the very first place, many methods and implementations like setting the maximum depth of Decision Tree or k-fold cross-validation were applied to avoid overfitting but it still remains an issue. Decision Trees are prone to overfitting when they have too few samples at a particular node to make reliable split decisions. Having more knowledge about breast cancer might increase the quality of feature selection, hence, avoid overfitting. Another method to be considered is an ensemble of decision trees - Random Forest, which would combine multiple decision trees and improve generalization.

6. References

[1]
https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data?fbclid=IwAR0NYASKERlw8rfpKibTvnJTIlgEazDDzJ4DGmGenOcCcX4q3p4Ego6AXFU

[2]
https://towardsdatascience.com/decision-tree-part-2-34b31b1dc328

[3] Pandas package for Python: https://pandas.pydata.org/pandas-docs/stable/index.html

[4] Scikit-learn package for Python: https://scikit-learn.org/stable/index.html

[5] A. Jung, "Machine Learning: The Basics," Springer, Singapore, 2022
https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf

[6] https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall

7. Appendices

appendices

October 11, 2023

```
[1]: %config Completer.use_jedi = False  # enable code auto-completion
     import warnings

     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns  #data visualization library
     from sklearn.metrics import accuracy_score, confusion_matrix ,precision_score #␣
       ↪evaluation metrics

     from sklearn.tree import export_text, plot_tree, DecisionTreeClassifier
     from sklearn.linear_model import LogisticRegression

     from sklearn.datasets import fetch_openml
     from sklearn.model_selection import train_test_split
     from sklearn.model_selection import KFold
     from sklearn.preprocessing import StandardScaler
     from sklearn.decomposition import PCA
```

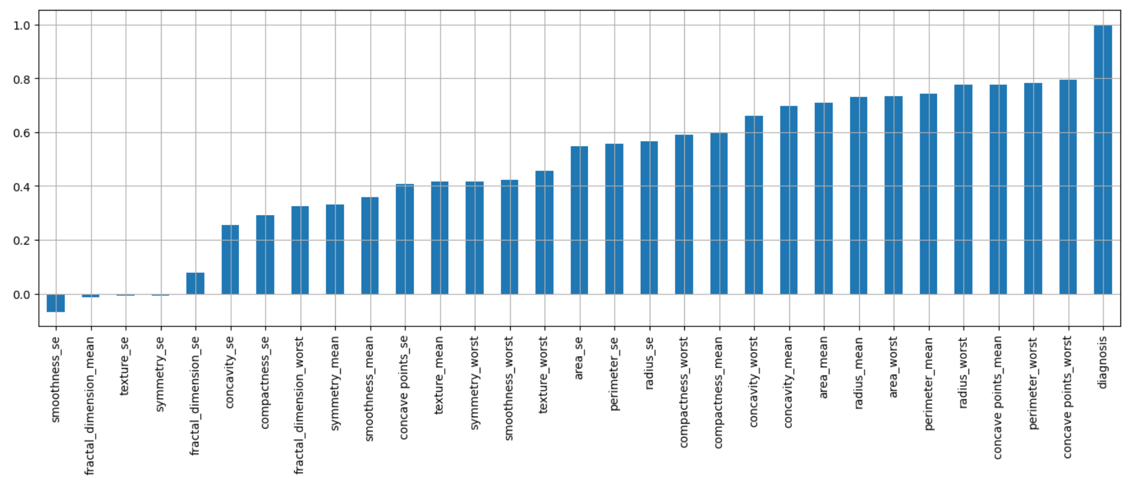
0.1 Processing The Data Set

```
[2]: ## read the data set
     df = pd.read_csv("breast-cancer.csv")

     ## drop id
     df.drop(columns = ['id'] , inplace = True )

     ## transform diagnosis into binary value
     df.replace({'diagnosis': {'B': 0, 'M': 1}}, inplace=True)
     diagnosis = df['diagnosis']
     ## We use StandardScaler to transform the feature
     scaler = StandardScaler()
     df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

     #set diagnosis back to binary
     df['diagnosis'] = diagnosis
     df.head()
```

[2]:
```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1     1.097064     -2.073335        1.269934   0.984375
1          1     1.829821     -0.353632        1.685955   1.908708
2          1     1.579888      0.456187        1.566503   1.558884
3          1    -0.768909      0.253732       -0.592687  -0.764464
4          1     1.750297     -1.151816        1.776573   1.826229

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0         1.568466          3.283515        2.652874             2.532475
1        -0.826962         -0.487072       -0.023846             0.548144
2         0.942210          1.052926        1.363478             2.037231
3         3.283553          3.402909        1.915897             1.451707
4         0.280372          0.539340        1.371011             1.428493

   symmetry_mean  …  radius_worst  texture_worst  perimeter_worst  \
0       2.217515  …      1.886690      -1.359293         2.303601
1       0.001392  …      1.805927      -0.369203         1.535126
2       0.939685  …      1.511870      -0.023974         1.347475
3       2.867383  …     -0.281464       0.133984        -0.249939
4      -0.009560  …      1.298575      -1.466770         1.338539

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0    2.001237          1.307686           2.616665         2.109526
1    1.890489         -0.375612          -0.430444        -0.146749
2    1.456285          0.527407           1.082932         0.854974
3   -0.550021          3.394275           3.893397         1.989588
4    1.220724          0.220556          -0.313395         0.613179

   concave points_worst  symmetry_worst  fractal_dimension_worst
0              2.296076        2.750622                 1.937015
1              1.087084       -0.243890                 0.281190
2              1.955000        1.152255                 0.201391
3              2.175786        6.046041                 4.935010
4              0.729259       -0.868353                -0.397100

[5 rows x 31 columns]
```

0.2 Visualize the correlation between features and label:

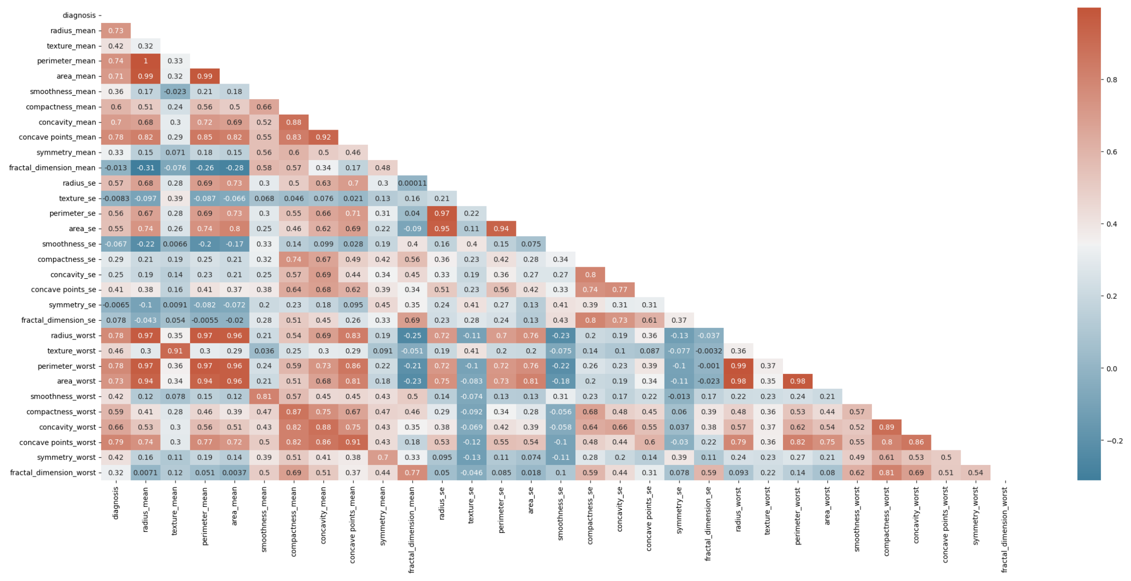
[3]:
```
plt.figure(figsize=(17,5))
df.corr()["diagnosis"].sort_values(ascending=True).plot(kind="bar")
plt.grid()
plt.show()
```



0.3 Visualize the correlation betweens features with heat map:

[4]:
```python
corr = df.corr()
f, ax = plt.subplots(figsize=(29, 12))
mask = np.triu(np.ones_like(corr, dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr, annot=True, mask = mask, cmap=cmap)
```

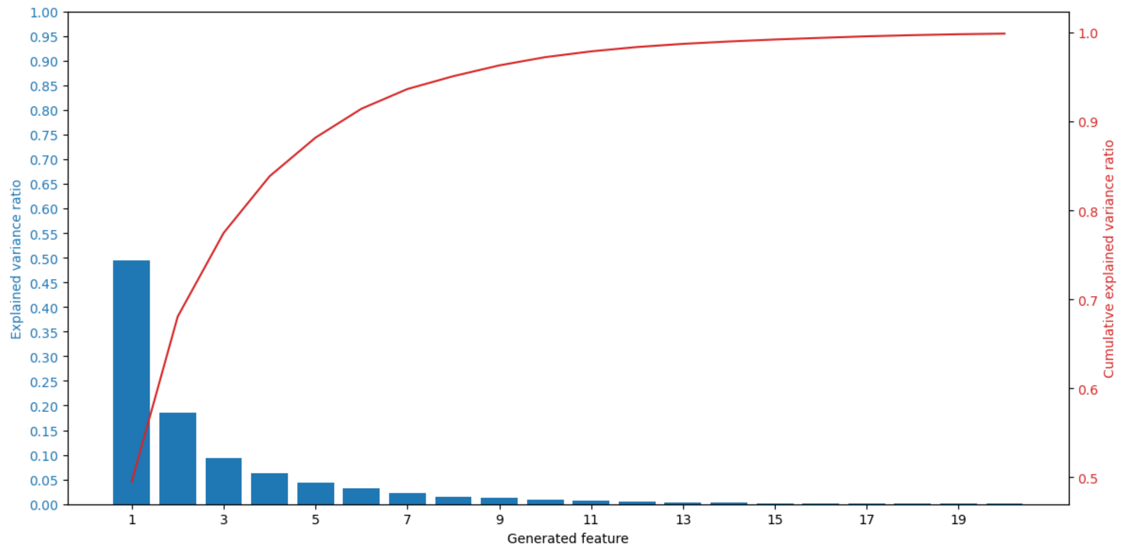
[4]:
```
<Axes: >
```



0.4 Feature engineering with PCA:

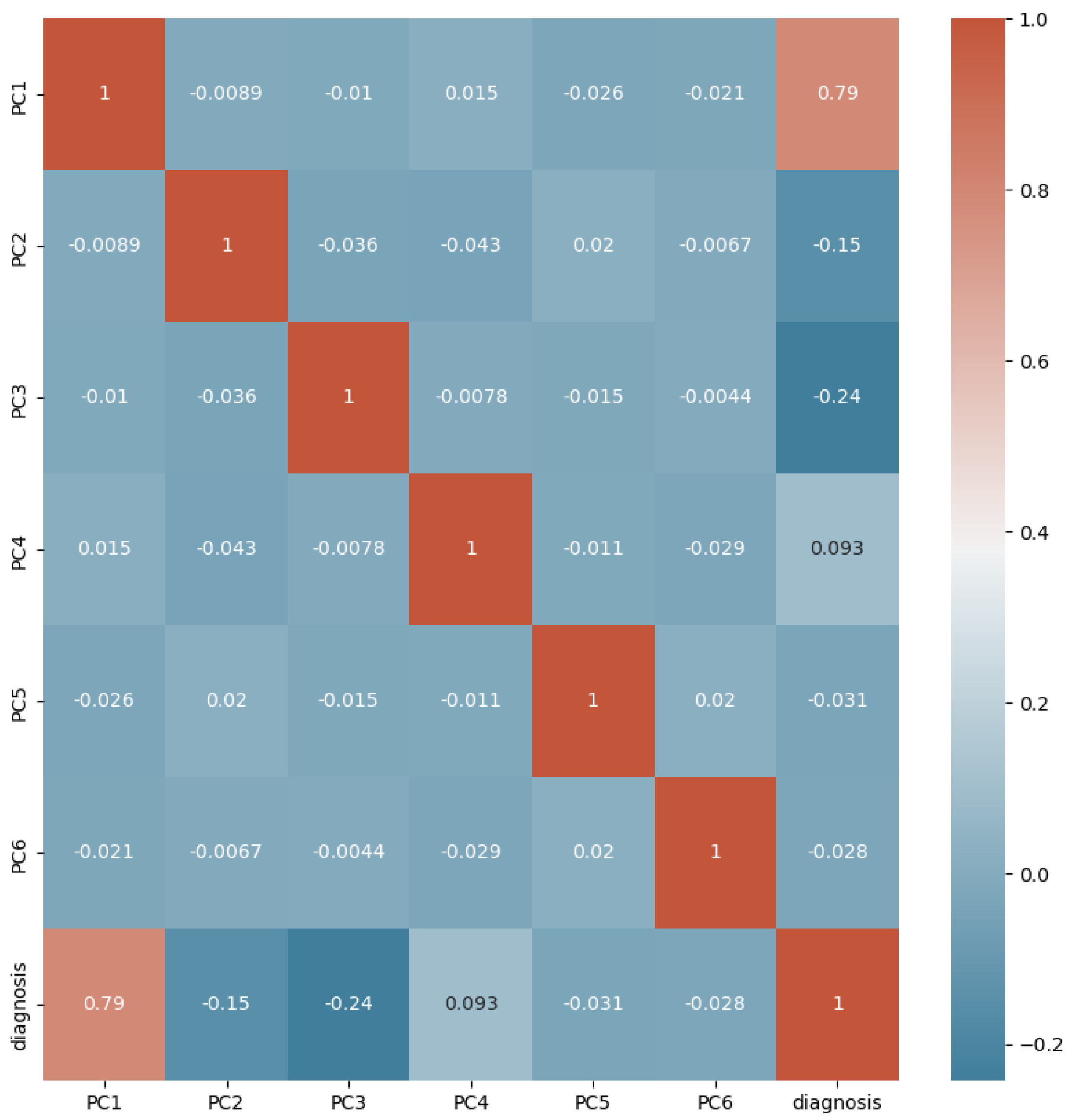
we start by plotting Cumulative and Explained variance ratio with number of generated feature.

[5]:
```
x = df.drop(columns =␣
↪['diagnosis','symmetry_se','fractal_dimension_mean','texture_se','smoothness_se']␣
↪)
y = df['diagnosis']
X_train_val, X_test, y_train_val, y_test = train_test_split(x, y, test_size=0.
↪2, random_state=42)

N = 20
pca = PCA(n_components=N)
X_train_reduced = pca.fit_transform(X_train_val)

fig, ax1 = plt.subplots(figsize=(12, 6))
color = 'tab:blue'
ax1.bar(1+np.arange(N), pca.explained_variance_ratio_, color=color)
ax1.set_xticks(1+np.arange(N, step=2))
ax1.tick_params(axis='y', labelcolor=color)
ax1.set_ylabel("Explained variance ratio", color=color)
ax1.set_xlabel("Generated feature")

ax1.set_yticks(np.arange(0, 1.05, 0.05))
ax2 = ax1.twinx()
color = 'tab:red'
ax2.tick_params(axis='y', labelcolor=color)
ax2.plot(1+np.arange(N), np.cumsum(pca.explained_variance_ratio_), color=color)
ax2.set_ylabel("Cumulative explained variance ratio", color=color)
fig.tight_layout()
plt.show()
```



[6]:
```
N = 6
pca.set_params(n_components = N)
X_train_reduced = pca.fit_transform(X_train_val)


newdf = pd.DataFrame(pca.transform(x), columns=[f'PC{i}' for i in range(1,␣
 ↪N+1)])
newdf['diagnosis'] = y
corr = newdf.corr()
f, ax = plt.subplots(figsize=(10, 10))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr, annot=True,  cmap=cmap)
```

[6]:
```
<Axes: >
```



[7]:
```
##Define k fold
k = 5

seed = 3

kfold = KFold(n_splits=k, shuffle=True, random_state=seed)
```

[8]:
```
# Initialize variables
average_train_precision_tree_pca = 0
average_train_accuracy_tree_pca = 0
average_val_precision_tree_pca = 0
```

```
average_val_accuracy_tree_pca = 0

average_train_precision_Logistic_pca = 0
average_train_accuracy_Logistic_pca = 0
average_val_precision_Logistic_pca = 0
average_val_accuracy_Logistic_pca = 0

# Loop through all folds
for train_index, val_index in kfold.split(y_train_val):
    X_train, y_train = X_train_reduced[train_index], y_train_val.
 ↪iloc[train_index]
    X_val, y_val = X_train_reduced[val_index], y_train_val.iloc[val_index]

    # Decision Tree Classifier
    clf_1 = DecisionTreeClassifier(criterion="entropy", max_depth=5,␣
 ↪random_state=42)
    clf_1.fit(X_train, y_train)
    y_pred_val_1 = clf_1.predict(X_val)

    average_train_accuracy_tree_pca += accuracy_score(y_train, clf_1.
 ↪predict(X_train))
    average_train_precision_tree_pca += precision_score(y_train, clf_1.
 ↪predict(X_train))

    average_val_precision_tree_pca += precision_score(y_val, y_pred_val_1)
    average_val_accuracy_tree_pca += accuracy_score(y_val, y_pred_val_1)

    # Logistic Regression
    clf_2 = LogisticRegression()
    clf_2.fit(X_train, y_train)
    y_pred_val_2 = clf_2.predict(X_val)

    average_train_accuracy_Logistic_pca += accuracy_score(y_train, clf_2.
 ↪predict(X_train))
    average_train_precision_Logistic_pca += precision_score(y_train, clf_2.
 ↪predict(X_train))

    average_val_precision_Logistic_pca += precision_score(y_val, y_pred_val_2)
    average_val_accuracy_Logistic_pca += accuracy_score(y_val, y_pred_val_2)

# Calculate and print results
print("Decision Tree Classifier with PCA:")
print("Average Training Accuracy:", average_train_accuracy_tree_pca / k)
print("Average Training Precision:", average_train_precision_tree_pca / k)
print("Average Validation Accuracy:", average_val_accuracy_tree_pca / k)
print("Average Validation Precision:", average_val_precision_tree_pca / k)
```

```
y_pred_test_tree = clf_1.predict(pca.transform(X_test))
print("Test Accuracy:", accuracy_score(y_test, y_pred_test_tree))
print("Test Precision:", precision_score(y_test, y_pred_test_tree))
print()

print("Logistic Regression with PCA:")
print("Average Training Accuracy:", average_train_accuracy_Logistic_pca / k)
print("Average Training Precision:", average_train_precision_Logistic_pca / k)
print("Average Validation Accuracy:", average_val_accuracy_Logistic_pca / k)
print("Average Validation Precision:", average_val_precision_Logistic_pca / k)

y_pred_test_log = clf_2.predict(pca.transform(X_test))
print("Test Accuracy:", accuracy_score(y_test, y_pred_test_log))
print("Test Precision:", precision_score(y_test, y_pred_test_log))
```

```
Decision Tree Classifier with PCA:
Average Training Accuracy: 0.9934065934065934
Average Training Precision: 0.992514869262709
Average Validation Accuracy: 0.9384615384615385
Average Validation Precision: 0.9237593984962407
Test Accuracy: 0.9122807017543859
Test Precision: 0.8837209302325582

Logistic Regression with PCA:
Average Training Accuracy: 0.976923076923077
Average Training Precision: 0.9733397570290243
Average Validation Accuracy: 0.9626373626373625
Average Validation Precision: 0.9482721956406166
Test Accuracy: 0.9824561403508771
Test Precision: 0.9767441860465116
```

[9]:
```
# Initialize variables
average_train_precision_tree = 0
average_train_accuracy_tree = 0
average_val_precision_tree = 0
average_val_accuracy_tree = 0

average_train_precision_Logistic = 0
average_train_accuracy_Logistic = 0
average_val_precision_Logistic = 0
average_val_accuracy_Logistic = 0

# Loop through all folds
for train_index, val_index in kfold.split(y_train_val):
    X_train, y_train = X_train_val.iloc[train_index], y_train_val.
 ↪iloc[train_index]
```

```
    X_val, y_val = X_train_val.iloc[val_index], y_train_val.iloc[val_index]

    # Decision Tree Classifier
    clf_1 = DecisionTreeClassifier(criterion="entropy", max_depth=5,␣
 ↪random_state=42)
    clf_1.fit(X_train, y_train)
    y_pred_1 = clf_1.predict(X_val)

    average_val_precision_tree += precision_score(y_val, y_pred_1)
    average_val_accuracy_tree += accuracy_score(y_val, y_pred_1)

    average_train_accuracy_tree += accuracy_score(y_train, clf_1.
 ↪predict(X_train))
    average_train_precision_tree += precision_score(y_train, clf_1.
 ↪predict(X_train))

    # Logistic Regression
    clf_2 = LogisticRegression(max_iter=5000, penalty='l1', solver='saga')
    clf_2.fit(X_train, y_train)
    y_pred_2 = clf_2.predict(X_val)

    average_train_accuracy_Logistic += accuracy_score(y_train, clf_2.
 ↪predict(X_train))
    average_train_precision_Logistic += precision_score(y_train, clf_2.
 ↪predict(X_train))

    average_val_precision_Logistic += precision_score(y_val, y_pred_2)
    average_val_accuracy_Logistic += accuracy_score(y_val, y_pred_2)

# Calculate and print results
print("Decision Tree Classifier without PCA:")
print("Average Training Accuracy:", average_train_accuracy_tree / k)
print("Average Training Precision:", average_train_precision_tree / k)
print("Average Validation Accuracy:", average_val_accuracy_tree / k)
print("Average Validation Precision:", average_val_precision_tree / k)

y_pred_test_tree = clf_1.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_pred_test_tree))
print("Test Precision:", precision_score(y_test, y_pred_test_tree))

print()

print("Logistic Regression without PCA:")
print("Average Training Accuracy:", average_train_accuracy_Logistic / k)
print("Average Training Precision:", average_train_precision_Logistic / k)
print("Average Validation Accuracy:", average_val_accuracy_Logistic / k)
print("Average Validation Precision:", average_val_precision_Logistic / k)
```

```
y_pred_test_log = clf_2.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_pred_test_log))
print("Test Precision:", precision_score(y_test, y_pred_test_log))
```

```
Decision Tree Classifier without PCA:
Average Training Accuracy: 0.9972527472527473
Average Training Precision: 1.0
Average Validation Accuracy: 0.9252747252747253
Average Validation Precision: 0.9020448726900341
Test Accuracy: 0.9385964912280702
Test Precision: 0.9090909090909091

Logistic Regression without PCA:
Average Training Accuracy: 0.9846153846153847
Average Training Precision: 0.9852719721146741
Average Validation Accuracy: 0.9670329670329672
Average Validation Precision: 0.9759868421052632
Test Accuracy: 0.9736842105263158
Test Precision: 0.9545454545454546
```

[ ]:

[ ]: