CS-C3240 - Machine Learning D

Project Stage 1 - 19 September 2025

Classifying pancreatic cancer with biomarker data

1 Introduction

Machine learning (ML) approaches are increasingly applied in medical diagnostics for early cancer detection using simple, noninvasive bioindicators. In 2020, Debernardi et al. (2020) investigated a highly accurate diagnostic method for PDAC (pancreatic ductal adenocarcinoma)—the most common type of pancreatic cancer—based on the level of certain proteins in the patient's urine. My project utilizes this data set provided in their publication to develop a supervised multi-class classification ML model that can distinguish an assigned patient into three different diagnostic groups: (1) healthy, (2) having non-cancerous pancreatic diseases, and (3) having PDAC.

This report has five sections: this first introductory section briefly explains the scenario of the application; the second section analyzes the dataset and formulates the problem; the third section focuses on selected methods; the fourth section discusses results; and the final section concludes on overall findings.

2 Problem Formulation

This data set is collected from various clinical centers, and each data point from this set represents a single specimen collected from a specific patient. Among 590 urine specimens, there are 183 healthy samples, 208 non-cancerous (or benign) samples, and 199 PDAC samples (Figure 1). This dataset was directly downloaded from Kaggle (https://www.kaggle.com/datasets/johnjdavisiv/urinary-biomarkers-for-pancreatic-cancer). The features that will be examined are the patients' age and the quantitative measurements of six specific proteins: plasma CA19-9, creatinine, LYVE1, REG1B, TFF1, and REG1A (Appendix Table 1). These biomarkers are scientifically considered relevant for the detection of pancreatic cancer (Debernardi et al., 2020). In addition, Figure 1 also demonstrates relatively clear differences in the distribution of these characteristics in three diagnostic classes, so that they are highly relevant in the prediction of pancreatic cancer.

3 Methods

3.1 Data preprocessing

Initially, redundant columns from the original dataset are removed; these columns include sample ID, sample origin, group of patients in the research, specific pancreatic disease diagnostic for non-cancerous patients, and cancer stages for PDAC-suffering patients. From the original dataset, some entry values for plasma CA19-9 and REG1A were missing. If all the data points with a missing value are removed, there will be only 209 data points

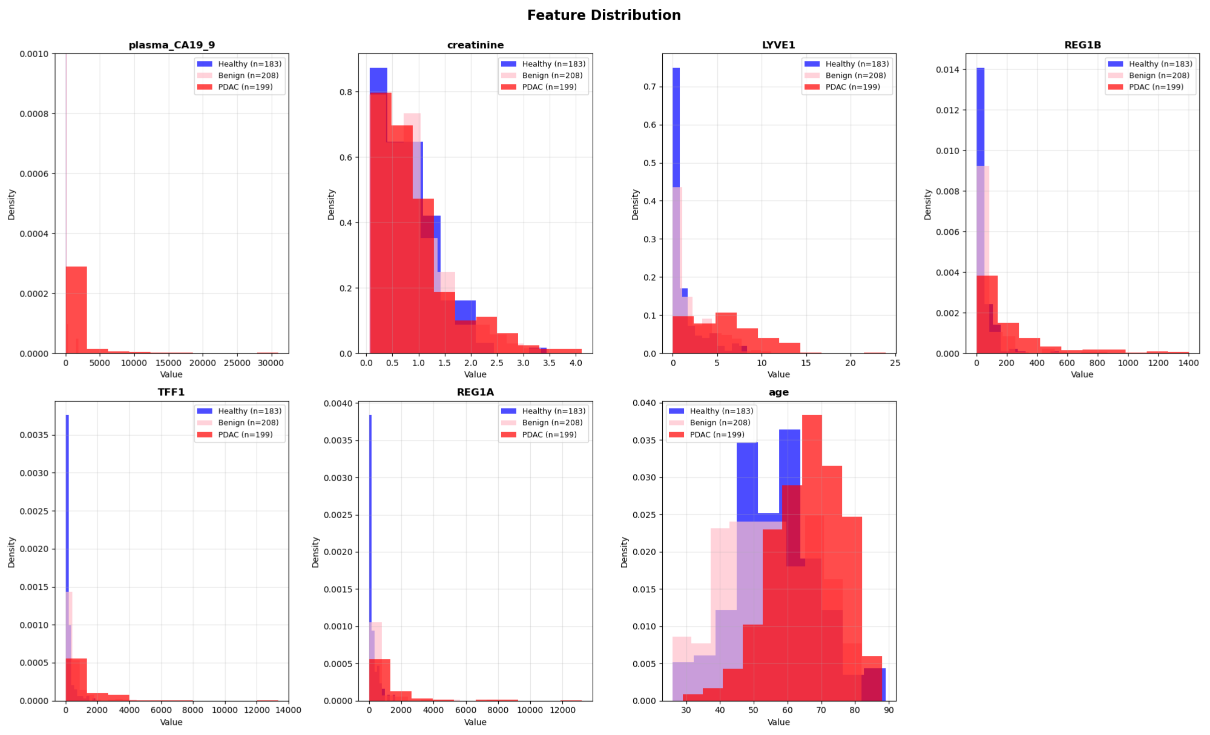


Figure 1: Distribution of the features respect to three diagnostic targets.

remaining, which is insufficient for this ML model to effectively study seven features. Therefore, either the mean or the median of the existing values from the corresponding features is used to compromise these missing entries. From the histogram for each feature with respect to three target diagnostic categories (healthy, benign, and PDAC) (Figure 1), the data is extremely skewed for plasma CA19-9 and REG1A. Therefore, the median is selected to fill in the missing data entries.

3.2 ML model

3.2.1 Multinomial logistic regression

Logistic regression could be a suitable algorithm for this classification task. With three targets, the original binary logistic regression classifier may not be suitable, so in this case, a multinomial logistic regression (MLR) is implemented. In short, for each of these three classes, the MLR model computes a logit following the dot product:

$$\mathrm{score}(\mathbf{X}_i, k) = \boldsymbol{\beta}_k \cdot \mathbf{X}_i,$$

where $\mathbf{X}_i$ is the vector of features' values for patient i, and $\boldsymbol{\beta}_k$ is the regression coefficients for target k (Wikipedia, 2025). Subsequently, the probability that patient i is classified in category k is computed with the softmax function:

$$\Pr(Y_i = k) = \mathrm{softmax}(k, \boldsymbol{\beta}_1 \cdot \mathbf{X}_i, \ldots, \boldsymbol{\beta}_K \cdot \mathbf{X}_i)$$

MLR is primarily selected because it's directly suitable for this multi-category task. Furthermore, through the underlying algorithms, its results are potentially interpretable for the doctors in case they need to comprehensively understand or examine how the clinical decision is predicted.

3.2.2 Random Forest

Random forest (RF) is the second method being considered for this classification problem. In essence, RF combines two techniques, bagging and feature randomness, to generate several decision trees. Specifically, each tree is trained on a random subset (bagging), and each tree evaluates a random subset of features (feature randomness) so that the decision trees in the forest are uncorrelated (IBM, 2025). This method is robust and less sensitive to outliers and skewed distributions. Therefore, it is suitable for this problem, which has a multi-dimensional dataset and some skewed features. In addition, unlike logistic regression, this approach does not assume a linear relationship between features. Thus, RF is an appropriate method to be compared with MLR, and is potentially more suitable to evaluate complex biological correlations.

3.3 Loss function

With respect to an MLR model, multinomial logistic loss (cross-entropy loss) is selected as the most suitable loss function (Gn, 2023). Accordingly, this loss function is also used to train the RF model. Regarding the algorithm of this loss function, initially, a true label y—a one-hot encoded vector—is created where only the correct target is marked as 1 (e.g., [0,0,1] if the PDAC class is correct), while the model generates a vector $\hat{y}$ containing predicted probabilities for each target. Subsequently, the loss for a single sample is generally calculated as:

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -\sum_{k=1}^{K} y_k \log(\hat{y}_k) \\ &= -\log \hat{y}_c \end{aligned}$$

where K here is the number of targets.

3.4 Model validation

The size of this dataset is relatively small, and fitting with the median is already carried out in the data preprocessing stage. Thus, to enhance the reliability of this process, k-fold cross-validation is implemented. For this method, the dataset is divided into k equal-sized subsets, or "folds." The model is iteratively trained k times; in each iteration, it is trained on k - 1 subsets (training sets) and validated on the single remaining subset (validation set) (Refaeilzadeh et al., 2009). In this case, $k = 5$ is selected so that each fold contains 118 data points. Therefore, in each iteration, the model is trained with 472 samples (in $k-1 = 4$ folds) and validated with 118 samples. Eventually, after 5 iterations, the average performance score and corresponding standard deviation are computed.

4 Results

From the appendix code, the average training loss and validation loss across 5 folds of MLR are 0.74 and 0.77, respectively. Meanwhile, the corresponding values for RF are 0.17 for training loss and 0.64 for validation loss. As the training and validation losses of MLR are considerably close, this model is not overfit. However, these errors are relatively high; therefore, MLR's accuracy score is only 63.05%, which is possibly not reliable enough for a medical decision. In contrast, the RF model is overfitting since there is a significant gap between validation and training losses. RF model achieves an overall 72.03% accuracy score, with a higher performance score than that of MLR in almost every fold. Thus, it outperforms the MLR model. Based on these results, RF is selected as the final approach for this classification task.

5 Conclusion

In summary, this ML project develops a multi-class classifier with the random forest to diagnose pancreatic cancer mainly based on urinary biomarkers. The final model achieves a test accuracy of 77.12% and a test loss of 0.5688, with 80% training data and 20% validation data.

Based on the appendix confusion matrices and plots, both MLR and RF clearly demonstrate higher correctness when identifying PDAC samples (category 3), while errors often occur between healthy and benign samples (categories 1 and 2). The possible interpretation for this issue could be the high similarities in the biological profiles of some healthy and benign samples. Thus, to further investigate the applicability of this model, RF is implemented again with a binary classification task, where the features comprise 6 biomarker levels and patients' ages, and the target includes only two categories, in which 0 indicates non-cancerous samples, and 1 indicates PDAC ones. In this task, the ML model gained a significantly higher result with an accuracy score of 90%. This result shows that biomarker data is a promising indicator for an efficient non-invasive screening tool.

However, these results directly indicate the need for further improvement. The first approach is to increase the size of this dataset, since the number of samples is small and some features have missing values. When screening through the works of other developers with this dataset on Kaggle, I found that gradient boosting machines (GBMs), such as LightGBM and XGBoost, are highly suitable for this task. These approaches can be found in the 'Codes' section on the Kaggle data source attached above. In general, GBMs are also tree-ensembled techniques, in which a tree is built once at a time and the new tree is trained to improve the performance of the previous trees (boosting). These approaches demonstrated better performance with non-linear biological relationships and tabular data of this problem. For example, several GBM-based works on Kaggle achieve approximately 90% accuracy score in a relatively similar categorical classification task; thus, these methods have high potential in improving ML prediction correctness in this problem.

References

Debernardi, S., O'Brien, H., Algahmdi, A. S., Malats, N., Stewart, G. D., Plješa-Ercegovac, M., Costello, E., Greenhalf, W., Saad, A., Roberts, R., Ney, A., Pereira, S. P., Kocher, H. M., Duffy, S., Blyuss, O., & Crnogorac-Jurcevic, T. (2020). A combination of urinary biomarker panel and PancRISK score for earlier detection of pancreatic cancer: A case–control study. *PLoS Medicine*, *17*(12), e1003489. https://doi.org/10.1371/journal.pmed.1003489

Gn, S. (2023, February). Multinomial Logistic Regression: Defintion, math, and implementation. https://www.quarkml.com/2022/03/multinomial-logistic-regression-definition-math-and-implementation.html

IBM. (2025, June). Random Forest. https://www.ibm.com/think/topics/random-forest

Refaeilzadeh, P., Tang, L., & Liu, H. (2009). *Cross-Validation.* https://doi.org/10.1007/978-0-387-39940-9\{_}565

Wikipedia. (2025, March). Multinomial logistic regression. https://en.wikipedia.org/wiki/Multinomial_logistic_regression

A Appendix

Table 1: Features and target examined in this application (adapted from Debernardi et al. (2020))

Column Name	**Role**	**Type**	**Details**
Age	Feature	Continuous	Age in years
Diagnosis	Target	Categorical	1 = control (no pancreatic disease), 2 = benign hepatobiliary disease (119 of which are chronic pancreatitis); 3 = Pancreatic ductal adenocarcinoma, i.e., pancreatic cancer
Plasma CA19-9 (U/ml)	Feature	Continuous	Blood plasma levels of CA 19–9 monoclonal antibody that is often elevated in patients with pancreatic cancer. Only assessed in 350 patients.
Creatinine (mg/ml)	Feature	Continuous	Urinary biomarker of kidney function
LYVE1 (ng/ml)	Feature	Continuous	Urinary levels of Lymphatic vessel endothelial hyaluronan receptor 1, a protein that may play a role in tumor metastasis
REG1B (ng/ml)	Feature	Continuous	Urinary levels of a protein that may be associated with pancreas regeneration.
TFF1 (ng/ml)	Feature	Continuous	Urinary levels of Trefoil Factor 1, which may be related to regeneration and repair of the urinary tract
REG1A (ng/ml)	Feature	Continuous	Urinary levels of a protein that may be associated with pancreas regeneration. Only assessed in 306 patients.

projectML

October 8, 2025

```
[1]: from IPython.display import display, HTML
```

```
[3]: display(HTML("""<div style='
         background: white;
         padding: 10px;
         color: black;
         font-weight: bold;
         font-size: 24px;
         text-align: left;'>
     Section 0: Importing
     </div>"""))
```

```
<IPython.core.display.HTML object>
```

```
[162]: import warnings

       import numpy as np
       import pandas as pd
       import matplotlib.pyplot as plt
       import seaborn as sns

       from sklearn.linear_model import LogisticRegression
       from sklearn.ensemble import RandomForestClassifier
       from sklearn.metrics import accuracy_score, log_loss, confusion_matrix,␣
        ↪classification_report

       from sklearn.datasets import fetch_openml
       from sklearn.model_selection import train_test_split, StratifiedKFold
       from sklearn.preprocessing import StandardScaler, LabelEncoder, RobustScaler
```

```
[6]: display(HTML("""<div style='
         background: white;
         padding: 10px;
         color: black;
         font-weight: bold;
         font-size: 24px;
         text-align: left;'>
Section 1: Data Processing
</div>"""))
```

```
<IPython.core.display.HTML object>
```

[7]:
```
df = pd.read_csv('Debernardi et al 2020 data.csv')

# remove redundant columns
df.drop(columns=['sample_id', 'benign_sample_diagnosis', 'stage',␣
 ↪'patient_cohort', 'sample_origin'],inplace=True)

# change gender type representation M/F to binary
label_encoder = LabelEncoder()
df['sex']= label_encoder.fit_transform(df['sex'])
df['sex']=df['sex'].astype(int)

# display current dataset
df.info()
display(df.describe())
df.head(5)
```

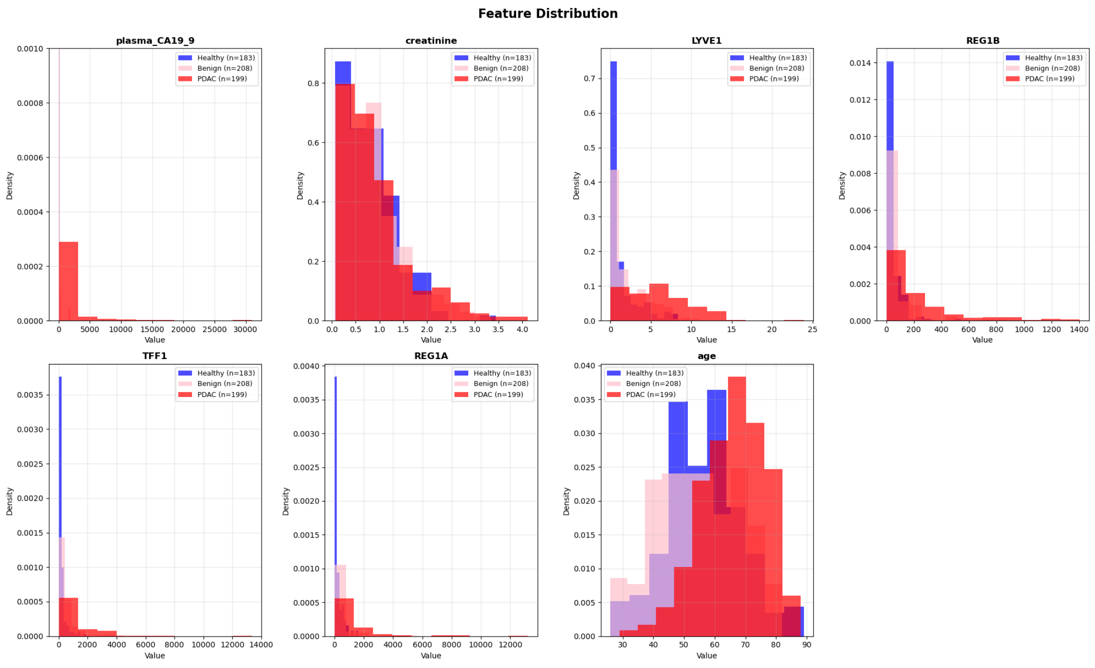
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 590 entries, 0 to 589
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   age            590 non-null    int64
 1   sex            590 non-null    int64
 2   diagnosis      590 non-null    int64
 3   plasma_CA19_9  350 non-null    float64
 4   creatinine     590 non-null    float64
 5   LYVE1          590 non-null    float64
 6   REG1B          590 non-null    float64
 7   TFF1           590 non-null    float64
 8   REG1A          306 non-null    float64
dtypes: float64(6), int64(3)
memory usage: 41.6 KB
```

```
              age         sex   diagnosis  plasma_CA19_9  creatinine  \
count  590.000000  590.000000  590.000000     350.000000  590.000000
mean    59.079661    0.493220    2.027119     654.002944    0.855383
std     13.109520    0.500378    0.804873    2430.317642    0.639028
min     26.000000    0.000000    1.000000       0.000000    0.056550
25%     50.000000    0.000000    1.000000       8.000000    0.373230
50%     60.000000    0.000000    2.000000      26.500000    0.723840
75%     69.000000    1.000000    3.000000     294.000000    1.139482
max     89.000000    1.000000    3.000000   31000.000000    4.116840
```

```
             LYVE1        REG1B          TFF1         REG1A
count   590.000000   590.000000    590.000000    306.000000
mean      3.063530   111.774090    597.868722    735.281222
std       3.438796   196.267110   1010.477245   1477.247724
min       0.000129     0.001104      0.005293      0.000000
25%       0.167179    10.757216     43.961000     80.692000
50%       1.649862    34.303353    259.873974    208.538500
75%       5.205037   122.741013    742.736000    649.000000
max      23.890323  1403.897600  13344.300000  13200.000000
```

[7]:
```
   age  sex  diagnosis  plasma_CA19_9  creatinine     LYVE1      REG1B  \
0   33    0          1           11.7     1.83222  0.893219   52.94884
1   81    0          1            NaN     0.97266  2.037585   94.46703
2   51    1          1            7.0     0.78039  0.145589  102.36600
3   61    1          1            8.0     0.70122  0.002805   60.57900
4   62    1          1            9.0     0.21489  0.000860   65.54000

         TFF1     REG1A
0  654.282174  1262.000
1  209.488250   228.407
2  461.141000       NaN
3  142.950000       NaN
4   41.088000       NaN
```

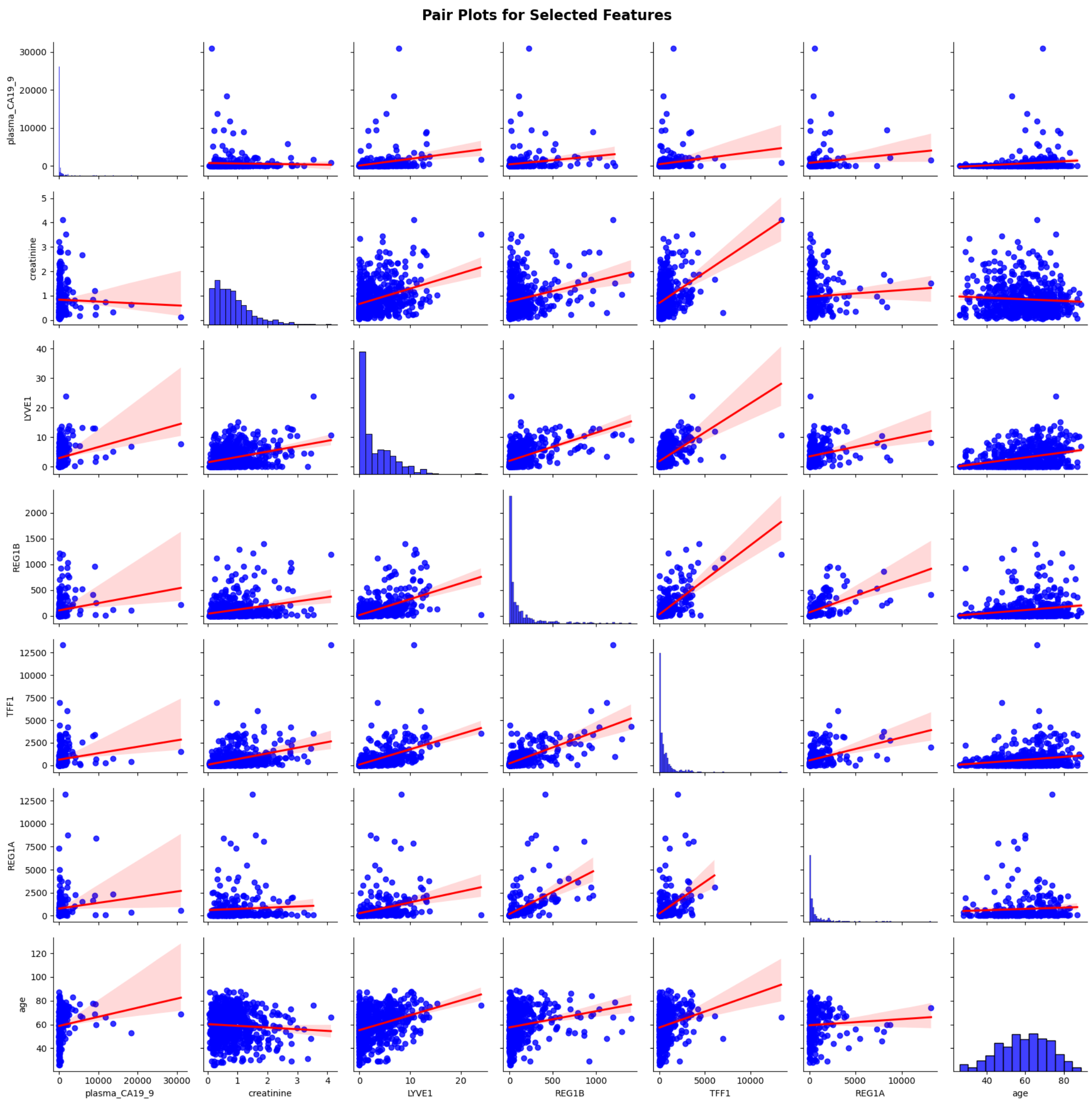
[9]:
```python
sns.set_palette("pastel")

features = ['plasma_CA19_9', 'creatinine', 'LYVE1', 'REG1B', 'TFF1', 'REG1A',␣
 ↪'age']
diagnosis_colors = {1: 'blue', 2: 'pink', 3: 'red'}
diagnosis_labels = {1: 'Healthy', 2: 'Benign', 3: 'PDAC'}

fig, axes = plt.subplots(2, 4, figsize=(20, 12))
fig.suptitle('Feature Distribution', fontsize=16, fontweight='bold', y=1)

axes = axes.flatten()

# plot histograms
for i, feature in enumerate(features):
    ax = axes[i]

    # for each diagnosis category
    for diagnosis in [1, 2, 3]:
        subset = df[df['diagnosis'] == diagnosis]
        ax.hist(subset[feature], alpha=0.7,
                color=diagnosis_colors[diagnosis],
                label=f'{diagnosis_labels[diagnosis]} (n={len(subset)})',
                density=True)
    if feature == 'plasma_CA19_9':
        ax.set_ylim(0, 0.001)

    # style
    ax.set_title(f'{feature}', fontweight='bold', fontsize=12)
    ax.set_xlabel('Value', fontsize=10)
    ax.set_ylabel('Density', fontsize=10)
    ax.grid(True, alpha=0.3)
    ax.legend(fontsize=9)

axes[7].set_visible(False)

plt.tight_layout()
plt.show()
```



[10]:
```
# pair plots
plt.figure(figsize=(12, 10))
pairplot = sns.pairplot(df[features], kind='reg', diag_kind='hist',
                        plot_kws={'scatter_kws': {'color': 'blue'},'line_kws':␣
↪{'color': 'red'}},
                        diag_kws={'color': 'blue'})

pairplot.fig.suptitle('Pair Plots for Selected Features', y=1, fontsize=16,␣
↪fontweight='bold',)
```

```
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x1000 with 0 Axes>
```



[15]:
```
# replace missing values with the corresponding median
# (from the plot, we see that the median is more suitable than the mean)
df['plasma_CA19_9']=df['plasma_CA19_9'].fillna(df['plasma_CA19_9'].median())
df['REG1A']=df['REG1A'].fillna(df['REG1A'].median())

# display updated dataset
df.info()
display(df.describe())
```

```
df.head(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 590 entries, 0 to 589
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   age            590 non-null    int64
 1   sex            590 non-null    int64
 2   diagnosis      590 non-null    int64
 3   plasma_CA19_9  590 non-null    float64
 4   creatinine     590 non-null    float64
 5   LYVE1          590 non-null    float64
 6   REG1B          590 non-null    float64
 7   TFF1           590 non-null    float64
 8   REG1A          590 non-null    float64
dtypes: float64(6), int64(3)
memory usage: 41.6 KB

              age         sex   diagnosis  plasma_CA19_9  creatinine  \
count  590.000000  590.000000  590.000000     590.000000  590.000000
mean    59.079661    0.493220    2.027119     398.747509    0.855383
std     13.109520    0.500378    0.804873    1896.028213    0.639028
min     26.000000    0.000000    1.000000       0.000000    0.056550
25%     50.000000    0.000000    1.000000      17.000000    0.373230
50%     60.000000    0.000000    2.000000      26.500000    0.723840
75%     69.000000    1.000000    3.000000      41.750000    1.139482
max     89.000000    1.000000    3.000000   31000.000000    4.116840

            LYVE1        REG1B          TFF1         REG1A
count  590.000000   590.000000    590.000000    590.000000
mean     3.063530   111.774090    597.868722    481.730488
std      3.438796   196.267110   1010.477245   1095.179818
min      0.000129     0.001104      0.005293      0.000000
25%      0.167179    10.757216     43.961000    195.201000
50%      1.649862    34.303353    259.873974    208.538500
75%      5.205037   122.741013    742.736000    224.007000
max     23.890323  1403.897600  13344.300000  13200.000000
```

```
[15]:    age  sex  diagnosis  plasma_CA19_9  creatinine     LYVE1      REG1B  \
      0   33    0          1           11.7     1.83222  0.893219   52.94884
      1   81    0          1           26.5     0.97266  2.037585   94.46703
      2   51    1          1            7.0     0.78039  0.145589  102.36600
      3   61    1          1            8.0     0.70122  0.002805   60.57900
      4   62    1          1            9.0     0.21489  0.000860   65.54000

               TFF1      REG1A
      0  654.282174  1262.0000
```

```
1  209.488250   228.4070
2  461.141000   208.5385
3  142.950000   208.5385
4   41.088000   208.5385
```

```
[16]: X = df.drop(columns=['diagnosis'])
      y = df['diagnosis']
```

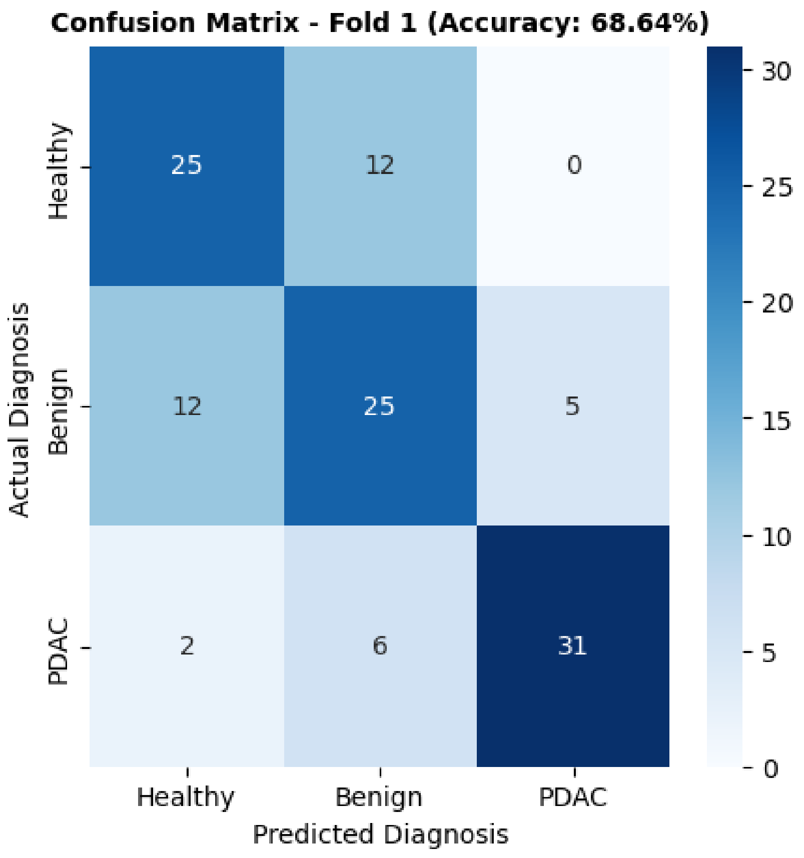
```
[17]: display(HTML("""<div style='
          background: white;
          padding: 10px;
          color: black;
          font-weight: bold;
          font-size: 24px;
          text-align: left;'>
      Section 2.1: Multinomial Logistic Regression
      </div>"""))
```

```
<IPython.core.display.HTML object>
```

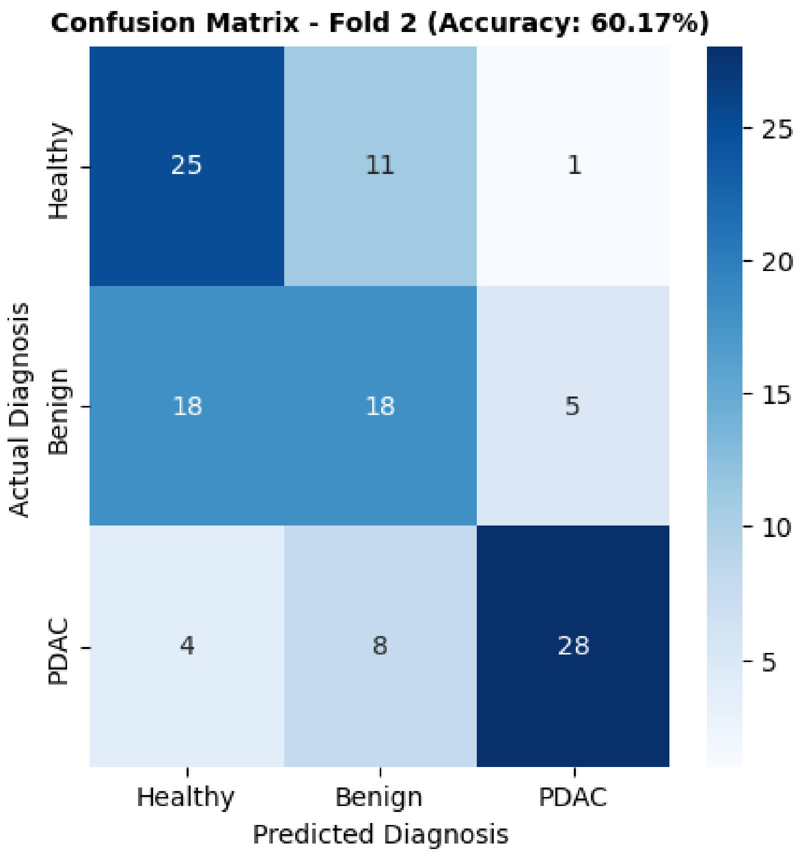
```
[69]: # Usage refers to this tutorial: https://scikit-learn.org/stable/modules/
       ↪generated/sklearn.model_selection.StratifiedKFold.html
      skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

      mlr = LogisticRegression(multi_class='multinomial', solver='lbfgs',␣
       ↪random_state=42, max_iter=300000)

      val_losses = []
      train_losses = []
      accuracy_scores = []

      for i, (train_index, test_index) in enumerate(skf.split(X, y)):
          # training
          X_train0, X_test0 = X.iloc[train_index], X.iloc[test_index]
          y_train, y_test = y.iloc[train_index], y.iloc[test_index]

          scaler = StandardScaler()
          X_train = scaler.fit_transform(X_train0)
          X_test = scaler.transform(X_test0)

          mlr.fit(X_train, y_train)

          y_val_pred_4loss = mlr.predict_proba(X_test)
          y_train_pred_4loss = mlr.predict_proba(X_train)
          y_pred = mlr.predict(X_test)

          # evaluating
```

```
    val_loss = log_loss(y_test, y_val_pred_4loss)
    train_loss = log_loss(y_train, y_train_pred_4loss)
    accuracy = accuracy_score(y_test, y_pred)

    val_losses.append(val_loss)
    train_losses.append(train_loss)
    accuracy_scores.append(accuracy)

    # result
    print(f"Fold {i+1}: Accuracy = {accuracy*100:.2f}%, Validation loss =␣
 ↪{val_loss:.2f}, Training loss: {train_loss:.2f}")

    # plot confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    targets = ['Healthy', 'Benign', 'PDAC']

    plt.figure(figsize=(5, 5))
    sns.heatmap(conf_matrix, annot=True, cmap='Blues',
                xticklabels=targets, yticklabels=targets)
    plt.title(f'Confusion matrix - fold {i+1} (Accuracy: {accuracy:.2%})',␣
 ↪fontsize=10, fontweight='bold')
    plt.ylabel('Actual diagnosis', fontsize=10)
    plt.xlabel('Predicted diagnosis', fontsize=10)
    plt.show()

print(f"Average validation loss: {np.mean(val_losses):.2f}")
print(f"Average train loss: {np.mean(train_losses):.2f}")
print(f"Average accuracy: {np.mean(accuracy_scores)*100:.2f}%")
```

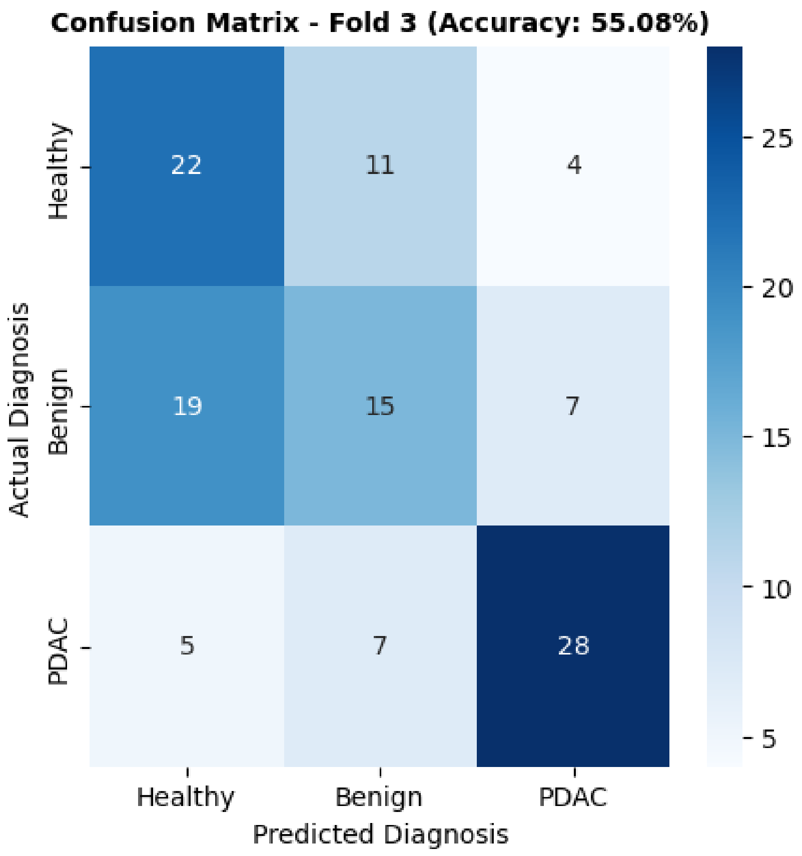
```
Fold 1: Accuracy = 68.64%, Validation Loss = 0.75, Training Loss: 0.74
```



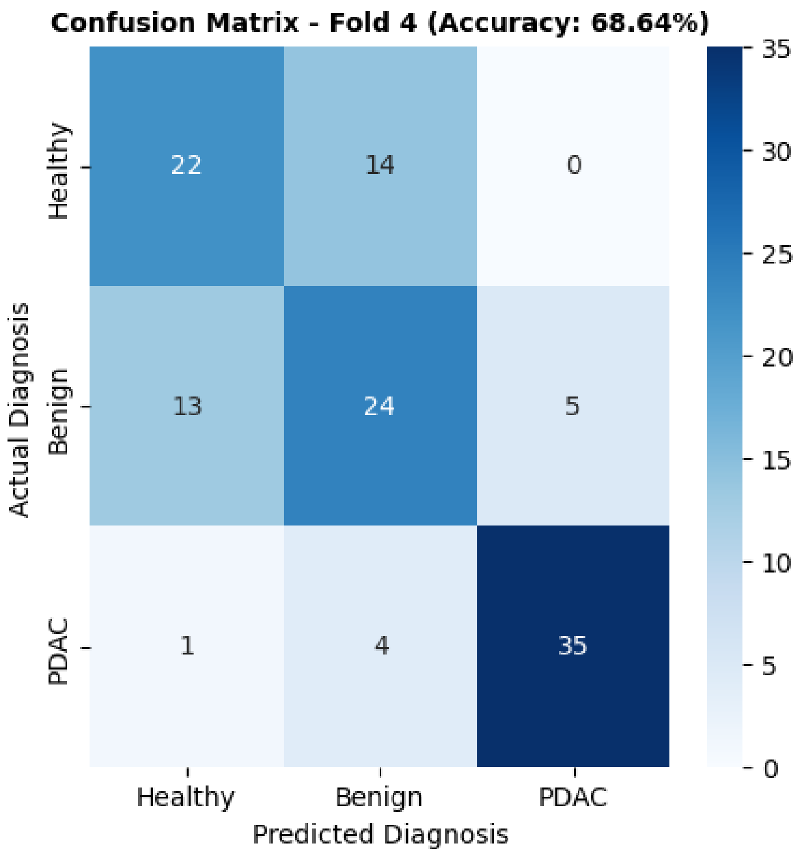
```
Fold 2: Accuracy = 60.17%, Validation Loss = 0.77, Training Loss: 0.74
```



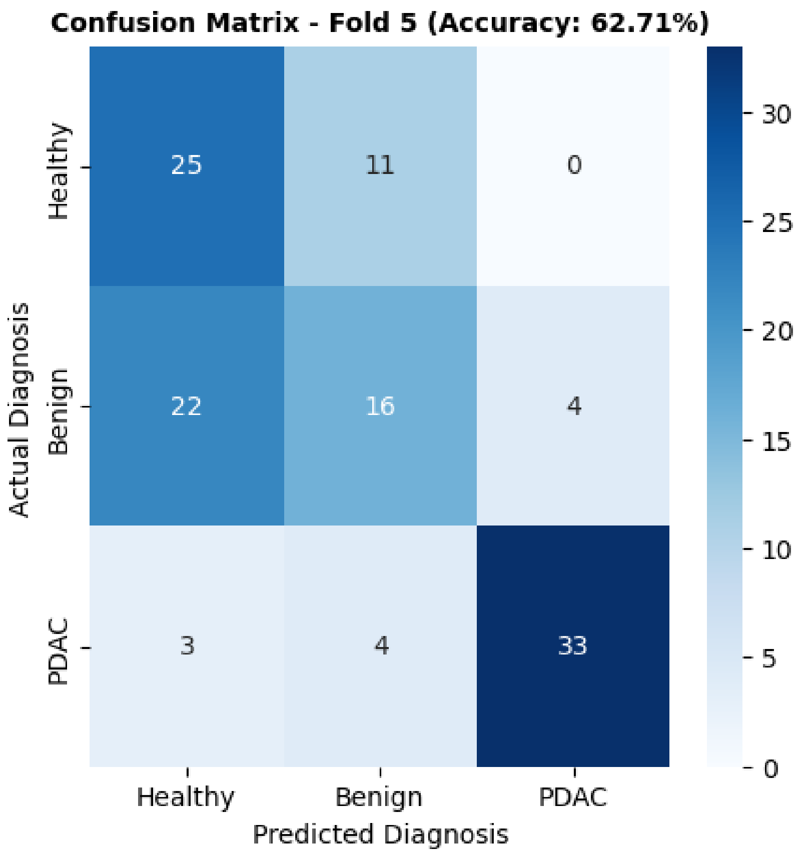
```
Fold 3: Accuracy = 55.08%, Validation Loss = 0.89, Training Loss: 0.71
```



```
Fold 4: Accuracy = 68.64%, Validation Loss = 0.70, Training Loss: 0.75
```



```
Fold 5: Accuracy = 62.71%, Validation Loss = 0.73, Training Loss: 0.74
```



```
Average Validation Loss: 0.77
Average Train Loss: 0.74
Average Accuracy: 63.05%
```

[70]:
```
display(HTML("""<div style='
    background: white;
    padding: 10px;
    color: black;
    font-weight: bold;
    font-size: 24px;
    text-align: left;'>
Section 2.2: Random Forest
</div>"""))
```

```
<IPython.core.display.HTML object>
```

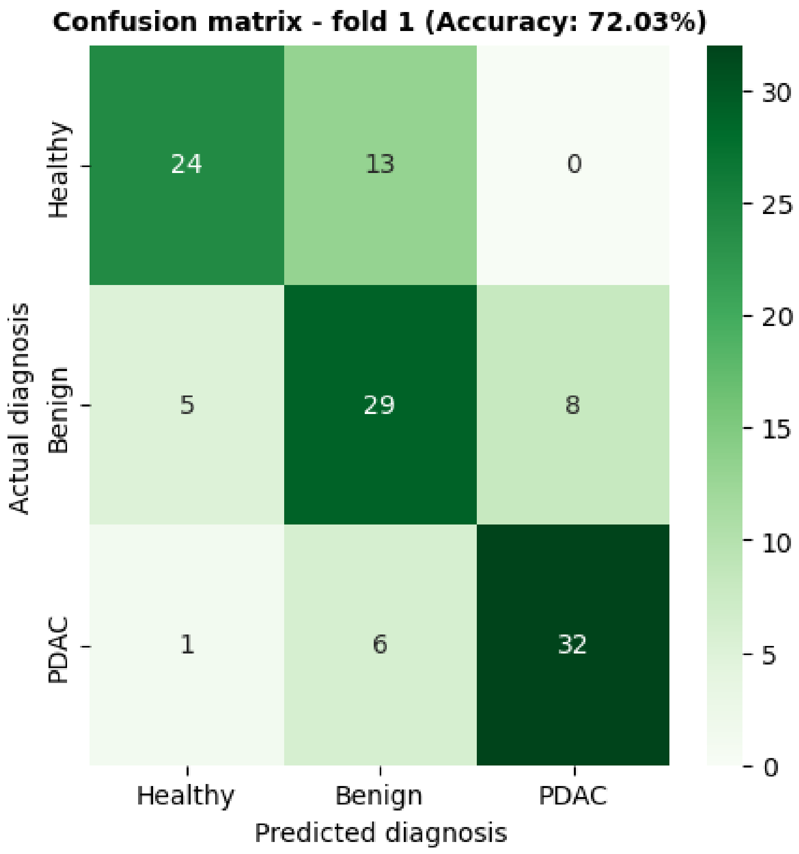
[112]:
```
rf = RandomForestClassifier(n_estimators=100)

accuracy_scores_rf = []
```

```
val_losses_rf = []
train_losses_rf = []

for i, (train_index, test_index) in enumerate(skf.split(X, y)):
    # training
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    rf.fit(X_train, y_train)

    y_pred = rf.predict(X_test)

    # evaluating
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores_rf.append(accuracy)

    y_val_pred_4loss = rf.predict_proba(X_test)
    y_train_pred_4loss = rf.predict_proba(X_train)

    val_loss = log_loss(y_test, y_val_pred_4loss)
    train_loss = log_loss(y_train, y_train_pred_4loss)

    val_losses_rf.append(val_loss)
    train_losses_rf.append(train_loss)

    # result
    print(f"Fold {i+1}: Accuracy = {accuracy*100:.2f}%, Validation loss =␣
 ↪{val_loss:.2f}, Training loss: {train_loss:.2f}")


    report = classification_report(y_test, y_pred, target_names=['Healthy',␣
 ↪'Benign', 'PDAC']) # Gini is implemented by default
    print(report)

    # plot confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    targets = ['Healthy', 'Benign', 'PDAC']

    plt.figure(figsize=(5, 5))
    sns.heatmap(conf_matrix, annot=True, cmap='Greens',
            xticklabels=targets, yticklabels=targets)
    plt.title(f'Confusion matrix - fold {i+1} (Accuracy: {accuracy:.2%})',␣
 ↪fontsize=10, fontweight='bold')
    plt.ylabel('Actual diagnosis', fontsize=10)
    plt.xlabel('Predicted diagnosis', fontsize=10)
    plt.show()
```

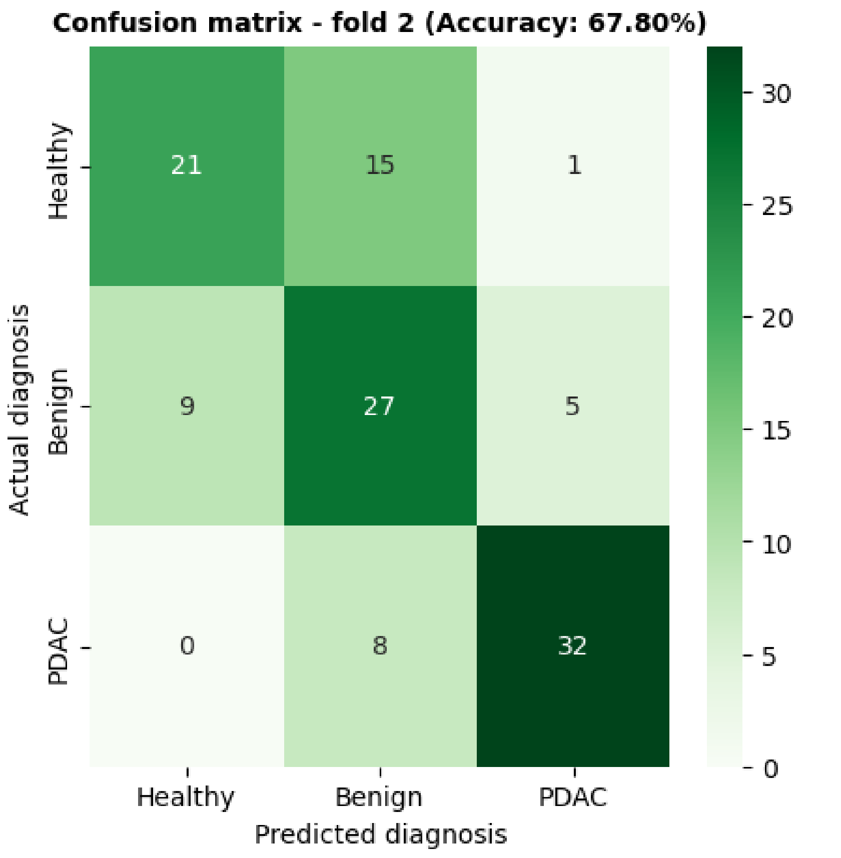
```
   # plot comparison between predicted and real values for model with larger␣
↪than 75% accuracy score.
   if accuracy > 0.75:

       results_df = pd.DataFrame({
           'Actual': y_test,
           'Predicted': y_pred,
           'Index': y_test.index
       })

       correct_preds = results_df[results_df['Actual'] ==␣
↪results_df['Predicted']]
       incorrect_preds = results_df[results_df['Actual'] !=␣
↪results_df['Predicted']]

       plt.figure()

       plt.scatter(correct_preds['Index'], correct_preds['Actual'],
                   alpha=0.7, facecolors='none', edgecolors='green',
                   linewidths=1.5, label='Correct prediction', marker='o')
       plt.scatter(incorrect_preds['Index'], incorrect_preds['Predicted'],
                   color='pink', alpha=1.0, label='Incorrect prediction',␣
↪marker='x')

       plt.yticks([1, 2, 3], targets)
       plt.ylabel('Diagnosis', fontsize=10)
       plt.xlabel('Sample index', fontsize=10)
       plt.title(f'Predicted vs. Actual values for fold {i+1} (Accuracy:␣
↪{accuracy:.2%})', fontsize=12, fontweight='bold')
       plt.legend()
       plt.grid(axis='y', linestyle='--', alpha=0.7)
       plt.show()

print(f"Average validation loss: {np.mean(val_losses_rf):.2f}")
print(f"Average train loss: {np.mean(train_losses_rf):.2f}")
print(f"Average accuracy: {np.mean(accuracy_scores_rf)*100:.2f}%")
```

```
Fold 1: Accuracy = 72.03%, Validation loss = 0.65, Training loss: 0.17
              precision    recall  f1-score   support

     Healthy       0.80      0.65      0.72        37
      Benign       0.60      0.69      0.64        42
        PDAC       0.80      0.82      0.81        39

    accuracy                           0.72       118
   macro avg       0.73      0.72      0.72       118
```

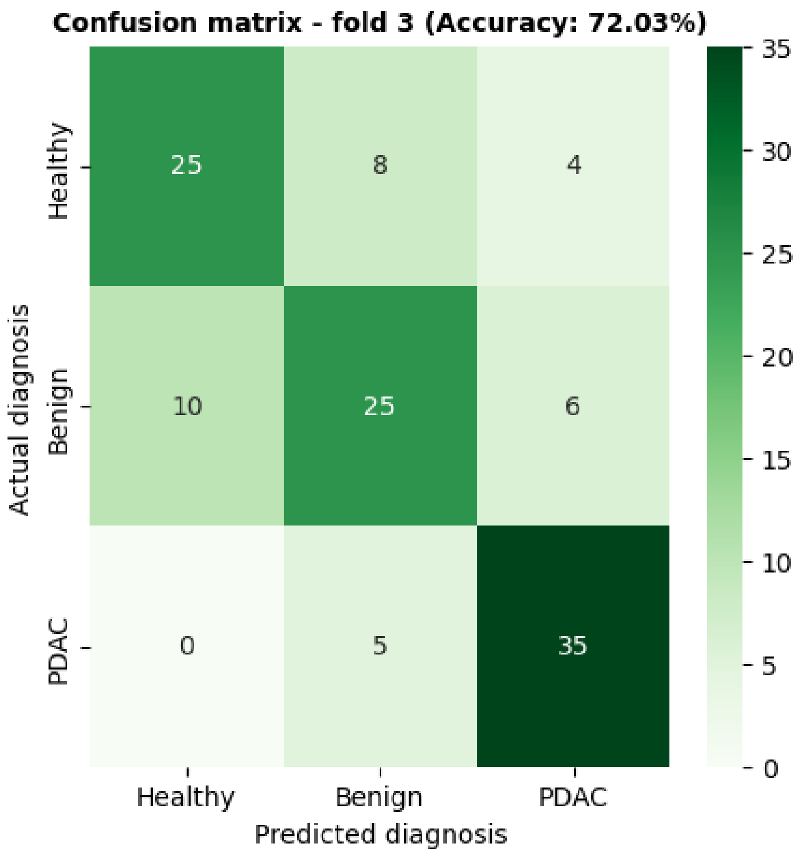
```
weighted avg       0.73      0.72      0.72       118
```



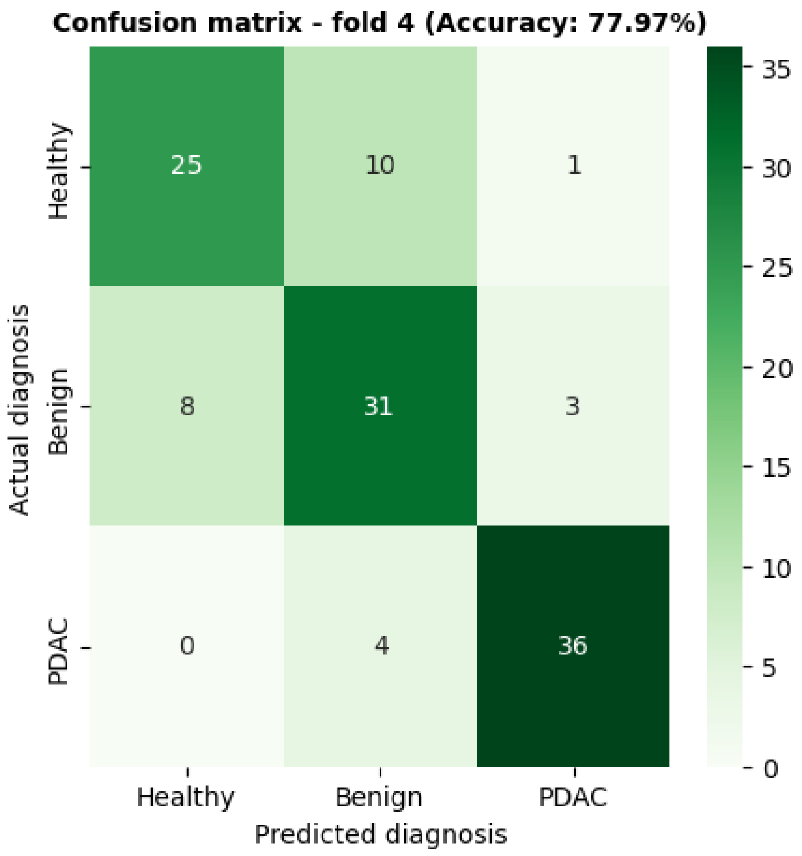
```
Fold 2: Accuracy = 67.80%, Validation loss = 0.65, Training loss: 0.17
              precision    recall  f1-score   support

     Healthy       0.70      0.57      0.63        37
      Benign       0.54      0.66      0.59        41
        PDAC       0.84      0.80      0.82        40

    accuracy                           0.68       118
   macro avg       0.69      0.68      0.68       118
weighted avg       0.69      0.68      0.68       118
```

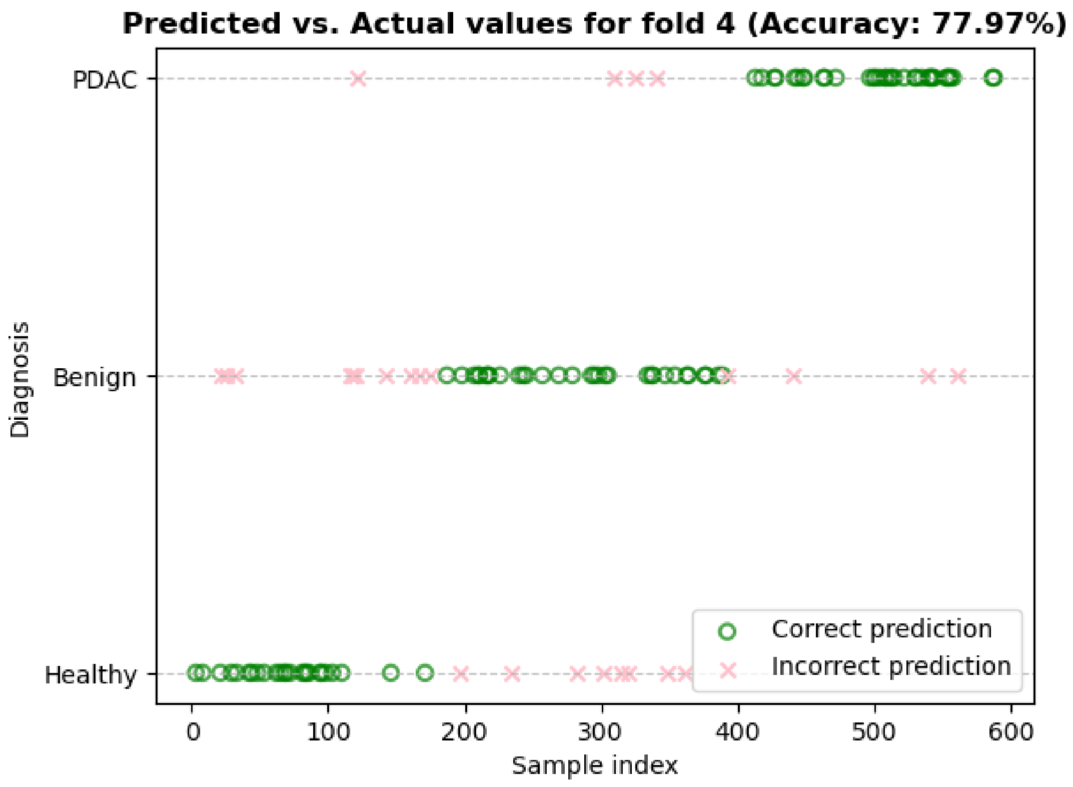


```
Fold 3: Accuracy = 72.03%, Validation loss = 0.70, Training loss: 0.17
              precision    recall  f1-score   support

     Healthy       0.71      0.68      0.69        37
      Benign       0.66      0.61      0.63        41
        PDAC       0.78      0.88      0.82        40

    accuracy                           0.72       118
   macro avg       0.72      0.72      0.72       118
weighted avg       0.72      0.72      0.72       118
```

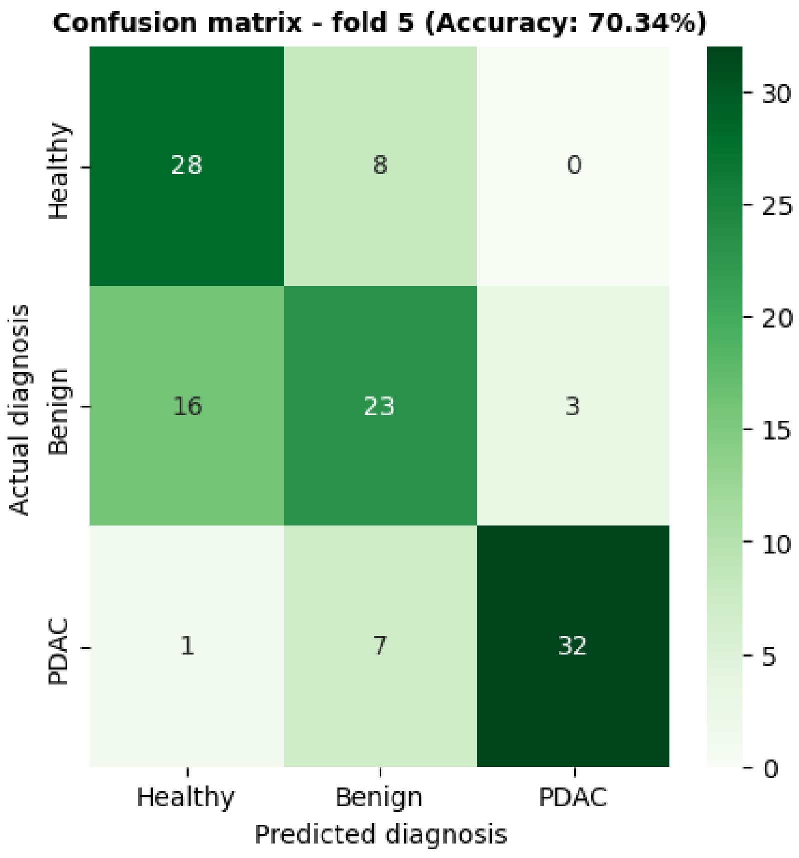


```
Fold 4: Accuracy = 77.97%, Validation loss = 0.59, Training loss: 0.18
              precision    recall  f1-score   support

     Healthy       0.76      0.69      0.72        36
      Benign       0.69      0.74      0.71        42
        PDAC       0.90      0.90      0.90        40

    accuracy                           0.78       118
   macro avg       0.78      0.78      0.78       118
weighted avg       0.78      0.78      0.78       118
```


Confusion matrix - fold 4 (Accuracy: 77.97%)
Actual diagnosis
Healthy
Benign
PDAC
Predicted diagnosis
Healthy
Benign
PDAC
25
10
1
8
31
3
0
4
36
35
30
25
20
15
10
5
0



```
Fold 5: Accuracy = 70.34%, Validation loss = 0.64, Training loss: 0.17
              precision    recall  f1-score   support

     Healthy       0.62      0.78      0.69        36
      Benign       0.61      0.55      0.57        42
        PDAC       0.91      0.80      0.85        40

    accuracy                           0.70       118
   macro avg       0.71      0.71      0.71       118
weighted avg       0.72      0.70      0.70       118
```



```
Average validation loss: 0.64
Average train loss: 0.17
Average accuracy: 72.03%
```

[143]:
```
for dropped_feature in features:
    X_dropped = X.drop(columns=[dropped_feature])

    accuracy_scores_rf_test_drop = []
    losses_rf_test_drop = []

    for i, (train_index, test_index) in enumerate(skf.split(X, y)):
        # training
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        rf.fit(X_train, y_train)

        y_pred = rf.predict(X_test)
        # evaluating
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_scores_rf.append(accuracy)

        y_val_pred_4loss = rf.predict_proba(X_test)
        val_loss = log_loss(y_test, y_val_pred_4loss)
        losses_rf_test_drop.append(val_loss)

    print(f"When {dropped_feature} is dropped: Average loss: {np.
 ↪mean(val_losses_rf):.2f}, Average accuracy: {np.mean(accuracy_scores_rf)*100:
 ↪.2f}%")
```

```
When plasma_CA19_9 is dropped: Average loss: 0.64, Average accuracy: 71.77%
When creatinine is dropped: Average loss: 0.64, Average accuracy: 71.78%
When LYVE1 is dropped: Average loss: 0.64, Average accuracy: 71.77%
When REG1B is dropped: Average loss: 0.64, Average accuracy: 71.73%
When TFF1 is dropped: Average loss: 0.64, Average accuracy: 71.75%
When REG1A is dropped: Average loss: 0.64, Average accuracy: 71.76%
When age is dropped: Average loss: 0.64, Average accuracy: 71.75%
```

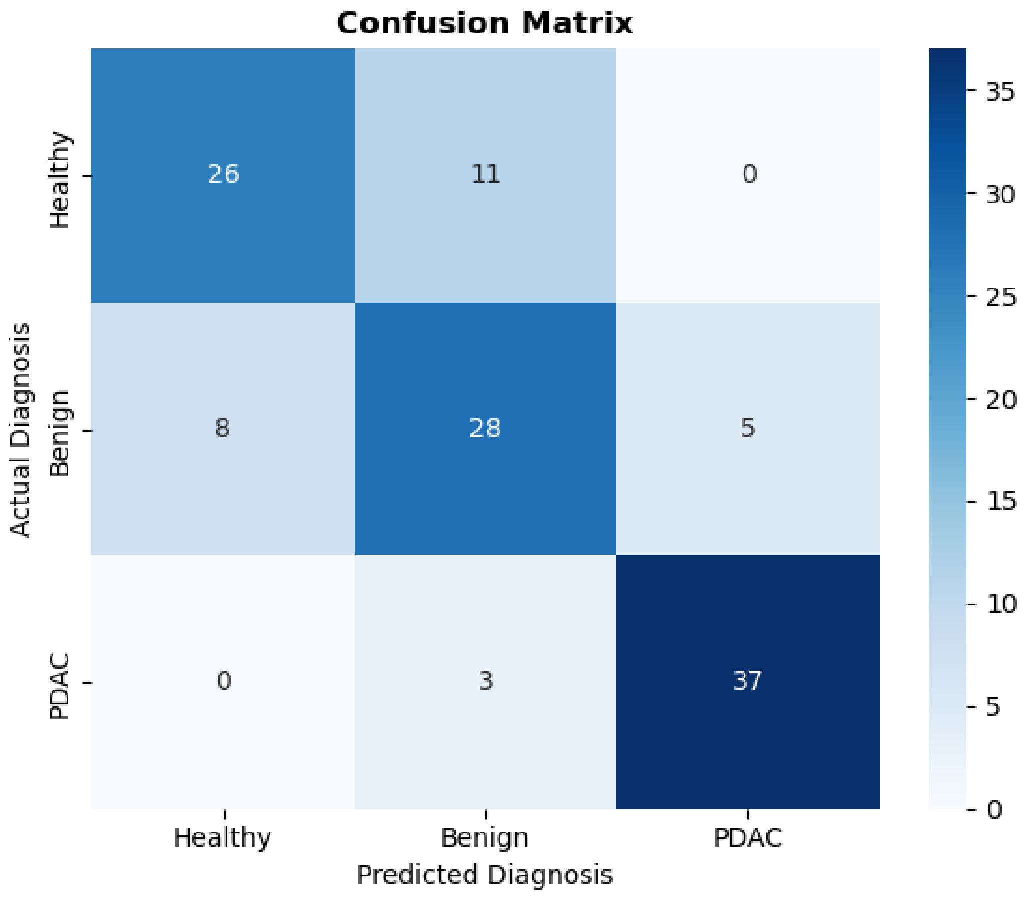
[168]:
```
# From the test above, we see that all selected features have positive␣
 ↪contribution to enhance the model performance.
# All features are remained in this final test.

rf2 = RandomForestClassifier(n_estimators=200, max_depth=15,␣
 ↪min_samples_split=5, min_samples_leaf=2,
                             class_weight='balanced', bootstrap=True,␣
 ↪random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Use RobustScaler for skewed data
scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

rf2.fit(X_train_scaled, y_train)

y_pred = rf2.predict(X_test_scaled)
y_pred_proba = rf2.predict_proba(X_test_scaled)
y_train_pred_proba = rf2.predict_proba(X_train_scaled)

# Calculate metrics
test_accuracy = accuracy_score(y_test, y_pred)
test_loss = log_loss(y_test, y_pred_proba)
train_loss = log_loss(y_train, y_train_pred_proba)
```

```
train_accuracy = accuracy_score(y_train, rf2.predict(X_train_scaled))

# Display results
print(f"Training accuracy: {train_accuracy*100:.2f}%")
print(f"Training loss: {train_loss:.4f}")
print(f"Test accuracy: {test_accuracy*100:.2f}%")
print(f"Test loss: {test_loss:.4f}")
```

```
Training accuracy: 99.58%
Training loss: 0.2943
Test accuracy: 77.12%
Test loss: 0.5688
```

[169]:
```
#report
report = classification_report(y_test, y_pred, target_names=['Healthy',␣
 ↪'Benign', 'PDAC'])
print(report)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
targets = ['Healthy', 'Benign', 'PDAC']

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=targets, yticklabels=targets)
plt.title(f'Confusion Matrix', fontsize=12, fontweight='bold')
plt.ylabel('Actual Diagnosis', fontsize=10)
plt.xlabel('Predicted Diagnosis', fontsize=10)
plt.tight_layout()
plt.show()
```

```
              precision    recall  f1-score   support

     Healthy       0.76      0.70      0.73        37
      Benign       0.67      0.68      0.67        41
        PDAC       0.88      0.93      0.90        40

    accuracy                           0.77       118
   macro avg       0.77      0.77      0.77       118
weighted avg       0.77      0.77      0.77       118
```



[150]:
```
display(HTML("""<div style='
    background: white;
    padding: 10px;
    color: black;
    font-weight: bold;
    font-size: 24px;
    text-align: left;'>
Section 3: Dicussion: Performance with binary classification task.
</div>"""))
```

```
<IPython.core.display.HTML object>
```

[114]:
```
# This section perform binary classification task (0: non-cancerous, 1:␣
 ↪cancerous) with Random Forest
# to further investigate the implementation of ML methods in this problem

# 0: healthy or benign, 1: PDAC
y_binary = y.map({1: 0, 2: 0, 3: 1})
```

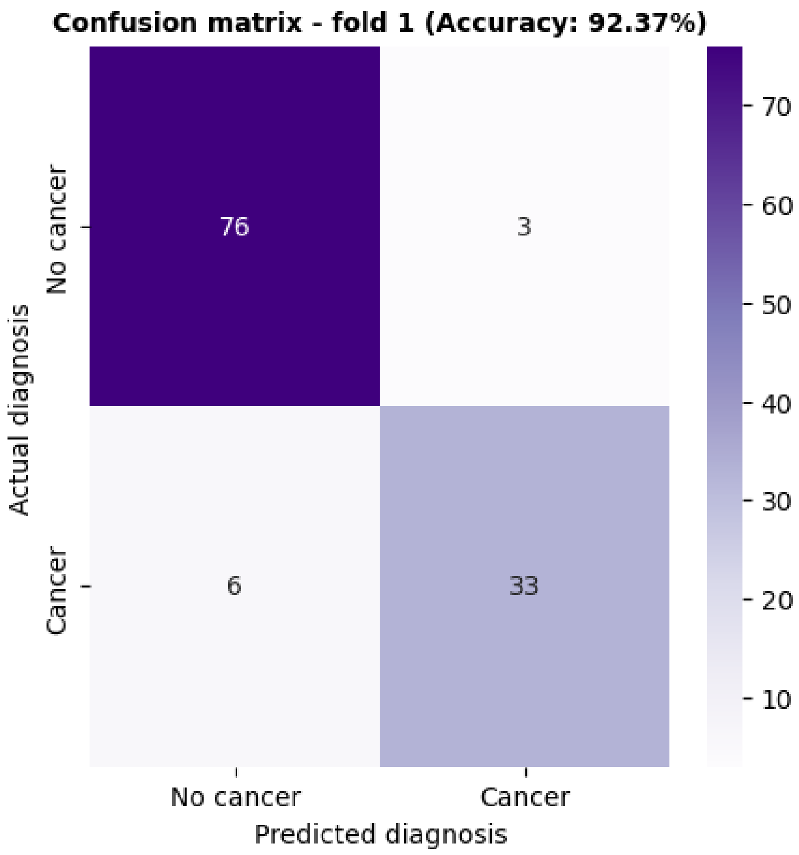
```
print('Categorical data:')
print(y.value_counts())
print('Binary data:')
print(y_binary.value_counts())
```

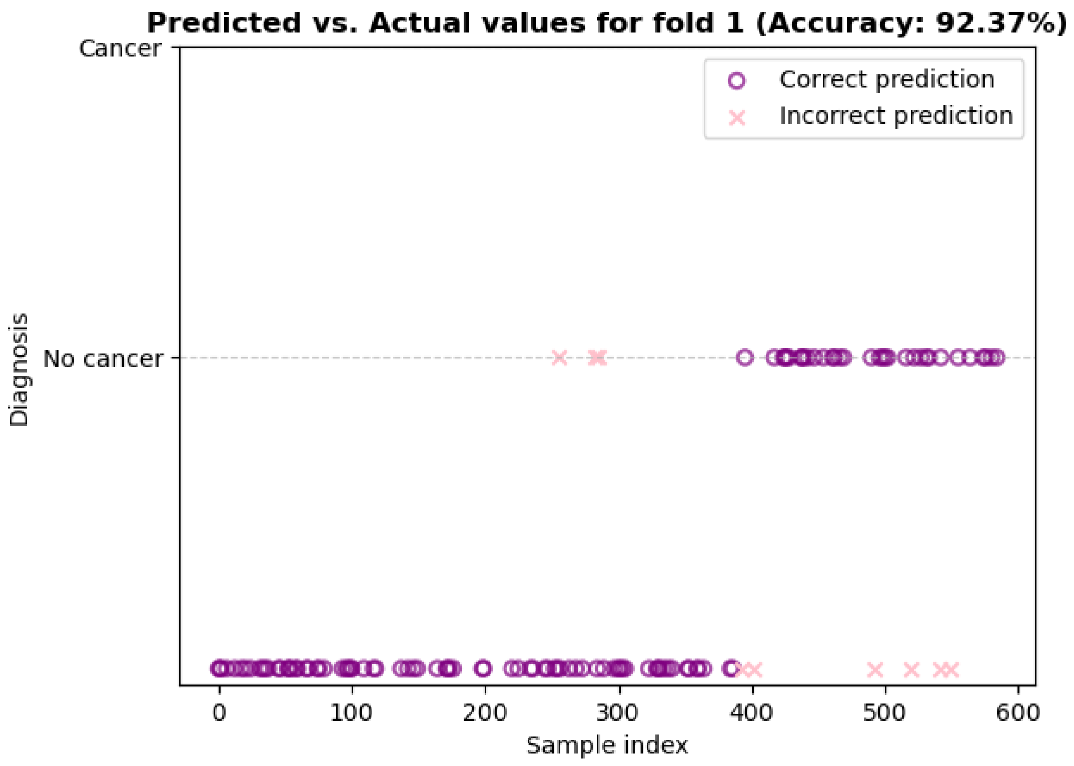
```
Categorical data:
diagnosis
2    208
3    199
1    183
Name: count, dtype: int64
Binary data:
diagnosis
0    391
1    199
Name: count, dtype: int64
```

[121]:
```
accuracy_scores_rf2 = []

for i, (train_index, test_index) in enumerate(skf.split(X, y_binary)):
    # training
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y_binary.iloc[train_index], y_binary.iloc[test_index]

    rf.fit(X_train, y_train)

    y_pred = rf.predict(X_test)

    # evaluating
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores_rf2.append(accuracy)

    # result
    print(f"Fold {i+1}: Accuracy = {accuracy*100:.2f}%")

    targets = ['No cancer', 'Cancer']

    # plot confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(5, 5))
    sns.heatmap(conf_matrix, annot=True, cmap='Purples',
            xticklabels=targets, yticklabels=targets)
    plt.title(f'Confusion matrix - fold {i+1} (Accuracy: {accuracy:.2%})',␣
 ↪fontsize=10, fontweight='bold')
    plt.ylabel('Actual diagnosis', fontsize=10)
```

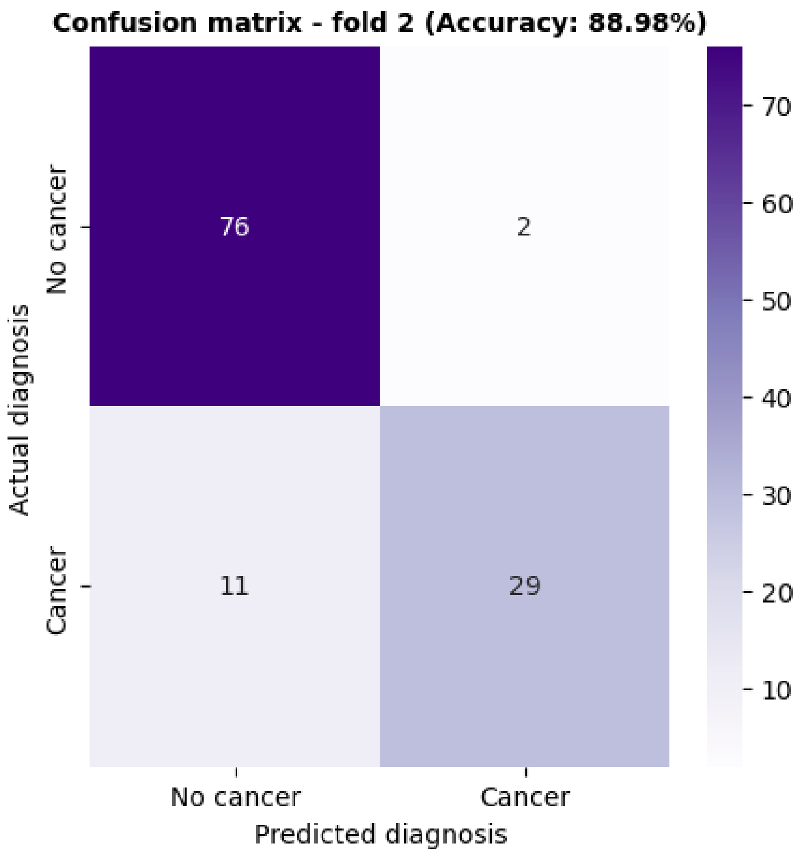
```python
    plt.xlabel('Predicted diagnosis', fontsize=10)
    plt.show()

    # plot comparison between predicted and real values for model with larger than 90% accuracy score.
    if accuracy > 0.90:

        results_df = pd.DataFrame({
            'Actual': y_test,
            'Predicted': y_pred,
            'Index': y_test.index
        })

        correct_preds = results_df[results_df['Actual'] == results_df['Predicted']]
        incorrect_preds = results_df[results_df['Actual'] != results_df['Predicted']]

        plt.figure()

        plt.scatter(correct_preds['Index'], correct_preds['Actual'],
                    alpha=0.7, facecolors='none', edgecolors='purple',
                    linewidths=1.5, label='Correct prediction', marker='o')
        plt.scatter(incorrect_preds['Index'], incorrect_preds['Predicted'],
                    color='pink', alpha=1.0, label='Incorrect prediction', marker='x')

        plt.yticks([1, 2], targets)
        plt.ylabel('Diagnosis', fontsize=10)
        plt.xlabel('Sample index', fontsize=10)
        plt.title(f'Predicted vs. Actual values for fold {i+1} (Accuracy: {accuracy:.2%})', fontsize=12, fontweight='bold')
        plt.legend()
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.show()

print(f"Average accuracy: {np.mean(accuracy_scores_rf2)*100:.2f}%")
```

```
Fold 1: Accuracy = 92.37%
```

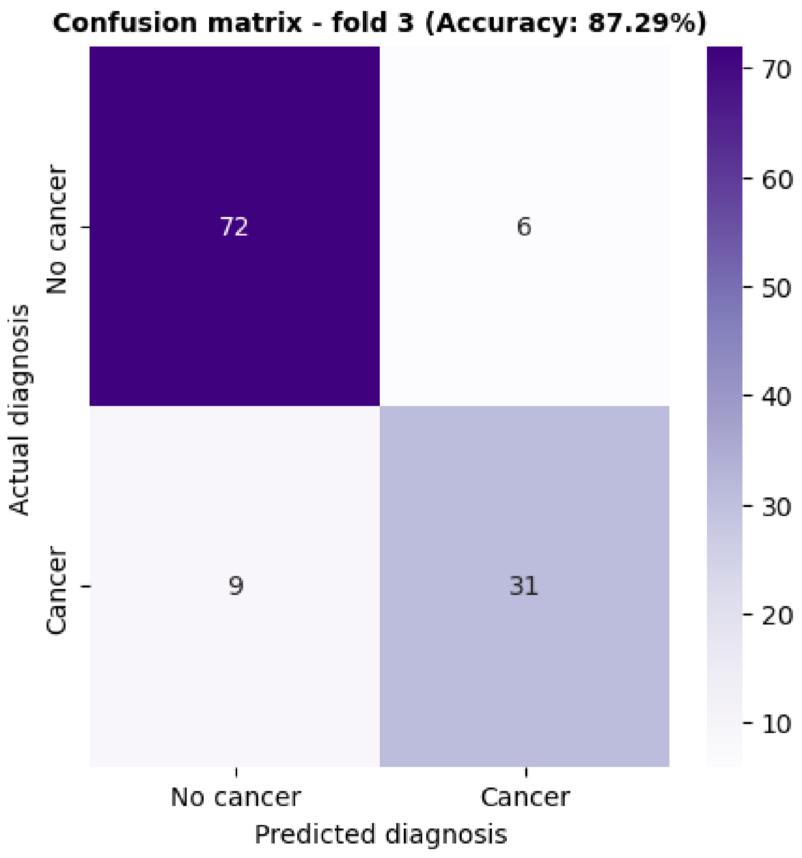

Confusion matrix - fold 1 (Accuracy: 92.37%)
Actual diagnosis
No cancer
Cancer
76
3
6
33
No cancer
Cancer
Predicted diagnosis
70
60
50
40
30
20
10



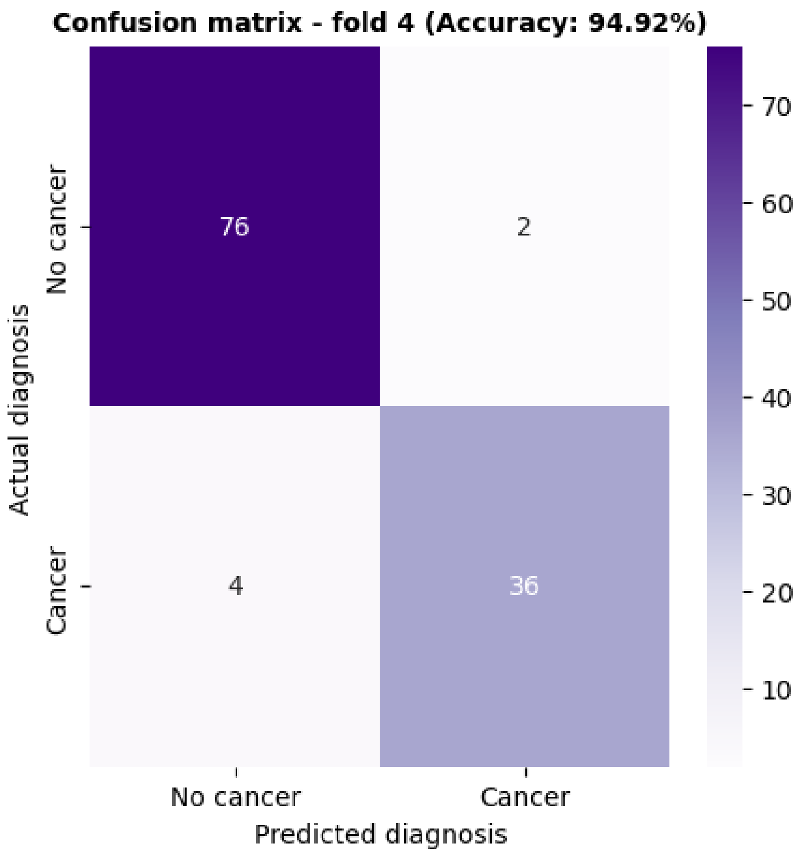
```
Fold 2: Accuracy = 88.98%
```

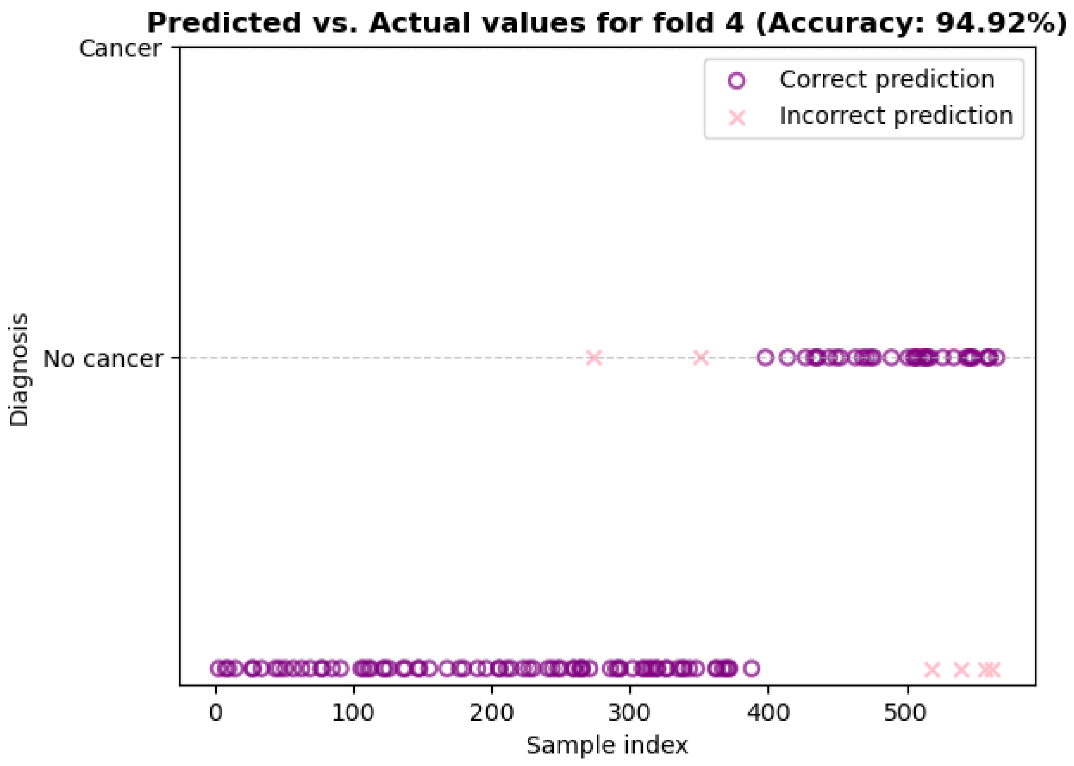


```
Fold 3: Accuracy = 87.29%
```

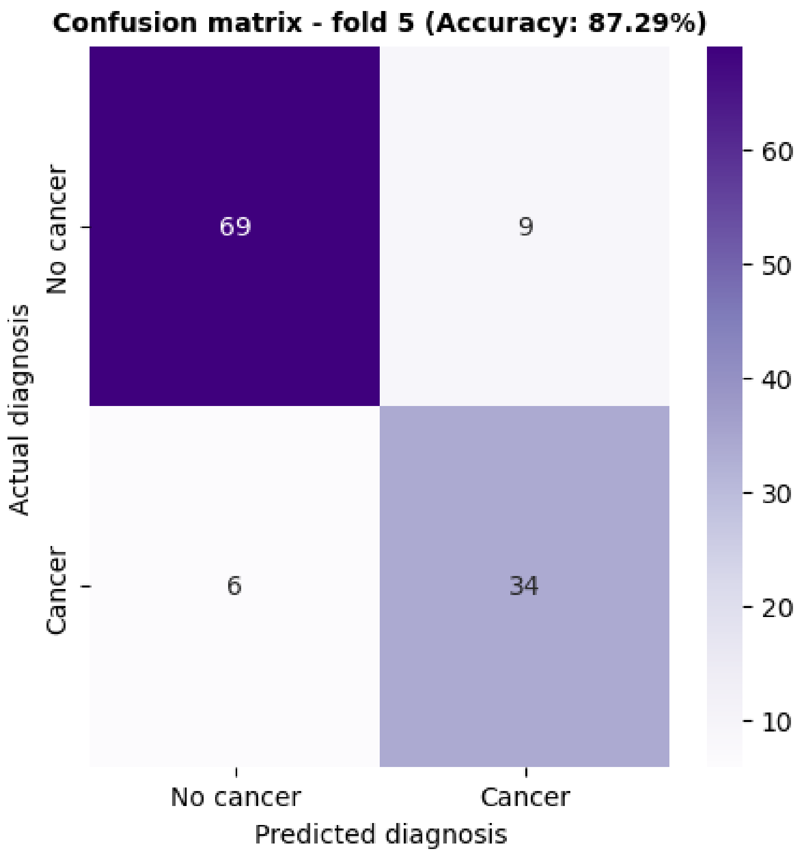


```
Fold 4: Accuracy = 94.92%
```


Confusion matrix - fold 4 (Accuracy: 94.92%)
Actual diagnosis
No cancer
Cancer
76
2
4
36
No cancer
Cancer
Predicted diagnosis
70
60
50
40
30
20
10



```
Fold 5: Accuracy = 87.29%
```



```
Average accuracy: 90.17%
```