Introduction to Machine Learning
Dr. Öğr. Üyesi Özkan Ufuk NALBANTOĞLU

Midterm Project

Diagnosis of Cancer Using Blood Microbiome Data

—

Dilara KIZILKAYA - 1030510143

Introduction

In this project I aimed to find the highest possible correct classification scores for the Cancer data using ML techniques. Throughout the project, I used Random Forest, XGBOOST and LightGBM models.

Project Steps

1. Preprocessing
2. Training / Testing
3. Evaluating the Results

1. Preprocessing

First and foremost I imported the necessary libraries in order to start coding. For the preprocessing step I only imported Pandas.

```
In [1]: import pandas as pd
```

Then using Pandas library's read_csv() function, I read the data and labels.

```
In [2]: labels_df = pd.read_csv('labels.csv')
        data_df = pd.read_csv('data.csv')
```

```
In [3]: labels_df
Out[3]:
```

	Sample	disease_type
0	s12859	colon cancer
1	s12878	colon cancer
2	s12857	colon cancer
3	s12882	colon cancer
4	s12871	colon cancer
...	...	...
350	s3971	prosrtate cancer
351	s3942	prosrtate cancer
352	s3975	prosrtate cancer
353	s3926	prosrtate cancer
354	s3952	prosrtate cancer

355 rows × 2 columns

```
In [4]: data_df
```

0	s12859	0
1	s12878	0
2	s12857	2
3	s12882	4
4	s12871	2
...	...	...
350	s3971	2
351	s3942	4
352	s3975	2
353	s3926	0
354	s3952	6

355 rows × 1837 columns

As seen on the picture above, our data.csv has 355 rows and 1837 columns which indicate the blood samples of 355 people with 4 most common cancer types (Colon cancer, breast cancer, lung cancer, and prostate cancer). Addition to the data.csv, we can see that the labels.csv has 355 rows and columns which shows which patient has which cancer type.

Then I first added a ‘Sample’ column to the data.csv in order to have the same ‘Sample’ column as some kind of an ID with the labels.csv, this will be very helpful while merging these two datasets.

```
In [22]: olderName = data_df.columns[0]
         newName = labels_df.columns[0]

         data_df.rename(columns={olderName: newName}, inplace=True)

         data_df
```

Out[22]:

	Sample	k__Viruses.f__Phycodnaviridae.g__Prasinovirus	k__Viruses.o__Caudovirales.f__Siphovi
0	s12859	0	
1	s12878	0	
2	s12857	2	
3	s12882	4	
4	s12871	2	
...	...	...	
350	s3971	2	
351	s3942	4	
352	s3975	2	
353	s3926	0	
354	s3952	6	

355 rows × 1837 columns

To be able to see whether any of the columns are blank places or entirely zeros, I first merged these two datasets then used the .isnull() and .sum() functions.

```
In [27]: merged_df.isnull().sum()
```

```
Out[27]: Sample
         0
         disease_type
         0
         k__Viruses.f__Phycodnaviridae.g__Prasinovirus
         0
         k__Viruses.o__Caudovirales.f__Siphoviridae.g__Sfi1unalikevirus
         0
         k__Viruses.o__Herpesvirales.f__Herpesviridae.g__Simplexvirus
         0

         ..
         k__Bacteria.p__Firmicutes.c__Clostridia.o__Clostridiales.f__Clostri
         0
         k__Archaea.p__Crenarchaeota.c__Thermoprotei.o__Desulfurococcales.f_
         0
         k__Bacteria.p__Proteobacteria.c__Betaproteobacteria.o__Neisseriales
         0
         k__Bacteria.p__Deferribacteres.c__Deferribacteres.o__Deferribacteral
```

Then I added this simple code below to check whether if there are any columns with entirely zero values.

```
In [28]: all_zeros_columns = []

         for column in merged_df.columns:
             if (merged_df[column] == 0).all():
                 all_zeros_columns.append(column)

         if len(all_zeros_columns) > 0:
             print("Columns with all zeros:")
             for column_name in all_zeros_columns:
                 print(column_name)
         else:
             print("No columns have all zeros.")
```

```
No columns have all zeros.
```

As can be seen from the output of the code, there is no columns which has entirely zero as it's data.

2. Training and Testing

As I did with the preprocessing step, I first important all the necessary libraries before I start training and testing the models.

```
In [1]: from sklearn.preprocessing import LabelEncoder
        import pandas as pd
        import xgboost
        from xgboost import XGBClassifier
        from sklearn.ensemble import RandomForestClassifier
        import lightgbm as lgb
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, ConfusionMatrixDisplay
        from sklearn.metrics import classification_report
```

All the libraries and modules I used can be seen above.

After the importing step, I read my csv files to X and y. The data and labels are given separately so I didn't use the dataset I created during the preprocessing step. With the given separated csv files, I was able to create X and y easily.

The label names were string values so I encoded them to integer values using a LabelEncoder()

```
In [3]: label_encoder = LabelEncoder()

        y_encoded = label_encoder.fit_transform(y)

        y_encoded

        /home/dilara/anaconda3/lib/python3.9/site-packages/sklearn/preprocessing/_label.py:11
        umn-vector y was passed when a 1d array was expected. Please change the shape of y to
        ng ravel().
          y = column_or_1d(y, warn=True)

Out[3]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
               2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3,
               3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
               3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
               3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
```

Then I created the train and test splits. I used 30% of the data as the test set and %70 of the data as the training set.

```
In [5]: X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size = 0.3, random_state=42)
```

Now everything is ready to train the models.

1. XGBOOST

```
In [6]: xgbc = XGBClassifier(learning_rate=0.1, objective='multi:softmax', num_class = 4)

In [7]: xgbc.fit(X_train, y_train)

Out[7]: XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
                      colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
                      early_stopping_rounds=None, enable_categorical=False,
                      eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
                      grow_policy='depthwise', importance_type=None,
                      interaction_constraints='', learning_rate=0.1, max_bin=256,
                      max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
                      max_depth=6, max_leaves=0, min_child_weight=1, missing=nan,
                      monotone_constraints='()', n_estimators=100, n_jobs=0,
                      num_class=4, num_parallel_tree=1, objective='multi:softmax', ...)

In [8]: y_pred_xgb = xgbc.predict(X_test)
```

For XGBOOST parameters, I set the learning_rate to 0.1 and objective as ‘multi:softmax’ as well as the num_class to 4 since we are dealing with a multi-class classification problem.

Normally for the XGBOOST we should’ve turn out datas into a DMatrix but now with the latest XGBOOST version we can give our data to the model directly without turning it into a DMatrix.

2. Random Forest

```
In [29]: rfc = RandomForestClassifier(n_estimators=100,max_depth=None)

In [30]: rfc.fit(X_train, y_train)

Out[30]: RandomForestClassifier()

In [31]: y_pred_rf = rfc.predict(X_test)
```

For the Random Forest parameters, I set n_estimators to 100 and max_depth to None.

3. LightGBM

```
In [22]: lgbc = lgb.LGBMClassifier(max_depth = -3, learning_rate = 0.02, objective = 'multiclass', metric = 'multi_logloss',
         num_classes = 4, boosting_type = 'gbdt')

In [23]: lgbc.fit(X_train, y_train)

Out[23]: LGBMClassifier(learning_rate=0.02, max_depth=-3, metric='multi_logloss',
                        num_classes=4, objective='multiclass')

In [24]: y_pred_lgb = lgbc.predict(X_test)
```

For the LightGBM parameters, I set max_depth as -3, learning_rate as 0.02, objective as 'multiclass', metric 'multi_logloss', num_classes as 4 and boosting_type as 'gbdt' since out problem is multiclass classification.

3. Evaluating the Results

Before evaluating the models, I wrote this simple code to calculate Sensitivity and Specifity.

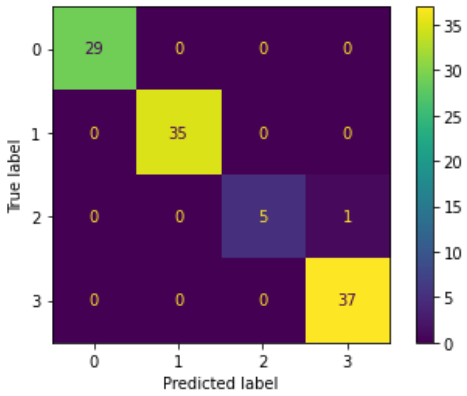
```
In [50]: def specifity_sensitivity(cm):

             class_num = len(cm)
             specifity = {}
             sensitivity = {}
             FP=0
             TN=0
             FN=0

             for i in range(class_num):

                 TP = cm[i][i]

                 for j in range(class_num):
                     if i != j:
                         FP +=cm[j][i]
                 for j in range(class_num):
                     for k in range(class_num):
                         if j!=i and k!=i:
                             TN +=cm[j][k]
                 for j in range(class_num):
                     if j!=i:
                         FN +=cm[i][j]

                 specifity[i] = TN / (TN + FP)
                 sensitivity[i] = TP / (TP + FN)

             return specifity, sensitivity
```

This function takes a confusion matrix as it's input then using len() function it calculates the length of the matrix. Then I created two dictionaries called specificity and sensitivity. After those in the for loops I calculate the False Positive, False Negative, True Positive and True

Negative from the confusion matrix. Then I use these values to calculate specificity and sensitivity.

1. XGBOOST

```
In [10]: accuracy_xgb
Out[10]: 0.9906542056074766

In [11]: xgb_cm = confusion_matrix(y_test, y_pred_xgb)

         cm_display = ConfusionMatrixDisplay(xgb_cm).plot()
```

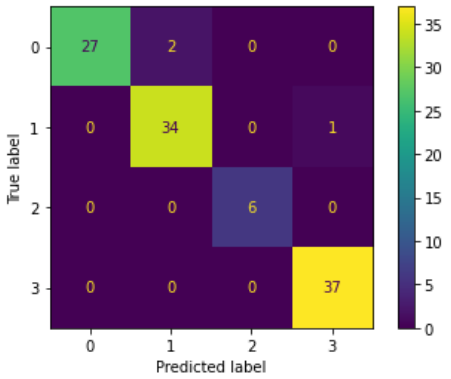



```
In [14]: print(classification_report(y_test, y_pred_xgb, target_names=target_names))
                   precision    recall  f1-score   support

     Colon cancer       1.00      1.00      1.00        29
   Breast cancer        1.00      1.00      1.00        35
     Lung cancer        1.00      0.83      0.91         6
Prostate cancer         0.97      1.00      0.99        37

         accuracy                           0.99       107
        macro avg       0.99      0.96      0.97       107
     weighted avg       0.99      0.99      0.99       107


In [51]: specifity_sensitivity(xgb_cm)
Out[51]: ({0: 1.0, 1: 1.0, 2: 1.0, 3: 0.9968847352024922},
          {0: 1.0, 1: 1.0, 2: 0.8333333333333334, 3: 0.9736842105263158})
```

2. Random Forest

```
In [33]: accuracy_rf
Out[33]: 0.9719626168224299
```

```
In [34]: rf_cm = confusion_matrix(y_test, y_pred_rf)

         cm_display = ConfusionMatrixDisplay(rf_cm).plot()
```




```
In [35]: print(classification_report(y_test, y_pred_rf, target_names=target_names))
```

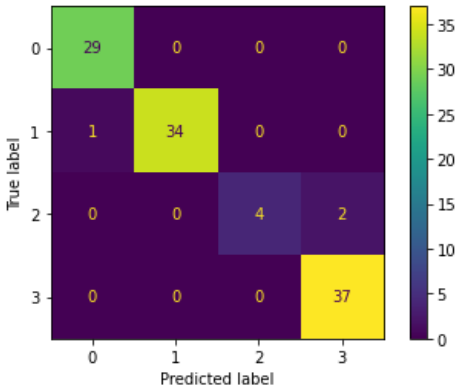
	precision	recall	f1-score	support
Colon cancer	1.00	0.93	0.96	29
Breast cancer	0.94	0.97	0.96	35
Lung cancer	1.00	1.00	1.00	6
Prostate cancer	0.97	1.00	0.99	37
accuracy			0.97	107
macro avg	0.98	0.98	0.98	107
weighted avg	0.97	0.97	0.97	107

```
In [52]: specifity_sensitivity(rf_cm)
Out[52]: ({0: 1.0, 1: 0.9866666666666667, 2: 0.9920318725099602, 3: 0.9906542056074766},
          {0: 0.9310344827586207,
           1: 0.918918918918919,
           2: 0.6666666666666666,
           3: 0.925})
```

3. LightGBM

```
In [26]: accuracy_lgb
Out[26]: 0.9719626168224299
```

```
In [27]: lgb_cm = confusion_matrix(y_test, y_pred_lgb)

         cm_display = ConfusionMatrixDisplay(lgb_cm).plot()
```




```
In [28]: print(classification_report(y_test, y_pred_lgb, target_names=target_names))
```

	precision	recall	f1-score	support
Colon cancer	0.97	1.00	0.98	29
Breast cancer	1.00	0.97	0.99	35
Lung cancer	1.00	0.67	0.80	6
Prostate cancer	0.95	1.00	0.97	37
accuracy			0.97	107
macro avg	0.98	0.91	0.94	107
weighted avg	0.97	0.97	0.97	107

```
In [53]: specify_sensitivity(lgb_cm)
Out[53]: ({0: 0.9871794871794872,
           1: 0.9933333333333333,
           2: 0.9960159362549801,
           3: 0.9906542056074766},
          {0: 1.0, 1: 0.9714285714285714, 2: 0.5714285714285714, 3: 0.925})
```

As can be seen from the code outputs above, the best accuracy score was with the XGBOOST model.