T.C.
ERCİYES ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

Diagnosis of Cancer Using Blood Microbiome Data

Hazırlayan
Duygu Gözde KAYABAŞI
1030510338

Bilgisayar Mühendisliği
Introduction to Machine Learning
Midterm Project

Mayıs
2024
KAYSERİ

1. The Goal for this midterm project is :

With this project it is expected to have the highest possible correct classification scores (see performance measures below). 4 different classifications (each cancer vs. others) are going to be performed.

For this we have to following this rules:

1. Import the data

   Importing the data is the initial step in the machine learning workflow, where raw data is brought into the analytical environment for processing and analysis. This involves reading data from various sources such as CSV files, databases, spreadsheets, or online repositories into a format that can be used by machine learning libraries. In Python, popular libraries such as Pandas facilitate this process with functions like read_csv, read_excel, or database connectors that streamline data importation.

   Briefly, we are using the 'pandas' library that belongs to python . According to the file extension, there are specific functions. Right now, we will use `pandas.read_csv()` because our file extension is `.csv` .

2. Clean the data

   Cleaning the data is a critical step in the data preprocessing phase of machine learning, essential for ensuring the quality and reliability of the model's predictions. This process involves several tasks aimed at correcting or removing inaccuracies and inconsistencies within the dataset. Common steps include handling missing values by either imputing them with statistical measures like the mean or median or by removing the affected rows or columns altogether. Duplicate records are also detected and eliminated to prevent biased results. By thoroughly cleaning the data, we ensure that the model is trained on accurate, consistent, and relevant information, leading to improved model performance and more reliable insights.

   If there are duplicates and empty columns , we have to get rid of them. There are a few functions that we can use for this situation like `drop_duplicates()` , `drop()` . First of all, we have to drop the first column in `"data.csv"` but

there is no header , I change that using Microsoft excel. In other ways , this code snippets will work as well .

```
data.columns.values[0] = 'Sample'
features = data.drop('Sample', axis = 1)
```

3. Split the data into training/test sets

Splitting data into training and test sets is a fundamental step in the machine learning workflow, crucial for assessing the performance and generalizability of a model. This process involves dividing the available dataset into two distinct subsets: the training set and the test set. The training set, typically comprising around 70-80% of the total data, is used to train the model, allowing it to learn the underlying patterns and relationships within the data. The remaining 20-30% forms the test set, which is used to evaluate the model's performance on unseen data. This separation is essential because it simulates how the model will perform on real-world data, helping to identify issues such as overfitting, where the model performs well on the training data but poorly on new, unseen data. Properly splitting the data ensures that the evaluation of the model's accuracy, precision, and other performance metrics reflects its true predictive power and not just its ability to memorize the training data. This method is foundational in developing reliable and effective machine learning models.

In this step , we just split `"data.csv"` and `"labels.csv"` in a way, training 75% and testing 25% of all data . We used these libraries by importing.
`from sklearn.model_selection import train_test_split.`

4. Create a model

To create a model , we will use Random Forest and Gradient Boosting in this project. Those are two powerful ensemble learning algorithms widely used in machine learning for their robustness and predictive accuracy.

Random Forest, an ensemble of decision trees, operates by constructing multiple decision trees during training and outputting the mode of the classes for classification tasks or the mean prediction for regression tasks. This algorithm is particularly effective in reducing overfitting by averaging multiple trees, thus lowering the variance of the model. To create a Random Forest model, one would typically start by splitting the dataset into training and testing sets, then use the

training set to build a multitude of decision trees, each trained on a random subset of the data features and samples.

On the other hand, Gradient Boosting builds trees sequentially, where each new tree corrects the errors of the previous ones. By focusing on the mistakes made by prior models and adding weak learners to create a strong predictive model, Gradient Boosting achieves high accuracy, albeit at the cost of being more computationally intensive than Random Forest. Constructing a Gradient Boosting model involves iteratively adding decision trees, each trained to minimize the residual errors from the combined prediction of all previous trees.

Implementing these models using libraries such as Scikit-learn in Python

```
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
```

And

```
random_forest = RandomForestClassifier(n_estimators = 100,
max_depth = 3)
gradient_boosting = GradientBoostingClassifier(n_estimators =
100, max_depth = 3)
```

5. Train the model

Training a machine learning model is a critical process that involves teaching the algorithm to recognize patterns in data and make accurate predictions. This process begins with gathering a comprehensive dataset, which is then divided into training and testing sets (we already did that) to evaluate the model's performance. During training, the model iteratively adjusts its parameters by minimizing the error between its predictions and the actual outcomes, a process driven by algorithms like gradient descent. Various techniques, such as cross-validation, are employed to ensure the model generalizes well to unseen data and to prevent overfitting. Feature engineering, which includes selecting and transforming relevant variables, is also crucial for improving model accuracy. As the model learns from the training data, it continually updates its internal parameters to optimize performance. Once trained, the model's effectiveness is assessed using the testing set, providing insights into its predictive capabilities and areas for improvement. This iterative cycle of training, validation, and refinement is fundamental to developing robust and reliable machine learning models capable of making accurate predictions in real-world applications.

```
random_forest.fit(X_train ,y_train)
gradient_boosting.fit(X_train, y_train)
```

6. Make predictions

Making predictions is the final step in the machine learning process, where the trained model is applied to new, unseen data to generate outcomes based on the learned patterns and relationships. The accuracy and reliability of these predictions depend heavily on the quality of the data and the effectiveness of the training process.

```
accuracy_random_forest = round(random_forest.score(X_train,
y_train) * 100, 2)
accuracy_random_forest
prediction_from_random_forest = random_forest.predict(X_test)
accuracy_score_random_forest =
accuracy_score(prediction_from_random_forest, y_test)

accuracy_gradient = round(gradient_boosting.score(X_train,
y_train) * 100, 2)
prediction_from_gradient_boost =
gradient_boosting.predict(X_test)
accuracy_score_gradient_boost =
accuracy_score(prediction_from_gradient_boost, y_test)
```

7. Find Sensitivity and Specificity

Sensitivity (also known as recall or true positive rate) and specificity are important metrics used to evaluate the performance of a classification model. They can be derived from a confusion matrix, which shows the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions.

Explanation:

True Positives (TP): The model correctly predicts the positive class.

True Negatives (TN): The model correctly predicts the negative class.

False Positives (FP): The model incorrectly predicts the positive class (Type I error).

False Negatives (FN): The model incorrectly predicts the negative class (Type II error).

Formulae:

Sensitivity (Recall/True Positive Rate): Sensitivity = $\frac{TP}{TP+FN}$

Specificity (True Negative Rate): Specificity = $\frac{TN}{TN+FP}$

```
true_positive , false_positive , false_negative , true_negative =
confusion_matrix(y_test, prediction_from_random_forest)
sensitivity_random_forest = numpy.empty(0)
specificity_random_forest = numpy.empty(0)

sensitivity_random_forest = numpy.append(sensitivity_random_forest,
numpy.array([true_positive / (true_positive + false_negative)]))
specificity_random_forest = numpy.append(specificity_random_forest,
numpy.array([true_negative / (true_negative + false_positive)]))

true_negative, false_positive, false_negative, true_positive =
confusion_matrix(y_test, prediction_from_gradient_boost)
sensitivity_gradient_boost = numpy.empty(0)
specificity_gradient_boost = numpy.empty(0)

sensitivity_gradient_boost = numpy.append(sensitivity_gradient_boost,
numpy.array([true_positive / (true_positive + false_negative)]))
specificity_gradient_boost = numpy.append(specificity_gradient_boost,
numpy.array([true_negative / (true_negative + false_positive)]))
```

8. Evaluate and improve

Evaluating and improving a machine learning model is a critical iterative process aimed at ensuring the model's accuracy, reliability, and generalizability. These metrics help quantify the model's performance on both the training and test datasets, highlighting areas where the model excels and where it may fall short.

Cross-validation, where the data is split into multiple training and validation sets, is often employed to ensure that the evaluation is robust and not biased by a single train-test split.

Improving the model involves various techniques to enhance its performance. This can include tuning hyperparameters, which are settings that govern the training process, using methods such as grid search or random search to find the optimal

values. Feature engineering is another crucial aspect, where new features are created, and irrelevant ones are removed to improve the model's predictive power. The cycle of evaluation and improvement continues until the model achieves satisfactory performance, ensuring it is well-suited for deployment in real-world applications.

2. Coding :

```
import numpy
import pandas
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

numpy.seterr(divide='ignore', invalid='ignore')
data = pandas.read_csv("data.csv")
#data.describe()
labels = pandas.read_csv("labels.csv")
#labels.describe()
data.drop_duplicates()
labels.drop_duplicates()
#data.columns.values[0] = 'Sample' I ALREADY DID THAT
#1836 different microorganisms appear as features.
input = data.drop('Sample', axis = 1)
#input
output = labels['disease_type']
#output
#train %75 test %25
X_train, X_test, y_train, y_test = train_test_split(input, output,
test_size = 0.25, random_state= 42)
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

random_forest = RandomForestClassifier(n_estimators = 100, max_depth =
3)
random_forest.fit(X_train ,y_train)
accuracy_random_forest = round(random_forest.score(X_train, y_train) *
100, 2)
accuracy_random_forest

prediction_from_random_forest = random_forest.predict(X_test)
```

```
accuracy_score_random_forest =
accuracy_score(prediction_from_random_forest, y_test)

#correct_first_class , total_first_class , correct_second_class ,
total_second_class
true_positive , false_positive , false_negative , true_negative =
confusion_matrix(y_test, prediction_from_random_forest)
sensitivity_random_forest = numpy.empty(0)
specificity_random_forest = numpy.empty(0)

sensitivity_random_forest = numpy.append(sensitivity_random_forest,
numpy.array([true_positive / (true_positive + false_negative)]))
specificity_random_forest = numpy.append(specificity_random_forest,
numpy.array([true_negative / (true_negative + false_positive)]))

print('Training Features Shape:', X_train.shape)
print('Training Labels Shape:', y_train.shape)
print('Testing Features Shape:', X_test.shape)
print('Testing Labels Shape:', y_test.shape)

print("The Sensitivity of Random Forest Algorithm is :",
sensitivity_random_forest)
print("The Specificity for Random Forest Algorithm is :",
specificity_random_forest)

print("The Accuracy Score of Random Forest Algorithm is :")
print(round(accuracy_random_forest,2,), "%")

gradient_boosting = GradientBoostingClassifier(n_estimators = 100,
max_depth = 3)

gradient_boosting.fit(X_train, y_train)

accuracy_gradient = round(gradient_boosting.score(X_train, y_train) *
100, 2)

prediction_from_gradient_boost = gradient_boosting.predict(X_test)
accuracy_score_gradient_boost =
accuracy_score(prediction_from_gradient_boost, y_test)

true_negative, false_positive, false_negative, true_positive =
confusion_matrix(y_test, prediction_from_gradient_boost)
sensitivity_gradient_boost = numpy.empty(0)
```

```
specificity_gradient_boost = numpy.empty(0)

sensitivity_gradient_boost = numpy.append(sensitivity_gradient_boost,
numpy.array([true_positive / (true_positive + false_negative)]))
specificity_gradient_boost = numpy.append(specificity_gradient_boost,
numpy.array([true_negative / (true_negative + false_positive)]))

print("The Sensitivity of Gradient Boosted Tree Algorithm is :",
sensitivity_gradient_boost)
print("The Specificity of Gradient Boosted Tree Algorithm is :",
specificity_gradient_boost)

print("The Accuracy Score of Gradient Boosted Tree Algorithm is :")
print(round(accuracy_gradient,2,), "%")

#Output

Training Features Shape: (266, 1836)
Training Labels Shape: (266,)
Testing Features Shape: (89, 1836)
Testing Labels Shape: (89,)
The Sensitivity of Random Forest Algorithm is : [1. 1. 0. 1.]
The Specificity for Random Forest Algorithm is : [       nan 0.
1.         0.93548387]
The Accuracy Score of Random Forest Algorithm is :
94.74 %
The Sensitivity of Gradient Boosted Tree Algorithm is : [nan nan  0.
1.]
The Specificity of Gradient Boosted Tree Algorithm is : [ 1.  0. nan
nan]
The Accuracy Score of Gradient Boosted Tree Algorithm is :
100.0 %
```

3. **Source:**

Matrix Data Structure - GeeksforGeeks

Python Machine Learning Tutorial (Data Science)

StatQuest: Random Forests Part 2: Missing data and clustering

- A Gentle Introduction to Machine Learning
- Machine Learning Fundamentals: Cross Validation
- Machine Learning Fundamentals: The Confusion Matrix
- StatQuest: Logistic Regression
- StatQuest: Random Forests Part 1 - Building, Using and Evaluating
- Gradient Boost Part 1 (of 4): Regression Main Ideas
- Support Vector Machines Part 1 (of 3): Main Ideas!!!
- Gradient Boosting Classification with Python
- How to implement Random Forest from scratch with Python
- Random Forest Classification | Machine Learning | Python
- Machine Learning Tutorial Python - 11 Random Forest

confusion_matrix — scikit-learn 1.5.0 documentation

numpy.array — NumPy v1.26 Manual

- How to Build and Interpret Confusion Matrix Using Python & Sklearn

1.11. Ensembles: Gradient boosting, random forests, bagging, voting, stacking — scikit-learn 1.5.0 documentation

train_test_split — scikit-learn 1.5.0 documentation

Glossary of Common Terms and API Elements — scikit-learn 1.5.0 documentation

General functions — pandas 2.2.2 documentation