PREDICTING THE PROBABILITY OF A STROKE BASED ON BIOLOGICAL AND SOCIOECONOMIC FEATURES USING CLASSIFICATION METHODS

CS-C3240 Machine Learning Project

1. INTRODUCTION

The goal of this project is to try different classification methods to predict a person's probability of getting a stroke based on their biological and socioeconomic features. The methods chosen for this project are **logistic regression** and **decision tree classification**. This kind of problem could be used in healthcare when predicting one's probability of getting a stroke. Section 2 discusses the problem formulation. It defines the data points, features, and labels. Section 3 discusses the methods used in this project. It starts with the data set and its source then goes into preprocessing the data and normalizing it. In addition, it discusses how the data was split into three different sets *(train, test val)* and finally presents the two different classification methods that were used to predict the labels. Section 4 presents the results gotten from the models and analyzes them. Section 5 concludes the project and discusses some conclusions that can be made from the results.

2. PROBLEM FORMULATION

In this problem, there are 5110 data points and each data point is characterized with 10 different features. A single data point represents a person, or more specifically, the description of their health and socioeconomic status. Features of a data point are gender, age, whether the person has hypertension or not, whether the person has heart issues or not, has the person ever been married, what is their work type, what is their residence type, average glucose level, BMI, and smoking status.
The label or the quantity of interest in this project is whether the person has a stroke or not. In the data set, there is a known label value in the column 'stroke'. This is a binary label since there are two possible outcomes (No stroke: 0 and Stroke: 1)

3. METHODS

The dataset used in this project is from https://www.kaggle.com/fedesoriano/stroke-prediction-dataset. The original dataset contained 5110 data points with 12 attributes, but since some of the rows were containing missing values, they had to be removed. One of the attributes is the patient ID, which is not relevant for this problem so it was removed. The final dataset consisted of 4909 observations. At first, the dataset was preprocessed. It contained some categorical variables that had to be transformed into numerical:

- smoking_status: 'Unknown' to 0, 'never smoked' to 1, 'formerly smoked' to 2 and 'smokes' to 3
- gender: 'Male' to 0, 'Female' to 1 and 'Other' to 2
- ever_married: 'No' to 0 and 'Yes' to 1
- work_type: 'children' to 0, 'Never_worked' to 1, 'Govt_job' to 2, 'Private' to 3 and 'Self-employed' to 4

- Recidence_type: 'Rural' to 0 and 'Urban' to 1

```
from sklearn import preprocessing

y = strokedata.pop("stroke")
X_bin = strokedata[["hypertension", "heart_disease", "ever_married", "Residence_type"]]
strokedata = strokedata.drop(X_bin.columns, axis=1)
X = preprocessing.normalize(strokedata)
X = pd.DataFrame(X, columns = strokedata.columns, index=strokedata.index)
X = pd.concat([X,X_bin], axis=1)
```

The dataset was fairly imbalanced because originally 95% of the data points were labeled as 0 (=No stroke) and 5% 1 (=Stroke). To represent the minority (Stroke) group properly and to make sure the accuracy stayed good, a *Synthetic Minority Over-Sampling Technique* (SMOTE) was used. [1]

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state = 2)
X, y = sm.fit_resample(X, y)
```

The data was split into three sets: testing (20%), training (64%), and validation (16%) sets. The model is trained with the train set and the performance of the model is evaluated with the validation set. After the best model is chosen, the performance of the model can be assessed with the test set.

```
from sklearn.model_selection import train_test_split

temp_train_feas, X_test_i, temp_train_labs, y_test = train_test_split(X.index, y, test_size = 0.2)

X_train_i, X_val_i, y_train, y_val = train_test_split(temp_train_feas, temp_train_labs, test_size =
    0.2)
print(X_train_i)
X_train = X.loc[X_train_i]
X_val = X.loc[X_val_i]
X_test = X.loc[X_test_i]
```

In this project, predictions were made with two different models trained on the same data set. Since the data points have multi-class features and binary labels, the best-suited models for this problem are logistic regression and decision tree classifier -algorithms. [2](Chapter 3)

3.1. Logistic Regression

Logistic regression is a classification method which can be used for binary prediction problem, such as in this project. It was chosen because it is fairly simple and easy to implement. [2] In logistic regression, each data point is classified with a classifier function. In this project, Python's scikit-learn library was used. It has multiple useful qualities and functions, which can be seen from the code below. [3] To evaluate the models, the accuracy score metric from sklearn.metrics package was used. The logistic loss was used as a loss function to assess the quality of the classifier. This loss function was chosen because it fits well with binary data. [3] [2] (Chapter 2)

$$L_{log}(y, p) = -(ylog(p) + (1 - y)log(1 - p)) \tag{1}$$

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss

clf = LogisticRegression().fit(X_train, y_train)
y_pred = clf.predict(X_val)
acc_lr = accuracy_score(y_val, y_pred)
print(acc_lr)
```

```
   y_pred_prob = clf.predict_proba(X_val)
   val_loss_lr = log_loss(y_val, y_pred_prob)
   print(val_loss_lr)

   y_pred_prob = clf.predict_proba(X_train)
   train_loss_lr = log_loss(y_train, y_pred_prob)
   print(train_loss_lr)
```

3.2. Decision Tree Classification

A decision tree is another binary predicting classification method. A decision tree is made out of nodes and edges. The computation starts on a root node and continues on the leaf nodes of the tree. It is an efficient classifier for this type of data and it leaves out the unimportant features easily. But besides, it tends to overfit. To avoid overfitting, an optimal maximum amount of leaf nodes that the tree can have had to be found. This was done with a simple algorithm presented below. In this project, Python's scikit-learn library was used to fit and predict the data. [2] (Chapter 3) [3] To evaluate the models, the accuracy score metric from sklearn.metrics package was used. The logistic loss was used as a loss function to assess the quality of the classifier. This loss function was chosen because it fits well with binary data. [3] [2] (Chapter 2)

```
from sklearn import tree
bestloss = 500
bestnodes = None
for i in range(2,200):
   clf_tree = tree.DecisionTreeClassifier(max_leaf_nodes = i).fit(X_train, y_train)
   y_pred_tree = clf_tree.predict(X_val)
   acc_tree = accuracy_score(y_val, y_pred_tree)

   y_pred_prob_t = clf_tree.predict_proba(X_val)
   val_loss_tree = log_loss(y_val, y_pred_prob_t)

   if val_loss_tree < bestloss:
      bestloss = val_loss_tree
      bestnodes = i

clf_tree = tree.DecisionTreeClassifier(max_leaf_nodes = bestnodes).fit(X_train, y_train)
y_pred_prob_t = clf_tree.predict_proba(X_train)
train_loss_tree = log_loss(y_train, y_pred_prob_t)
print(train_loss_tree)

y_train_pred = clf_tree.predict(X_train)
```

4. RESULTS

The final training and validation errors for the models are shown in Table 1. The errors are average values of the logistic loss function performed on the data sets. The normalized confusion matrices for the model predictions on testing set are shown in Fig 1

	LogReg	Tree
Training error	0.51079	0.31061
Validation error	0.51175	0.32204
Validation accuracy	0.76595	0.87533

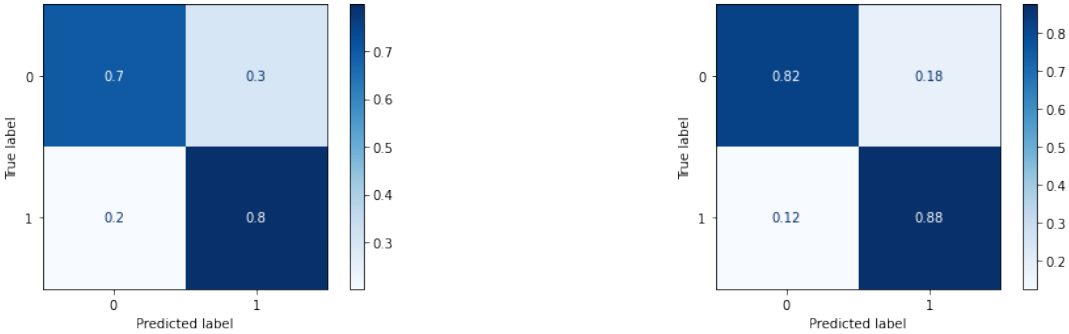
Table 1: Metrics of the two different models

Based on the results, **the decision tree classifier** predicted this problem better. It had lower error rates and better accuracy score. The training error seemed a bit lower for both models, but not significantly, so there was no indication of overfitting. By looking at the confusion matrices it was seen that the decision tree classifier model got more predictions right. The better performing model (decision tree) was therefore evaluated by creating predictions for the testing set:

```
y_test_pred = clf_tree.predict(X_test)
print(accuracy_score(y_test, y_test_pred))

y_test_pred_prob = clf_tree.predict_proba(X_test)
test_loss_tree = log_loss(y_test, y_test_pred_prob)
print(test_loss_tree)
```

Test error : 0.37375
Accuracy : 0.86010



(a) Logistic Regression

(b) Decision Tree Classifier

Fig. 1: Normalized confusion matrices of the test set

5. CONCLUSIONS

Based on the results, both of the models predicted the labels somewhat well. As seen, the training and validation errors are a bit lower on the decision tree -model, as well as the validation accuracy is higher. When running the tree model on the testing set, we get 0.86 accuracy and 0.37 test error, which are pretty good but some improvements could also be done. The first step to make the predictions better would be to add more data points. With more data, the model could be trained better and more accurately. Secondly, in this model, the validation set was split with the scikit-learn function train.test.split(). By choosing the validation set by using k-fold Cross-Validation, the whole dataset would be represented better. Thirdly, the categorical variables could be handled differently. At this project, they were assigned to numerical values in order. To make the predictions more accurate, there could be a better way to convert the categorical variables, such as binarizing all the categorical options. [4]

6. REFERENCES

[1] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, Jun 2002.

[2] Alex Jung. *Machine Learning. The Basics*. mlbook.cs.aalto.fi, 2021.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Alex Jung. CS-C3240 Machine Learning Course Lectures and Materials, 2021.