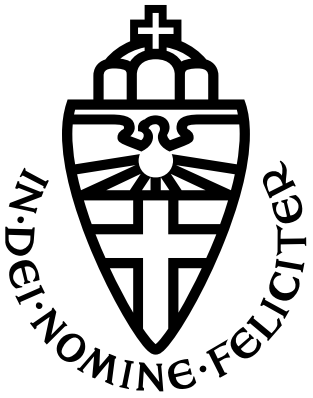
Radboud University Nijmegen



Faculty of Science
Data Mining

Stroke Prediction

How to Handle Biased Data

Gabriele Serafini & Marco Sousa-Poza

January 6, 2022

Abstract

Medical data is, in most cases, highly imbalanced. Therefore it is very easy to get a predictor that is very accurate. A naive approach for such an accurate predictor is to simply classify each instance to be part of the overrepresented target class. However, such a model is not helpful with detecting or predicting diseases. This paper uses medical data with stroke data as a case study in order to explore how adequate predictive models can be created by using the correct scoring methods, data preprocessing steps and class weights. The results show that reaching the same accuracy score with the modified classifiers as the naive model is almost impossible. Nonetheless, they produce more meaningful results by having a good trade-off between correctly predicted stroke and non-stroke cases.

1 Introduction

Medical predictions have become more and more reliable over the past few years, assisting many doctors in finding medical problems preemptively. A major contributor were the different machine learning algorithms that can process the big medical datasets collected from patients and ultimately predict medical problems on new patients. The problem taken in this paper is to accurately classify patients that got a stoke given some medical information.

There have been many attempts on Kaggle[1] to make predictive models that can accurately classify if a patient is at risk of getting a stroke. However many of these projects do not take into account that generally medical data is highly imbalanced, meaning that there are far fewer cases where the patient did have a stroke than not. Therefore the trained models turned out to be too simple and just predicted that it is impossible to get a stroke, regardless of your current health state. This leads to models that are very accurate with the predictions (guess right most of the time), but never actually predict correctly when it really matters and a patient is at risk of getting a stroke.

This paper describes the alterations made on various predictive models such that they produce better/more meaningful results and therefore perform better than the models that treat the data as if it were not imbalanced. The alterations consist of: using the correct cross-validator, using the correct scoring methods and establishing a class-weight. To quantify the differences between proper and improper imbalanced predictions, several predictive models are compared to a *naive classifier* that classifies each instance to be part of the overrepresented target class i.e. did not have a stroke.

2 Methods

In order to find a suitable predictive model for our problem we first need to explore the dataset by visualizing attributes and their relations. After we have a complete understanding of the dataset we are dealing with, we will process this information in order to improve the data quality and manipulate it such that it can be processed by the classification algorithms that we are going to use.

Thereafter, we are going to train four classification models, namely: classification tree, support vector machine, logistic regression and naive Bayes. In order to avoid over-fitting we will use cross validation procedure while we tune the hyper-parameters of our models. Finally, we compare our models on different measures such as recall, ROC curve's AUC, F1 and then discuss the results.

2.1 Data

2.1.1 Data Attributes

Our dataset is composed by 11 attributes. In the following table the attributes are shown including their type.

Attribute name	Attribute type
ID	Ordinal
Gender	Ordinal
Age	Ratio
Hypertension	Ordinal
Heart disease	Ordinal
Ever married	Ordinal
Work type	Nominal
Residence type	Nominal
Average glucose level	Ratio
BMI	Ratio
Smoking status	Nominal

Table 1: Attributes of data

The attribute ID can be removed, since it doesn't add any useful information to the dataset.

[1] data science platform

2.1.2 Attributes Distribution

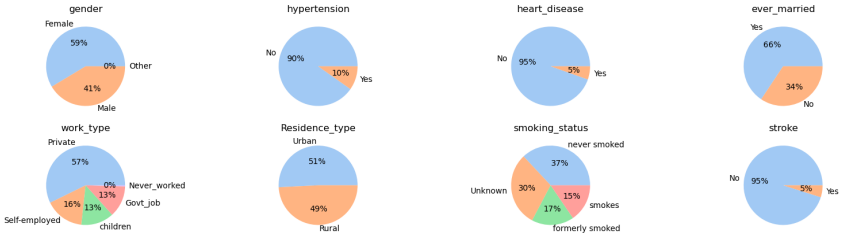


Figure 1: Distribution of Nominal attributes

In figure 1 one can observe the attributes and their distribution over their values. We can observe that the gender is relatively well distributed since 59% of the samples are female and 41% are male. Residence type is almost equally distributed and also the marriage status, where we can observe that nearly 40% of the persons never got married. With hypertension and heart disease we have a strong imbalance toward healthy samples. With smoking we can see that the majority never smoked, while the smokers and formerly smoked are almost the same frequency and we also have a 30% of unknown values. For the work type we can observe that the majority works in the private sector, while the remaining part is almost equally distributed among self-employed, children and government jobs. The most interesting observation that this diagram gives us is that 95% of the persons did not have a stroke, therefore we will have to keep in mind that we have a strongly imbalanced data-set. Ratio attributes are not observable in the figure 1 but are visible in figure 2, which shows the correlations between single attributes and the stroke attribute, we can see that the age is normally distributed, with a majority between the age of 30 and 60. The average glucose level and the BMI are manly distributed over what are considered to be their healthy values.

2.1.3 Correlations

To get a better understanding of which attributes correlate strongly with the target class, each stroke and non-stroke instance is plotted against each possible value of every attribute (see figure 2). As expected the age, the BMI and the average glucose levels show some positive correlation. Counter-intuitive is the the link between smoking and stroke as it does not appear to be as strong as one might expect. Again, the picture shows how imbalanced the dataset is, making it difficult to capture direct relationships between attributes and classification.

As a result, it could be useful to try using PCA to see how the the principal components can indicate variation in the data and how they relate to stroke instances. We will make the visualization in 3 dimensions using the first 3 principal components:

2.2 Data Preprocessing

As a model can only be as great as the data it gets, the data first needs to be properly cleaned. Although it is already quite good, there are some columns that need to be processed before utilization. To be more specific, there are four main points that will need some attention. Firstly all the nominal attributes need to be encoded, secondly all the nil values need to be handled, thirdly all the outliers have to be removed and finally, the data need to be standardized to improve the performance of some predictors.

(1) As one can observe in figure 1 there are some nominal attributes in the dataset. Those will pose problems for numerical machine-learning techniques such as support vector machines. Therefore they need to be encoded to numerical values. However, there are some nominal attributes that cannot be encoded without thinking about what kind of consequences there might be. For example, with the smoking status how should we encode the unknown values? Since they cover over the 30% of the dataset we chose to keep it as a value and let the model learn to account for the missing values making it also easier to classify new data points in the future. The other two possibilities that we considered were to eliminate the column all together, or create a model that tries to predict whether or not someone smokes, but both of these solutions would result in some loss of information.

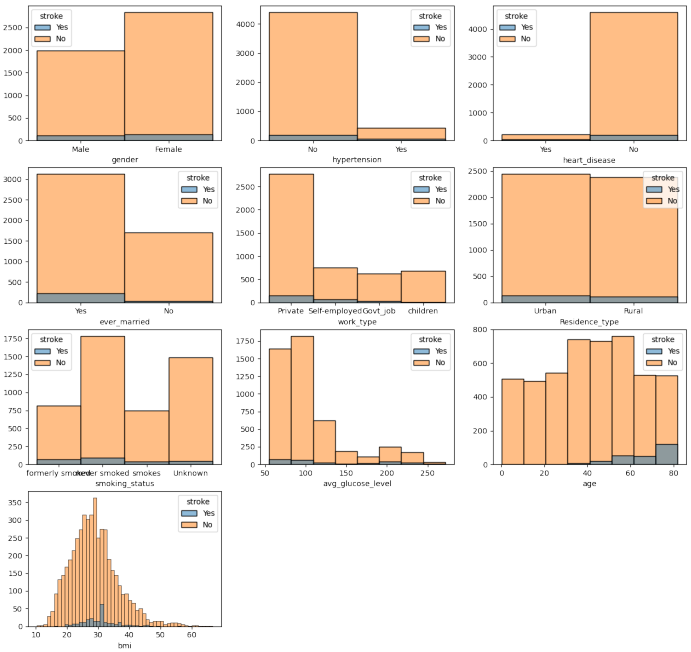


Figure 2: Stroke distribution of Nominal Attributes

Furthermore, for the second solution we computed the correlation between smoking status and other attributes noticing that the relations were not strong enough to create a model for it. For the columns `work_type` we have a very small numbers of never worked values, therefore those data points where removed since they are not a representative group in this data. The same applies to the values `other` of the column gender, where we only have one value. After this consideration, we encoded all the nominal attributes with the following code:

```
gender_dict = {'Female':1, 'Male':0}
bin_dict = {'Yes':1, 'No':0}
worktype_dict = {'Private':3, 'Self-employed':2, 'children':1, 'Govt_job':0}
residence_dict = {'Urban':1, 'Rural':0}
smoke_dict = {'never smoked':1,'formerly smoked':2,'smokes':3, 'Unknown':4}
health_numerical = df_health.replace({
    'gender' : gender_dict,
    'ever_married' :bin_dict,
    'work_type' : worktype_dict,
    'Residence_type' : residence_dict,
    'smoking_status' : smoke_dict,
    'hypertension' : bin_dict,
    'heart_disease' : bin_dict,
    'stroke' : bin_dict
})
```

(2) Handling nil values is also important, since most classification algorithms cannot process non numerical values. In our dataset we only have 201 unknown values, all of which reside in the BMI column. In order to avoid losing the information of these data points, we opted to infere the BMI of those instances by computing the average BMI of the people in the same age range and of the same gender.

```
df_mean = df_health.groupby(['gender', pd.cut(df_health['age'], np.arange(0, 100,
    10))]).mean()
df_mean.drop(['age','avg_glucose_level'], axis=1, inplace=True)
df_mean.reset_index(inplace=True)
df_mean.rename({'bmi':'avg_bmi', 'age':'age_group'}, axis=1, inplace=True)
df_health['age_group'] = pd.cut(df_health['age'], np.arange(0, 100, 10))
```

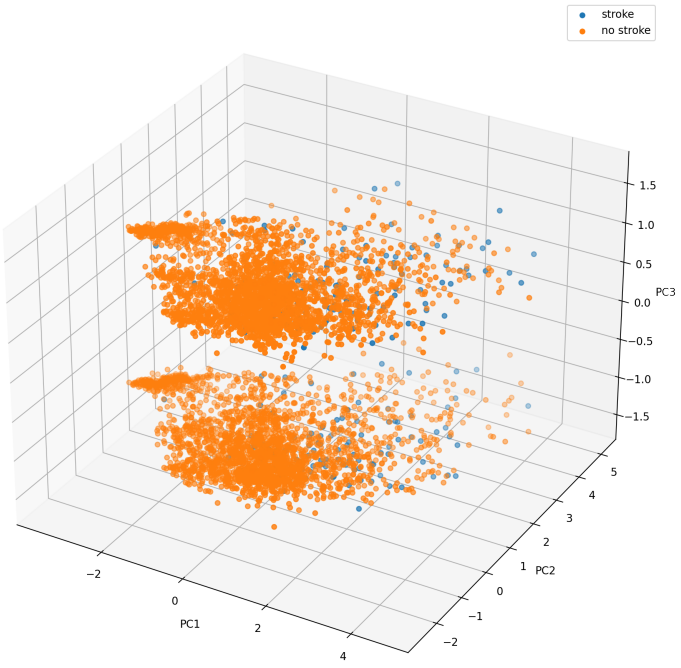


Figure 3: PCA

```
df_map = df_health.merge(df_mean, on=('gender', 'age_group'), how='inner')['avg_bmi'
    ]
df_health['bmi'] = df_health['bmi'].fillna(df_map)
df_health.drop('age_group', axis=1, inplace=True)
```

(3) Regarding the presence of outliers, the dataset seems to be already clean. In fact, we only found some outliers in the BMI column, where we have 4 data points with a BMI over 70. Furthermore, none of them has a stroke and Looking at average, these values seem extraordinary and do not represent the dataset well. We therefore decided to remove them from the dataset.

(4) Finally, before we proceed with the classification task, our data needs to be standardized. For the naive Bayes classifier we will use the original data and not the standardized one, but for the other 3 models it is important to remove the mean and scale it to the unite variance (for the decision tree it is actually not important if the data is scaled or not).[1]

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

2.3 Classification

2.3.1 Scoring

The first thing that needs to be discussed here it to answer the question "what is a good predictive model?". As mentioned in the introduction one can use a naive model which simply states that heart-attacks do not exist. This model would have an accuracy score of 95% when looking at the distribution in figure 1, meaning that 95% of the the model would predict accurately. This is however not a very good model

since we want to inform people that are in danger of getting a heart-attack before they actually get one. The naive model would not be able to do that.

Luckily there are many scoring methods that try to balance true positives (TP) and true negatives (TN) (see figure 4 for definitions of TP and TN). In this project the main scoring methods used were the F_1 and AUC scores:

$$\text{accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$
$$\text{recall} = \frac{TP}{TP + FN}$$
$$F_1 = \frac{2TP}{2TP + FP + FN}$$

The reason being that these scores are not dominated by the overrepresented target class. Therefore they should give better insight on how well the classifier is performing on the imbalanced data. The reason why recall itself is not sufficient is because it tends to overgeneralize the model, meaning that a model will have very few true positives and false negatives by overestimating how endangered someone is of getting a stroke.

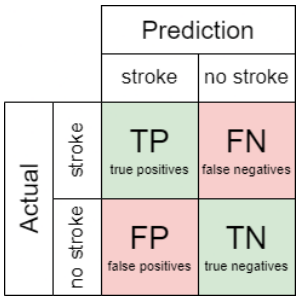


Figure 4: Confusion Matrix Legend

2.3.2 Handling Biased Data

When evaluating the model based on some train-test split and some chosen scoring method using cross validation it is important that all the cases of the target class are well represented in each split. Otherwise the best score could be based on train splits where the non-stroke cases are very dominant. Therefore stratified cross validation was used in order to keep the percentages of stroke and non-stroke instances constant. However this is not sufficient if we want to prioritize cases that had a stroke. After all those are the cases we are interested in. Therefore the false classification of our true-class (stroke cases) are penalized by some factor called *class weight*. Note that this penalization is only possible in optimization based machine-learning techniques. Luckily all the models used do have this property. The class weight for the true-class chosen in this case is inversely proportional to its frequency[4]:

$$\text{class-weight} = \frac{|\text{instances}|}{2 \cdot |\text{instances with stroke}|} \tag{1}$$

Note that this change in how a model is trained does not improve the the performance in the sense that it gets more accurate, but rather shifts its priority towards the underrepresented class.

2.4 Training

Several different machine learning methods were used in order to get a good estimator for stroke predictions. For all of which nested cross validation was used to find the best fitting hyper parameters, if applicable (some classifiers did not have hyper-parameters). As mentioned in the previous section stratified k-fold was used as cross-validator in order to keep the true-class equally represented in each of the k folds. In the following one finds this structure put in pseudo python code:

```
from sklearn.model_selection import cross_validate
from sklearn.something import SomeClassifier
from itertools import product
import pandas as pd

# Define K-Fold object
kf = RepeatedStratifiedKFold(
    n_splits=10,
    n_repeats=5,
    random_state=42
)

# Define hyper-parameters
hp1 = [...]
hp2 = [...]

# Create score data frame
avg_scores = pd.DataFrame(
    columns='test_f1 test_recall test_accuracy test_roc_auc hp1 hp2'.split()
)

# Start nested K-Fold (product=Cartesian product)
for p1, p2 in product(hp1, hp2):
    model = SomeClassifier(
        # add hyper-parameter to model
        parameter1 = p1
        parameter2 = p2
        ...
        # add class weight:
        class_weight = 'balanced'
    )
    scores = pd.DataFrame(
        cross_validate(
            model,
            X,
            y,
            scoring=('recall', 'accuracy', 'roc_auc', 'f1'),
            cv=kf,
            n_jobs=-1
        )
    ).mean()
    scores['hp1'], scores['hp2'] = p1, p2
    avg_scores = avg_scores.append(scores[avg_scores.columns], ignore_index=True)

# Prioritize f1 score & print 10 best scores
avg_scores.sort_values(by=['test_f1'], ascending=False).head(10)
```

The modules used are from the python package scikit-learn[3].

2.4.1 Naive Classifier

This classifier will always classify instances to the overrepresented class. It is only here in order to give something to compare to when looking at the results of the models where proper techniques for imbalanced data classification were used.

The model itself is based on the dummy classifier given in the `sklearn` package and is tested using cross-validation. In the following the code:

```
from sklearn.dummy import DummyClassifier
dc = DummyClassifier(strategy="most_frequent")

# Create dataframe for scores
scores = pd.DataFrame(
    cross_validate(
        dc,
        X,y,
        scoring='recall accuracy f1 roc_auc'.split(),
        cv=kf, n_jobs=-1,
        return_estimator=True
    )
)
scores.drop('fit_time score_time'.split(), axis=1, inplace=True)
scores.mean()
```

2.4.2 Decision Tree

Using a decision tree for predictions turned out to be one of the most reliable methods (see section 3). The hyper-parameters used for this model were to fix the minimum number of splits and the maximum depth that the tree was allowed to have. The impurity measure chosen was the gini-coefficient. After running nested cross validation on these specifications the best parameters where:

```
clf = DecisionTreeClassifier(
    criterion='gini',
    max_depth=12,
    min_samples_split=56,
    class_weight='balanced'
)
```

2.4.3 Complement Naive Bayes

If it is the case that all features are more or less independent from each other, a naive bayes classifier should perform reasonably. There is also evidence that shows that naive Bayes can be a good choice for medical predictions[2]. Therefore this classifier was used as well.

There are several different naive Bayes classifier defined in `sklearn`. The one that gets advertised as being the best for imbalanced data is the *complement naive Bayes classifier*. This classifier library does not take any hyper-parameters. Therefore only cross-validation was used to test the performance of the model. The classifier also only works if all the values of the data are positive. Therefore the non-standardized data was used to train and test the model:

```
from sklearn.naive_bayes import ComplementNB

# Create classifier
model = ComplementNB()

# Create dataframe for scores
scores = pd.DataFrame(
    cross_validate(
        model,
        health_np[:,:-1], #non-standardized data
        y,
        scoring='recall accuracy f1'.split(),
        cv=kf, n_jobs=-1,
        return_estimator=True)
    )
scores.drop('fit_time score_time'.split(), axis=1, inplace=True)
scores.mean()
```

2.4.4 Support Vector Machine

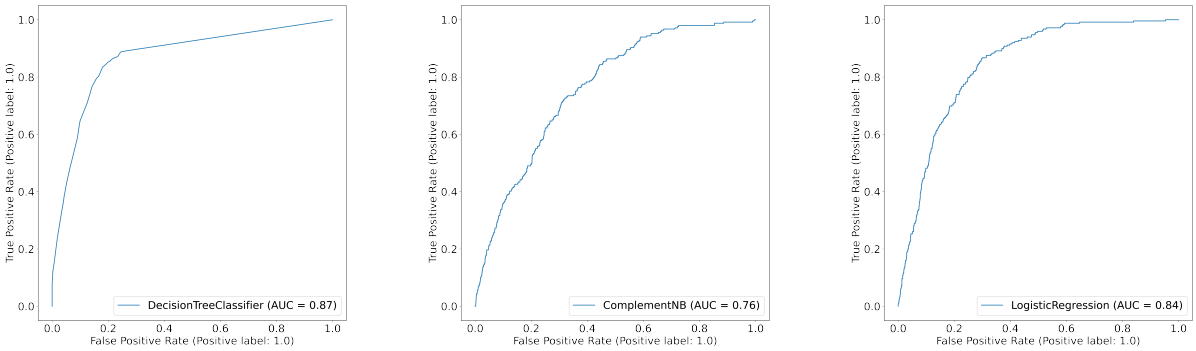
Support vector machines (SVM) use different kernel functions to transfer the data into higher dimensions such that they are linearly separable. To find which kernel is the best one nested cross-validation can be used again. More concretely the hyper-parameters consist of the 3 kernel functions "sigmoid", "radial basis" and "polynomial" from which polynomial also has variations on what degree is used. The SVM class in the `sklearn` also support the class balancing. Therefore it also should perform reasonably on our data.

2.4.5 Logistic Regression

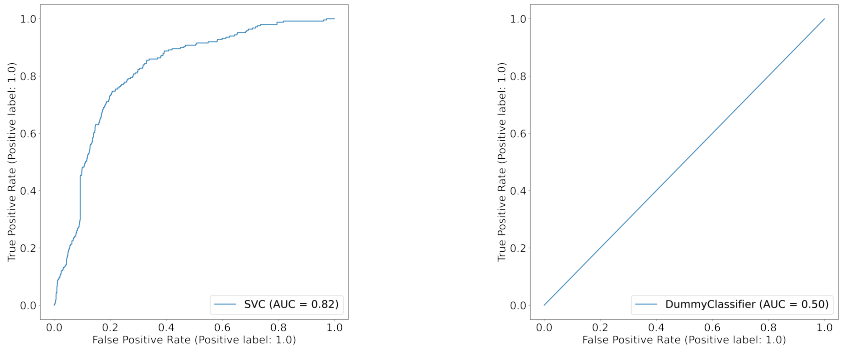
In the style of Occam's razor logistic regression is also included since we are only classifying two target classes. This model also allows for class balancing in the `sklearn` package. Other than that it does not require any additional hyper-parameters. Therefore only cross-validation is used to get the average result of different splits.

3 Results

Now that all the hyper-parameters were found and every model has been trained using those hyper-parameters we can finally do the comparison between the naive and the proper approach of doing



(a) Decision Tree (b) Naive Bayes (c) Log Reg



(d) SVM (e) Naive Classifier

Figure 5: ROC Curves

classification on imbalanced data as promised in the introduction. In the following the different scores of each model:

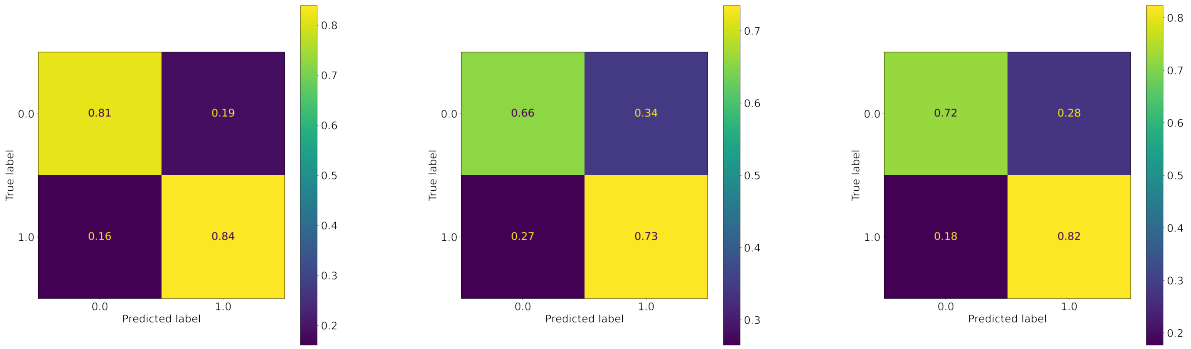
	Decision Tree	Naive Bayes	Logistic Regression	SVM	Naive Model
F1	0.22	0.17	0.23	0.23	0.0
Recall	0.59	0.74	0.80	0.72	0.0
Accuracy	0.78	0.65	0.73	0.76	0.95
AUC	0.72	0.76	0.83	0.80	0.5

As expected the naive classifier has the highest accuracy score, but looking at all the other scores it performs very poorly. Especially looking at the AUC score of 50% is proof that the model is not able to distinguish the different target classes. Furthermore the confusion matrices and ROC curves can be found in figures 6 and 5 to give an even more complete picture on why the naive approach is worse than the other models. Note that the AUC score in figure 5 is the score when the model is predicting all the data. In the table it is the average score when using cross-validation.

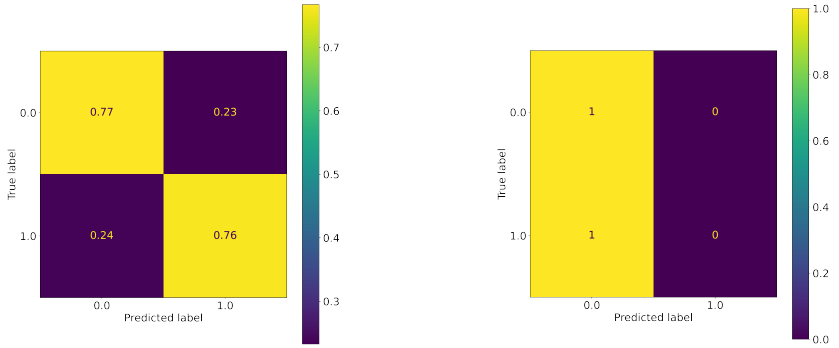
4 Discussion

Given the results is now well established that the naive classifier is not good. However it still needs to be determined if the other predictors are. Although every trained model has better scores in recall, F1 and AUC than the naive classifier, they still have a somewhat low accuracy score. There could be several reasons why this is the case. One reason could be that the data is simply too small to make perfect predictions. For instance, it could be helpful to add a column that includes average cholesterol levels or even genetic data. It could also be that the chosen models are simply not the best for this concrete problem.

Therefore, if this project where to be extended, more classifiers could be trained using similar techniques. For instance neural networks one could try to use. The reason why we limited ourselves to the models



(a) Decision Tree (b) Naive Bayes (c) Log Reg



(d) SVM (e) Naive Classifier

Figure 6: Confusion Matrices

used in this paper is because the `sklearn` package does not have class-weight parameters for all of the classification algorithms. There are of course ways to go around using class-weight using techniques such as oversampling. However, this exceeded the scope of this small project.

In conclusion this project is a good proof that using accuracy as the only mean of measuring the performance of a model is not always sufficient. Although non of the trained models can compete with the accuracy of the naive model, they all produce more meaningful predictions than the naive classifier.

A Tree Classifier

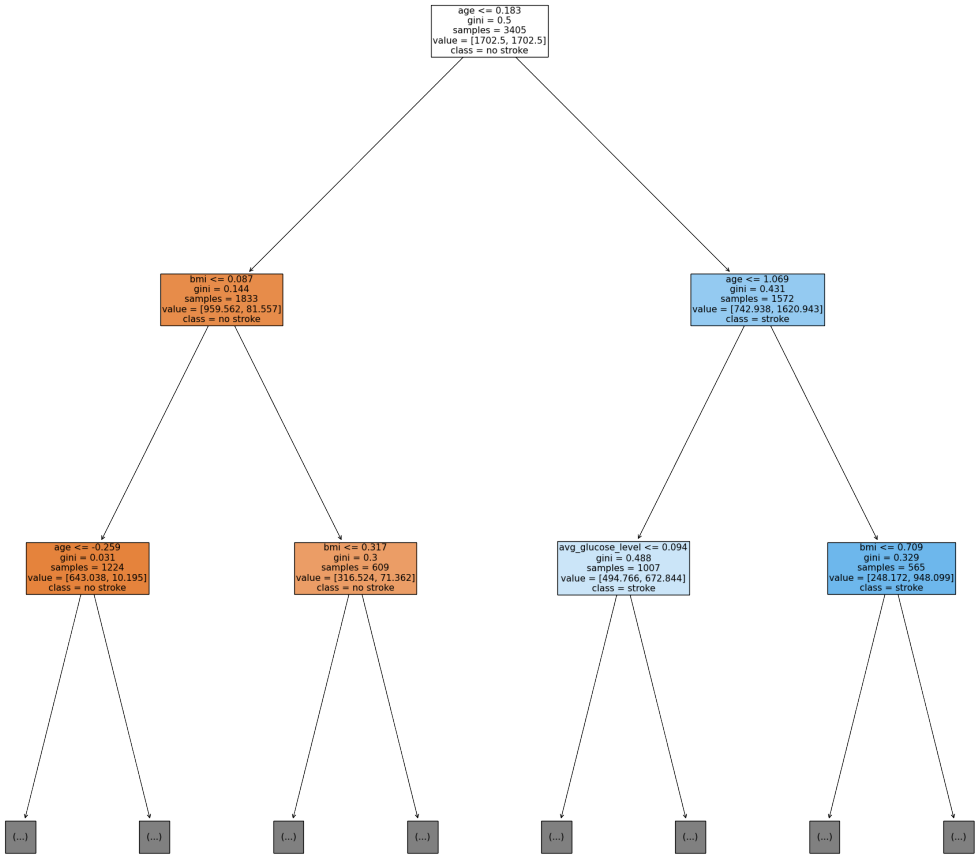


Figure 7: Decision Tree

References

[1] Tan Steinbach Kumar. *Introduction to Data Mining*. 2006.

[2] Mostafa Langarizadeh and Fateme Moghbeli. "Applying Naive Bayesian Networks to Disease Prediction: a Systematic Review". In: *Acta Informatica Medica* 24.5 (2016), p. 364. DOI: 10.5455/aim.2016.24.364-369. URL: https://doi.org/10.5455/aim.2016.24.364-369.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] sklearn. *sklearn.utils.class_ weight.compute_ class_ weight*. 1999. URL: https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html.