Binary Classification Model for Stroke Prediction

Problem description

Given information about patients screened for stroke, I decided to create, evaluate and compare three classification models, predicting the stroke occurrence. Such models can then be used to asses stroke risk for large groups of people automaticially. Althought this data originates from medical industry, it can have several practical applications across multiple domains:

- Risk Assessment for Insurance Companies
  - Insurance companies can use the models to assess the risk of stroke for policyholders, potentially leading to more accurate underwriting and pricing strategies.
- Targeted Interventions for Community Health Programs
  - Community health programs can use the models to identify specific populations or communities that may benefit the most from targeted stroke prevention interventions
- Policy Development for Government and Policymakers:
  - The model's findings can inform the development of public health policies aimed at reducing the overall burden of stroke in the population.

Personally, I was interested in this problem because the dataset had a mix of numerical and categorical variables, which would potentially allow for more complex analysis.

Data

Data overview and cleanup

Source of the data: kaggle. I am not able to judge on the data origin and quality. This might become an issue later on (GIGO).

Lets first see what data are we working with by displaying acouple of rows. Let's look only on categorical data:

	gender	ever_married	work_type	Residence_type	smoking_status
0	Male	Yes	Private	Urban	formerly smoked
1	Female	Yes	Self-employed	Rural	never smoked
2	Male	Yes	Private	Rural	never smoked
3	Female	Yes	Private	Urban	smokes
4	Female	Yes	Self-employed	Rural	never smoked

Let's see what are possible values for each of them:

Attribute	Values
Gender	Male, Female, Other
Work type	Private, Self-employed, Govt_job, children, Never_worked
Residence type	Urban, Rural
Smoking status	formerly smoked, never smoked, smokes, Unknown
Ever married	Yes, No

Let's now look at numerical data of first couple of rows:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
0	67	0	1	228.69	36.6	1
1	61	0	0	202.21	nan	1
2	80	0	1	105.92	32.5	1
3	49	0	0	171.23	34.4	1
4	79	1	0	174.12	24	1

and general statistics associated with them:

	age	hypertension	heart_disease	avg_glucose	bmi	stroke
count	5110	5110	5110	5110	4909	5110
mean	43.22	0.097456	0.0540117	106.148	28.8932	0.048728
std	22.61	0.296607	0.226063	45.2836	7.85407	0.21532
min	0.08	0	0	55.12	10.3	0
25%	25	0	0	77.245	23.5	0
50%	45	0	0	91.885	28.1	0
75%	61	0	0	114.09	33.1	0
max	82	1	1	271.74	97.6	1

Let's check for Nan values:

	0
gender	0
age	0
hypertension	0
heart_disease	0
ever_married	0
work_type	0

	0
Residence_type	0
avg_glucose_level	0
bmi	201
smoking_status	0
stroke	0

We could generate meaningful bmi's for missing rows but let's drop them for now. Let's also drop id collumn.

Data analysis

First of all, since we are working on a classifier, let's see what is the amount of stroke to non stroke patients, because if there is a big difference we will need to mitigate that:

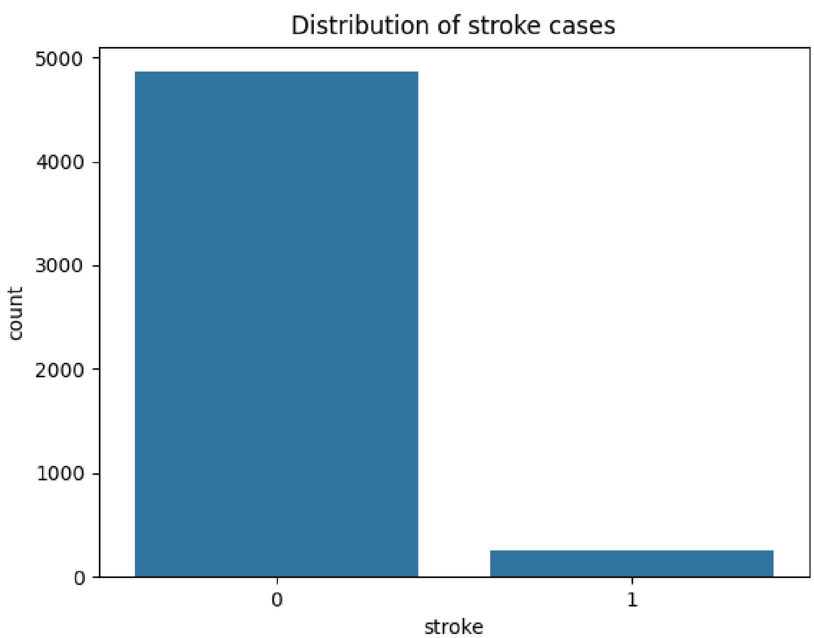
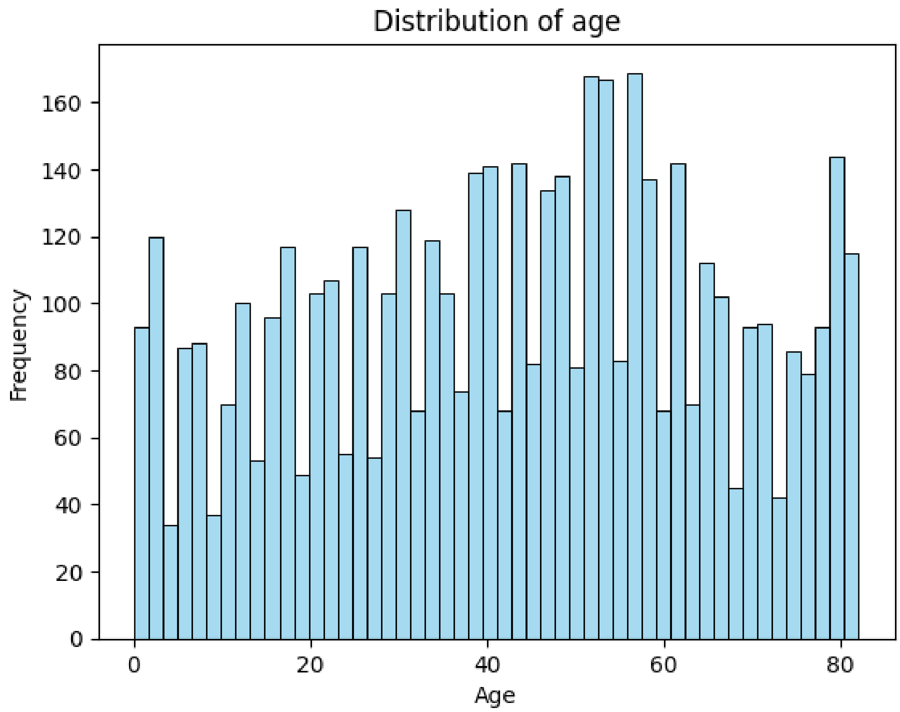
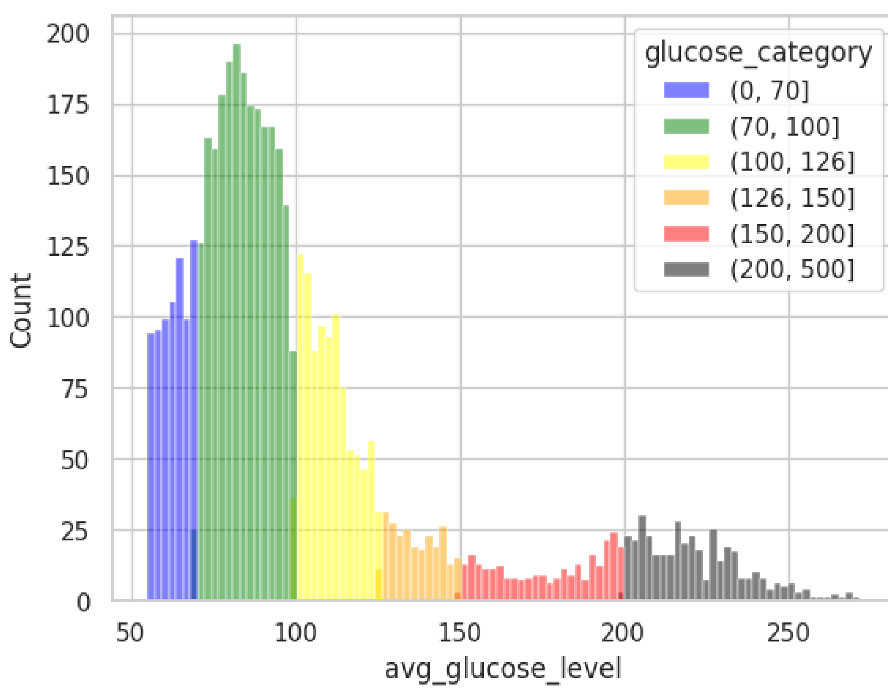


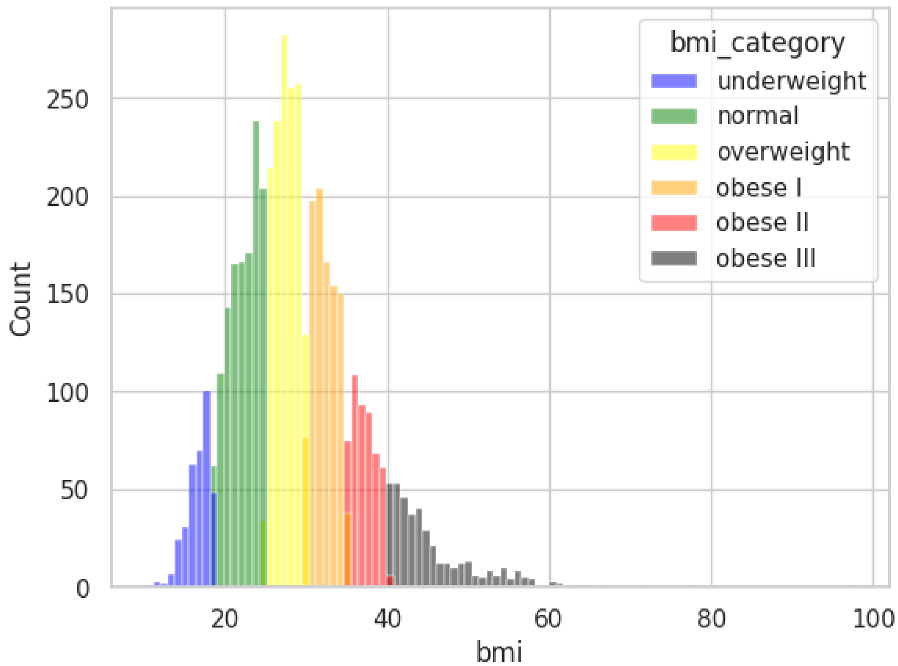
Figure 1: image

As we can see, the dataset is terribly unbalanced.

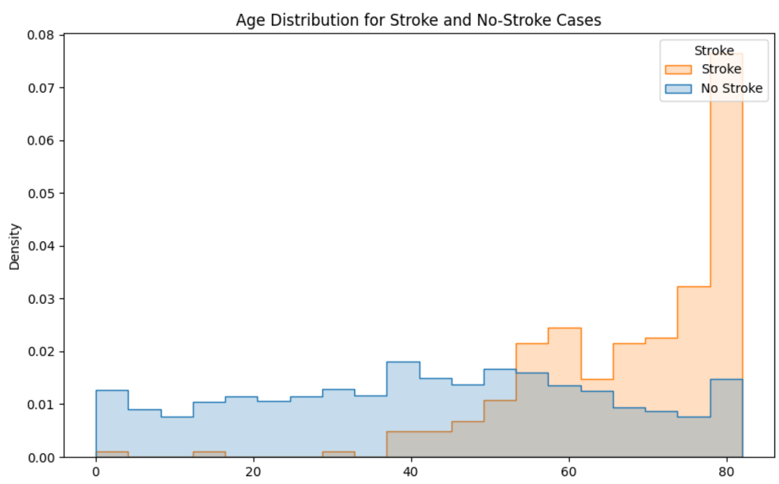
Then, lets see the distribution of numerical values: age, glucose levels and BMI

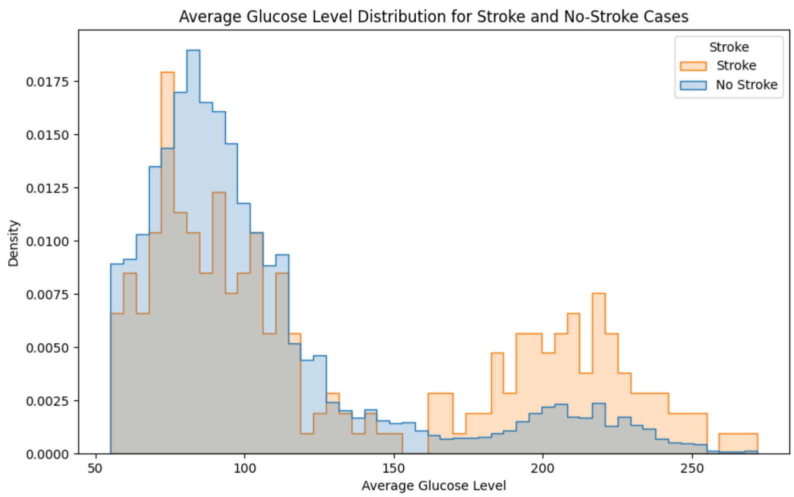

Distribution of age
Frequency
Age
0
20
40
60
80
100
120
140
160

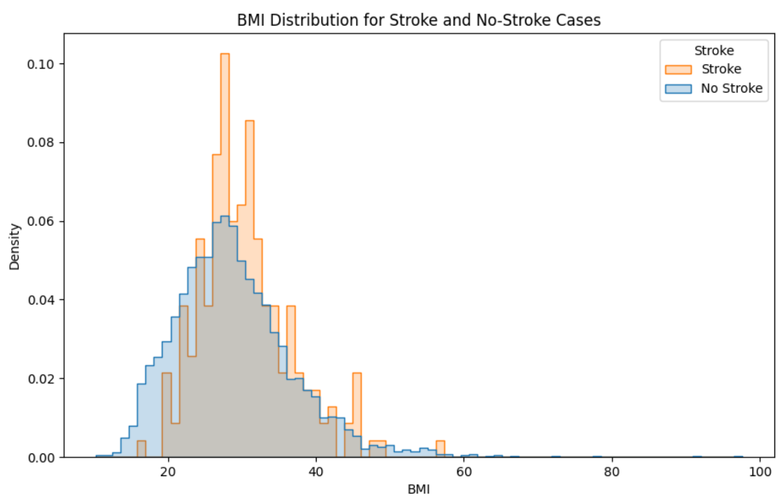

glucose_category
(0, 70]
(70, 100]
(100, 126]
(126, 150]
(150, 200]
(200, 500]
Count
avg_glucose_level
0
25
50
75
100
125
150
175
200
250



Since our dataset is unbalanced, it might be wort to compare the density of age, average glucose levels and bmi split between stroke victims and healthy patients:







There is a clear increase of stroke cases as the age goes up. The same thing happens as in glucose charts, althought not to the same degree. Suprisingly, the BMI seems to not be correlated with stroke. This is suspicious.

The dataset has information on markers that might be connected to stroke occurences, but it's unbalanced. We can to adress this issue in several ways and compare our models' accuracies for each approach.

Models

We will compare:

- Logistic Regression
- Random Forest
- Decision Tree.

For logical regression, I had to encode categorical variables as integers. This is not necessary in the case of the other models but I stuck with encoded data regardless. Given more time, I would compare the potential performance differences in both approaches. Let's first try learning models on

Training models on unaltered data

```python
from sklearn.metrics import mean_squared_error, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler


def create_and_evaluate_models(X_train_scaled, X_test_scaled, y_train, y_test):
    # Logistic Regression
    lr_model = LogisticRegression(random_state=42)
    lr_model.fit(X_train_scaled, y_train)
    lr_y_pred = lr_model.predict(X_test_scaled)

    # Decision Tree Classifier
    dtc_model = DecisionTreeClassifier(random_state=42)
    dtc_model.fit(X_train_scaled, y_train)
    dtc_y_pred = dtc_model.predict(X_test_scaled)

    # Random Forest Classifier
    rfc_model = RandomForestClassifier(random_state=42)
    rfc_model.fit(X_train_scaled, y_train)
    rfc_y_pred = rfc_model.predict(X_test_scaled)

    # compare models
    models = {
        'Logistic Regression': lr_y_pred,
        'Decision Tree': dtc_y_pred,
        'Random Forest': rfc_y_pred
    }

    for model_name, y_pred in models.items():
```

```python
        print("Model     :", model_name)
        print(classification_report(y_test, y_pred, zero_division=1))
        print("\n")


# Extract features and target variable
X = df[[
    'gender', 'age', 'hypertension',
    'heart_disease', 'ever_married',
    'work_type', 'Residence_type',
    'avg_glucose_level', 'bmi',
    'smoking_status'
]]

y = df['stroke']

# Perform one-hot encoding for categorical variables
X_encoded = pd.get_dummies(
    X,
    columns=['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status'],
    drop_first=True
)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_encoded, y, test_size=0.2, random_state=42
)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

create_and_evaluate_models(X_train_scaled, X_test_scaled, y_train, y_test)
```

Logistic Regression	Precision	Recall	F1-Score	Support
0	0.95	1.00	0.97	929
1	1.00	0.00	0.00	53
Accuracy			0.95	982
Macro Avg	0.97	0.50	0.49	982
Weighted Avg	0.95	0.95	0.92	982

Decision Tree	Precision	Recall	F1-Score	Support
0	0.95	0.97	0.96	929
1	0.18	0.13	0.15	53
Accuracy			0.92	982
Macro Avg	0.57	0.55	0.56	982
Weighted Avg	0.91	0.92	0.91	982

Random Forest	Precision	Recall	F1-Score	Support
0	0.95	1.00	0.97	929
1	1.00	0.00	0.00	53
Accuracy			0.95	982
Macro Avg	0.97	0.50	0.49	982
Weighted Avg	0.95	0.95	0.92	982

Our models turned out to have high accuracy, but with low recall, so much so that Random Forest and Logistic Regression models have not detected any stroke instances, classyfing all the test data as negative for stroke. Let's try fixing dataset imbalances.

Balancing the dataset: SMOT

```
from imblearn.over_sampling import SMOTE

# Apply SMOTE to over-sample the minority class
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

scaler = StandardScaler()
X_train_resampled = scaler.fit_transform(X_train_resampled)
X_test_scaled = scaler.transform(X_test)

create_and_evaluate_models(X_train_resampled, X_test_scaled, y_train_resampled, y_test)
```

Logistic Regression	Precision	Recall	F1-Score	Support
0	0.96	0.88	0.92	929
1	0.14	0.34	0.19	53
Accuracy			0.85	982
Macro Avg	0.55	0.61	0.55	982
Weighted Avg	0.91	0.85	0.88	982

Decision Tree	Precision	Recall	F1-Score	Support
0	0.95	0.93	0.94	929
1	0.15	0.23	0.18	53
Accuracy			0.89	982
Macro Avg	0.55	0.58	0.56	982
Weighted Avg	0.91	0.89	0.90	982

Random Forest	Precision	Recall	F1-Score	Support
0	0.95	0.97	0.96	929
1	0.09	0.06	0.07	53
Accuracy			0.92	982
Macro Avg	0.52	0.51	0.51	982
Weighted Avg	0.90	0.92	0.91	982

We've essentialy traded accuracy for other markers. This is certainly an improvement.

Balancing the dataset: undersampling

Let's see what will happen if we undersample:

```python
# load the data again
df = pd.read_csv('healthcare-dataset-stroke-data.csv')

df = df.dropna()
df = df.drop(['id'], axis=1)

stroke_data = df[df['stroke'] == 1]
no_stroke_data = df[df['stroke'] == 0]

# Undersample the majority class (no-stroke)
no_stroke_data = no_stroke_data.sample(n=len(stroke_data))

# Combine the balanced data
balanced_data = pd.concat([stroke_data, no_stroke_data], ignore_index=True)

balanced_data = balanced_data.sample(frac=1).reset_index(drop=True)

X = balanced_data.drop('stroke', axis=1)
y = balanced_data['stroke']

# Perform one-hot encoding for categorical variables
X_encoded = pd.get_dummies(
```

```python
    X,
    columns=['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status'],
    drop_first=True
)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_encoded, y, test_size=0.2, random_state=42, stratify=y
)


# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

create_and_evaluate_models(X_train_scaled, X_test_scaled, y_train, y_test)
```

Logistic Regression	Precision	Recall	F1-Score	Support
0	0.80	0.86	0.83	42
1	0.85	0.79	0.81	42
Accuracy			0.82	84
Macro Avg	0.82	0.82	0.82	84
Weighted Avg	0.82	0.82	0.82	84

Decision Tree	Precision	Recall	F1-Score	Support
0	0.59	0.76	0.67	42
1	0.67	0.48	0.56	42
Accuracy			0.62	84
Macro Avg	0.63	0.62	0.61	84
Weighted Avg	0.63	0.62	0.61	84

Random Forest	Precision	Recall	F1-Score	Support
0	0.73	0.83	0.78	42
1	0.81	0.69	0.74	42
Accuracy			0.76	84
Macro Avg	0.77	0.76	0.76	84
Weighted Avg	0.77	0.76	0.76	84

It seems like Logistic Regression is doing best out of all three. Generally, precision is down, but recall is better. I think there is just too few examples for model to work well.

Balancing the dataset: oversampling

Lets see what happens if we oversample:

```
df = pd.read_csv('healthcare-dataset-stroke-data.csv')

df = df.dropna()
df = df.drop(['id'], axis=1)

stroke_data = df[df['stroke'] == 1]
no_stroke_data = df[df['stroke'] == 0]

# Oversample the minority class (stroke)
stroke_data = stroke_data.sample(n=len(no_stroke_data), replace=True)

# Combine the balanced data
balanced_data = pd.concat([stroke_data, no_stroke_data], ignore_index=True)

balanced_data = balanced_data.sample(frac=1).reset_index(drop=True)

X = balanced_data.drop('stroke', axis=1)
y = balanced_data['stroke']

# Perform one-hot encoding for categorical variables
X_encoded = pd.get_dummies(
    X,
    columns=['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status'],
    drop_first=True
)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_encoded, y, test_size=0.2, random_state=42, stratify=y
)


# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

create_and_evaluate_models(X_train_scaled, X_test_scaled, y_train, y_test)
```

Logistic Regression	Precision	Recall	F1-Score	Support
0	0.77	0.72	0.74	940
1	0.74	0.78	0.76	940

Logistic Regression	Precision	Recall	F1-Score	Support
Accuracy			0.75	1880
Macro Avg	0.75	0.75	0.75	1880
Weighted Avg	0.75	0.75	0.75	1880

Decision Tree	Precision	Recall	F1-Score	Support
0	1.00	0.95	0.97	940
1	0.95	1.00	0.98	940
Accuracy			0.98	1880
Macro Avg	0.98	0.98	0.98	1880
Weighted Avg	0.98	0.98	0.98	1880

Random Forest	Precision	Recall	F1-Score	Support
0	1.00	0.98	0.99	940
1	0.98	1.00	0.99	940
Accuracy			0.99	1880
Macro Avg	0.99	0.99	0.99	1880
Weighted Avg	0.99	0.99	0.99	1880

Logistic regression performs worse than when undersampling, but the decision tree and random forest models are working suspiciously well.

Summary

This turned out to be more difficult than I anticipated. Unbalanced dataset has been the biggest issue, and my attempts to mitigate it were either not a great improvement (SMOT, undersampling) or suspiciously succesful (oversampling). Given more time I would investigate further, by plotting learning curves to check for potential under/over fitting issues.