Hanoi University of Science and Technologies

IT3190-123220 Machine Learning

Semester 20202

Course Project

Stroke Prediction

Group 18

Hoang Nguyen Minh Nhat - 20194445

Pham Thanh Hung - 20194437

Tran Quoc Lap - 20194443

Definition

Project overview

Stroke is one of the major causes of death.

In this project, we're building a model capable of early predicting whether a patient is likely to get a stroke or not. The prediction is made by learning from thousands of patients. Each patient's information includes gender, age, smoking status, hypertension status, marital status, etc.

Instead of building everything from scratch, we'll take advantage of various tools from the Scikit-learn, Pandas, Numpy, Imbalanced-learn library. This is due to 2 reasons:

1. We don't have enough time to build everything from scratch. Indeed, we'd tried and canceled because this took a big chunk of our time before we actually got round to the Stroke prediction.

2. Our goal is to get familiar with doing experiments in DS and ML, understand the workflow of a project and get an insight from the dataset as well as different algorithms.

We select CART, SVM, ANN algorithms in this project.

Problem Statement

The tasks involved are the following:

1. Download the Stroke dataset from Kaggle: https://www.kaggle.com/fedesoriano/stroke-prediction-dataset
2. Do basic data preparation including data cleaning.
3. Test the impact of different data transforming and sampling techniques on each model's performance.
4. Tune each model's parameters
5. Compare among 3 models on the final prediction.

Test Option and Evaluation Metric

We'll use Repeated Stratified 5-fold Cross Validation to estimate F1 score.

$$F_1 = 2\frac{Precision \times Recall}{Precision + Recall}$$

Where $Precision = \frac{True\ Positive}{True\ Positive + False\ positive}$ and $Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$

This metric is selected because our dataset is severely imbalanced. (see Fig. 1)

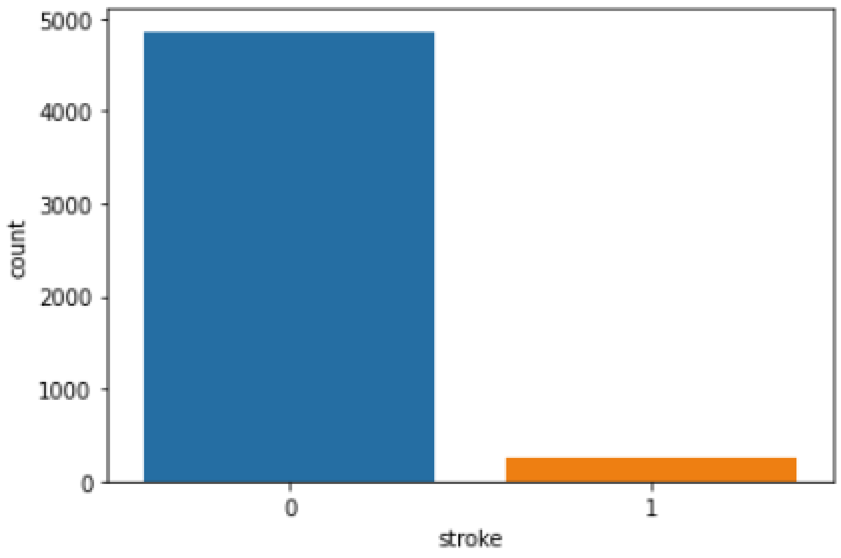


Fig. 1: `Stroke` distribution ("class_distribution.png"). Among 5110 records, 0 accounts for 4861 records (95.1%) and 1 accounts for 249 records (4.9%). This shows that our data is severely imbalanced.

This is beneficial in 2 ways:

1. Avoid misleading evaluation results: A 5-fold cross validation is appropriate for an imbalance dataset because a fold is ensured to be a representative sample of the domain. F1 score is considered to be a proper measure for severely imbalanced classification .
2. Explain our desire: In our specific problem, "stroke" is positive class, we would prefer to have both Precision and (especially) Recall as high as possible, which means we'll implement in order that F1 could be as high as possible.

During the training process, we use a validation set extracted from 10-time Repeated Stratified 5-fold Cross Validation.

Analysis

Data Exploration

The Stroke dataset has 5110 records, each record has the following fields:

❖ id	unique identifier (int)
❖ gender	"Male", "Female" or "Other" (string)
❖ age	age of the patient (float). Min: 0.08, Max: 82.0
❖ hypertension	0 for not having hypertension, 1 for having hypertension (int)
❖ heart_disease	0 for having heart diseases, 1 for having heart disease (int)
❖ ever_married	"No" or "Yes" (string)
❖ work_type	"children", "Govt_jov", "Never_worked", "Private" or "Self-employed" (string)
❖ Residence_type	"Rural" or "Urban" (string)
❖ avg_glucose_level	average glucose level in blood (float). Min 55.12, Max: 271.74
❖ bmi	body mass index (float). Min: 10.3, Max: 97.6
❖ smoking_status	"formerly smoked", "never smoked", "smokes" or "Unknown" (string)
❖ stroke	1 if the patient had a stroke or 0 if not (int)

Each of these attributes is observed in 5110 records, except for `bmi` which have 4909 records observed. This implies that `bmi` is having a fair number of missing values.

By observing the uniques values of each attributes, we can easily split these attributes into:

- Numerical variables: `age`, `avg_glucose_level`, `bmi`
- Categorical variables: `smoking_status`, `gender`, `hypertension`, `heart_disease`, `ever_married`, `work_type`, `Residence_type`

Exploratory Visualization

All data visualizations are done in "data_visualization.ipynb". Down here we show some figures worth mentioning.

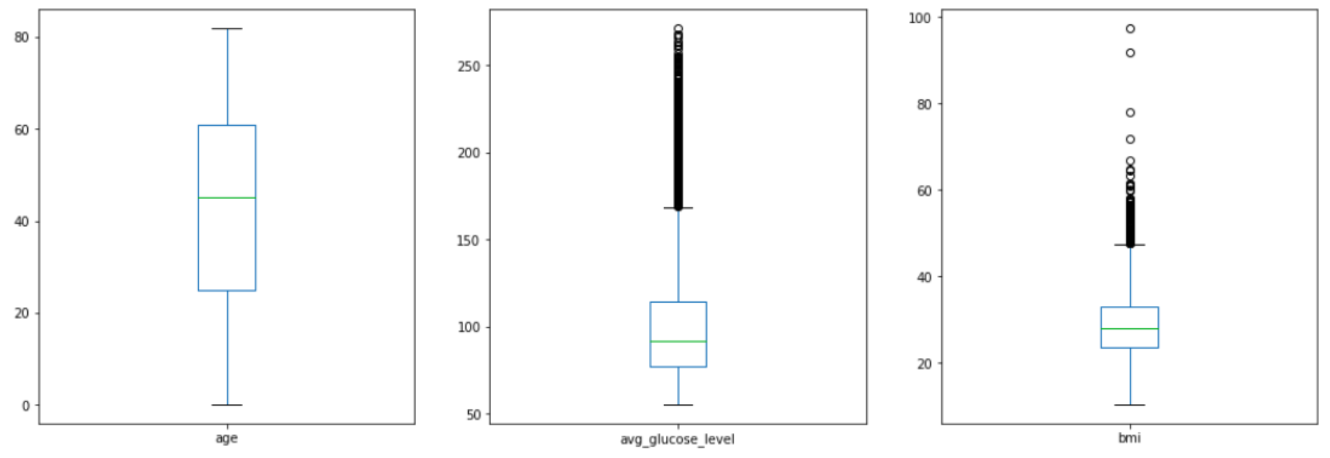


Fig. 2: Box and whisker plots for `age`, `avg_glucose_level`, `bmi` (“boxplot_before.png”).

We pay attention to the `bmi` whose several records are quite far from others. This suggests they could be outliers and possibly need removal (See the Data Preparation section).Fig. 3: Scatter pair plot with respect to `stroke` attribute (“pairplot.png”). **Note**: all values 0 of stroke are put BEHIND values 1 before plotting, indeed they OVERLAP each other.

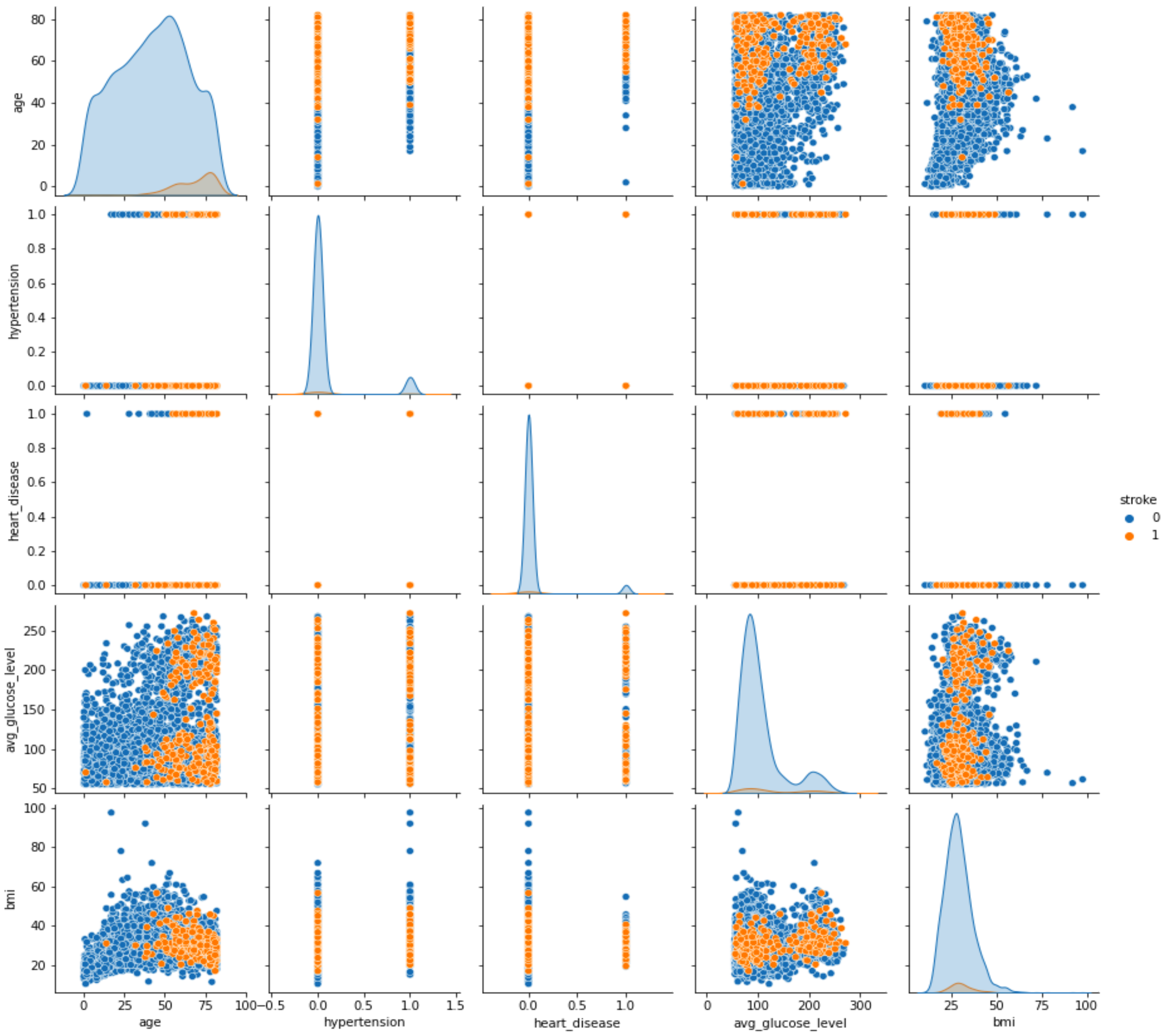
By eye, we can’t find any single attribute that can clearly classify `stroke`. The only characteristic we can realize is that: most stroke patients whose `age` is greater than 50 and whose `bmi` is smaller than 50.

Algorithms and Techniques

I. Classification and Regression Tree (CART)

CART is available in DecisionTreeClassifier in Scikit-learn, is one of the most widely-used algorithms in supervised learning. CART requires very little data preparation. It can work with both numerical and categorical variables, handle missing values, robust to noise, and capable of doing feature selection automatically. However with Scikit-learn DecisionTreeClassifier does not support handling missing values and categorical variables if they are not in numeric form.

The following parameter can be tuned to optimize DecisionTreeClassifier:



❖ splitter: Decision trees tend to overfit on data with a large number of features. However, it can do feature selection automatically by setting splitter="best" (which bases on (Im)purity). Another value is "random". If we have hundreds of features, "best" is preferred because "random" might result in features that don't give much information, which lead to a deeper, less precise tree.

❖ max_depth: This indicates how deep the tree can be. The deeper the tree, the more splits it has and it captures more information about the data. However, max_depth needs controlling to prevent overfitting.

- min_samples_split: The minimum number of samples required to split an internal node. When min_samples_split increases, the tree becomes more constrained.

- min_samples_leaf: The minimum number of samples required to be at a leaf node. At any depth, regardless of min_samples_split, a split point can only be accepted if each of its leaves have at least min_samples_leaf samples.

- max_features: The maximum number of features to consider when looking for the best split.

- ccp_alpha: Cost-complexity pruning alpha is used to post-pruning the tree in order to avoid overfitting. It defines cost-complexity measure R(T)=R(T) +T where R(T) is the total misclassification rate of leaf nodes and |T| is the number of leaf nodes. The nodes with the smallest effective alpha are pruned first.

II. Support Vector Machines (SVMs):
SVMs are useful techniques for data classification. It is known for its accuracy, stability and speed. Also, it is considered easier to use than Neural Networks. the SVC (C-based SVM) of Scikit-learn is chosen to implement the SVM algorithm for this problem

These are the parameters of SVC:

- C: Penalty parameter of the error terms, define how much the model penalizes for an error. It is also called the Regularization parameter, which is inversely proportional to the strength of regularization to the model.
- kernel: Kernel type to be used in the algorithm. Can be one of: Linear, Polynomial, RBF or Sigmoid
- gamma, coef0, degree: Each kernel requires at least one of these parameters (except the Linear)
- class_weight: A dictionary to specify weight for each class. If specified, the parameter C of class i will be modified to class_weight[i]*C. The purpose of this is to handle unbalanced dataset.

III. Artificial Neural-Network (ANN):
ANN is known to be effective in classification problems. Another point is that ANN is adaptive with uncleanse data and future missing data - which might probably happen with this problem. However, ANN has a critical drawback that it cannot show the process of making predictions clearly (It works somewhat similarly with the human brain - at a lower

level). Fortunately, in this problem, the results are more important and users might not really need to know about how the decisions are made.

In the scope of this problem, we use the Multi-layer Perceptron Classifier (MLPClassifier) of sklearn to learn. This model has several parameters to tune up such as:

- hidden_layer_sizes
- activation
- Solver
- Alpha
- Batch_size
- ...

However, the easiest-to-observe parameters are hidden_layer_sizes and max_iter which indicate the number of hidden layers and maximum number of iterations. Those information also demonstrate the complexity of the model and affect vastly on model performance.

Methodology

Data Preparation

In this Machine Learning course, we're not going to spend much energy in data preprocessing, because it seems more relevant to the Data Science course. Instead, we'll mostly focus on model-centric.

With CART

The preparation steps are done in cart.ipynb, which includes:

- ❖ Remove label noise and outliers (using Quantile Range Method).
- ❖ Split the dataset into a training set and test set (using Stratified train_test_split).
- ❖ Impute missing values in `bmi` with its mean.
- ❖ Do data transformation (Encode categorical variables & Discretize numerical variables).
- ❖ Do sampling training set (Oversampling & Undersampling)

With SVM

The preparation steps are:

- ❖ Impute missing values by a simple **decision tree model**
- ❖ Split the dataset into a training set and test set (using Stratified train_test_split).
- ❖ Do data transformation (Encode categorical variables & Scale numerical variables).

Data sampling

Severely imbalanced dataset might degrade a model's performance. The model is often biased toward the majority class, and the minority class is harder to learn. One approach to deal with imbalance classification is applying oversampling and undersampling techniques.

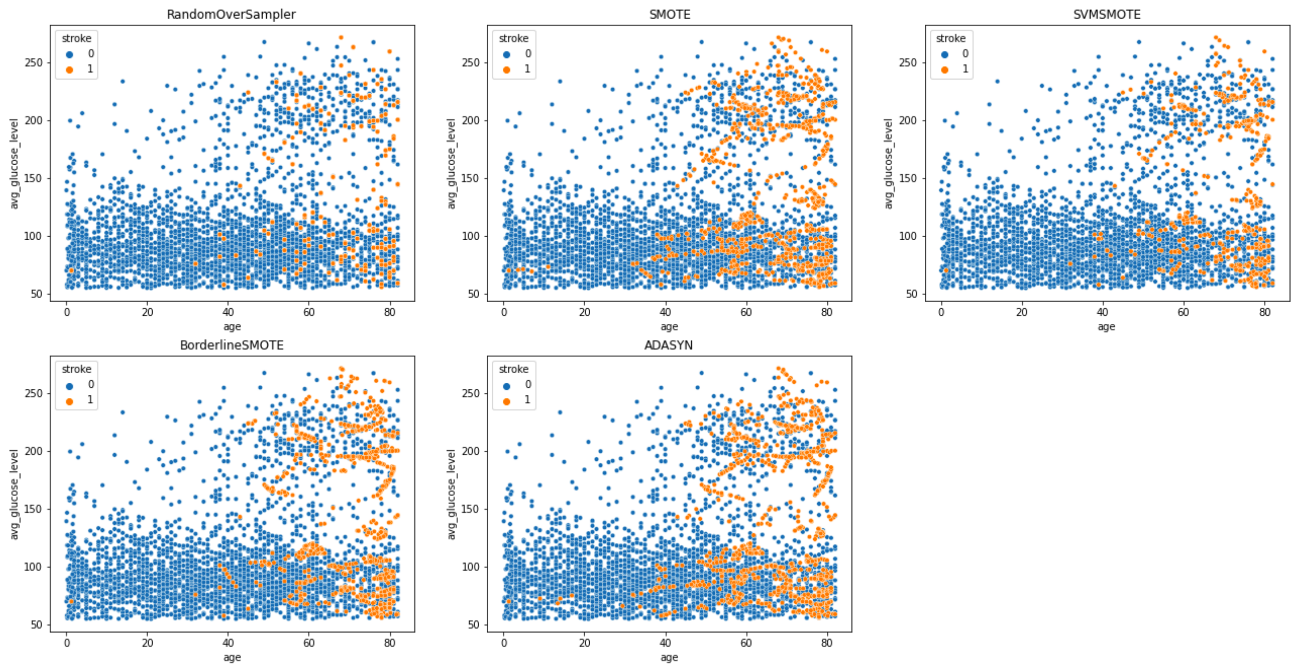


Fig. 4: Scatter plot for data distribution after oversampling (oversampling.png). The original distribution looks the same as the figure of RandomOverSampler. #stroke:#not_stroke is set at 3:10 for all techniques. ****Note****: all markers of 0 are put BEHIND all markers of 1 before plotting, indeed they OVERLAP each other.

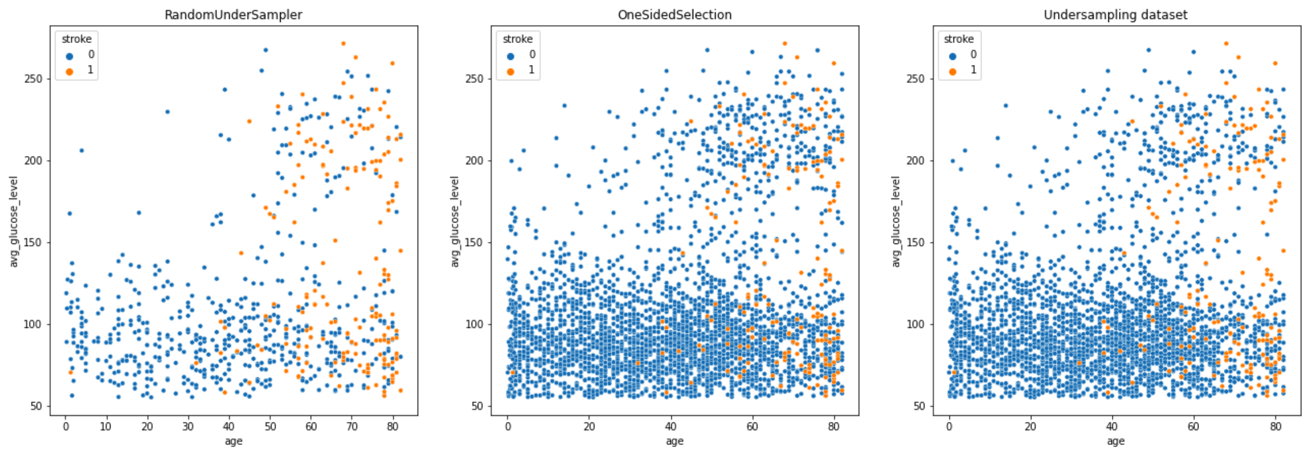


Fig. 5: Scatter plot for data distribution after undersampling (undersampling.png). #stroke:#not_stroke is set at 3:10 for RandomUnderSampling. ****Note****: all markers of 0 are put BEHIND all markers of 1 before plotting, indeed they OVERLAP each other.

Implementation

With CART

The implementation is divided into the following steps:

- ❖ Do data preparation as described in the previous section.
- ❖ Test the impact of different data sampling techniques as described in the previous section.
- ❖ Test the impact of class-weight on the model's performance.
- ❖ Test the impact of different encode techniques on the model's performance.
- ❖ Test the impact of discretization of the model's performance.
- ❖ Tune model and plot the results for the train set and validation set.

In the implementation with CART algorithm, we've done many experiments. However, for the sake of a brief report, we'll just refer to notable results and skip the others. For complete experimental results, please read cart.ipynb.

After reading articles, we're recommended to use sampling to balance data. We've done experiments with both oversampling and undersampling. In Fig. 6, we summarize the experimental result. Fig 6 demonstrates the general results of effects of different sampling techniques on CART model.

Data sampling is intuitively believed to improve decision tree's performance because the class assigned to a leaf node is affected by the number of instances from each class in that leaf. In our problem, we certainly want our CART to be more sensitive to \`stroke\` instances for the sake of early warning. If we don't oversampling or undersampling, the portion of \`stroke\` instances in a leaf might be too low, which causes more bias toward \`not stroke\` instances.

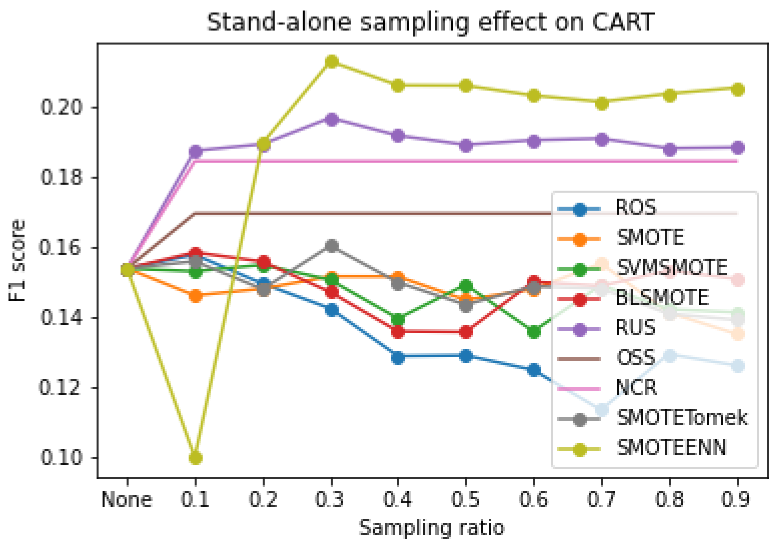


Fig. 6: Effect of sampling on CART. Model used: Random oversampling, SMOTE, SVM SMOTE, Borderline SMOTE, Random undersampling, One-sided selection, Neighbourhood cleaning rule, SMOTE Tomek, SMOTE Edited nearest neighbour. #stroke:#not_stroke is set from 0.1 to 0.9 for all sampling models, except OSS and NCR.

After many runs, SMOTE ENN shows the most promising scenario. This is unsurprising because beside balancing data via SMOTE, the technique also pays attention to the unambiguity of examples in the data set and increases the certainty of decision boundaries.

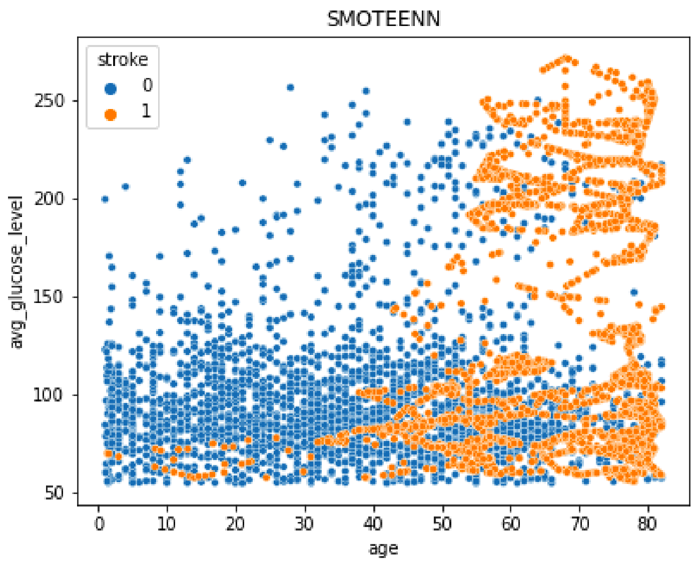


Fig. 7: Data distribution after SMOTEENN. #stroke:not_stroke is set at 3:10. **Note**: all markers of 0 are put BEHIND all markers of 1 before plotting, indeed they OVERLAP each other.

We can clearly see in the top right of Fig. 7 , in combination with extending the coverage of stroke instances, a lot of majority-class examples around the area covered by minority class are removed, which may help increase Recall while not decreasing Precision much.

According to some articles, a subfield of machine learning called cost-sensitive learning can be applied to solve the problem of imbalance classification. This can be carried out with CART by controlling the `class_weight` parameter in Scikit-learn DecisionTreeClassifier. Basically, the weight of each instance in a leaf node will account for the class determination of that leaf. In our experiment, we tested different class weights, where stroke_weight:not_stroke_weight ranges from 1:1 to 23:1. However, the performance of CART does not change, as illustrated in Fig.8, which is surprising.

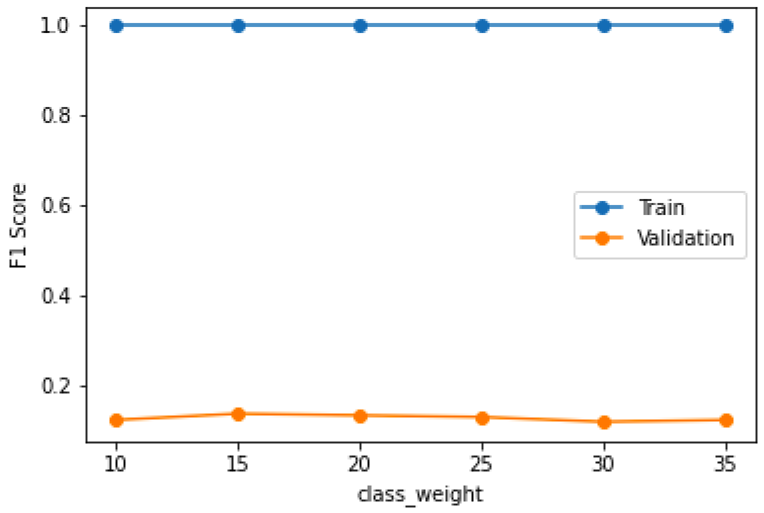


Fig .8: Class weight tuning for CART. #stroke:#not_stroke ranges from 10:1 to 35:1.

Ultimately, after several further experiments, we decided to choose SMOTENN with ratio 0.3 in the rest of the project, for the step of data transforming and parameter tuning.

The next major experiments involve comparing Ordinal vs Onehot encoding, and the difference between the two's impact on the performance is very little. In general, with categorical variables, onehot encoding is seemingly more preferred as it does not create additional relationships. However, with decision trees, some articles claim that onehot encoding degrades the performance as it creates many more variables with less feature importance. Unfortunately, we could not justify these claims in this project. In Fig. 9, the performance of CART model on these encoding strategies does not differ much, seemingly because our dataset has only 11 columns and they do not separate examples well. Another experiment we do is testing if discretization is good for our problem because decision trees prefer discrete variables. However, the performance of CART degrades after discretization. Again, we could not explain, and for the sake of a brief report we leave source code and figure in cart.ipynb.

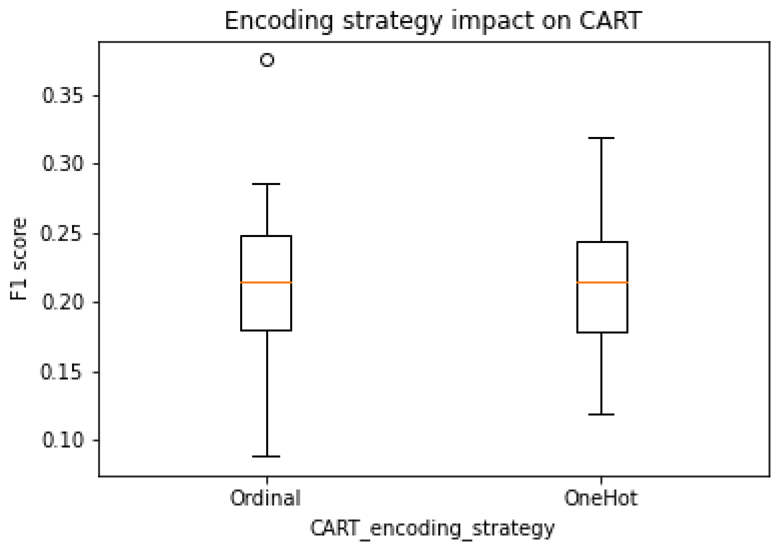


Fig. 9: Encoding strategies' impact on CART. Average F1 score for Ordinal is 0.212643, and for Onehot is 0.214212.

The next step is tuning parameter of CART. We've chosen `splitter`, `min_samples_split`, `min_samples_leaf`, `max_depth`, `max_features`, `ccp_alpha`. For the sake of a brief report, we'll mention the most notable results only. You can view the complete result in cart.ipynb.

Concisely, by utilizing grdd search, we found a good combination of `min_samples_split`, `min_samples_leaf`, which slightly improved the performance of CART. With that combination, we continue to tune `max_depth` and `ccp_alpha`. Fig. 10 illustrates the experiment.

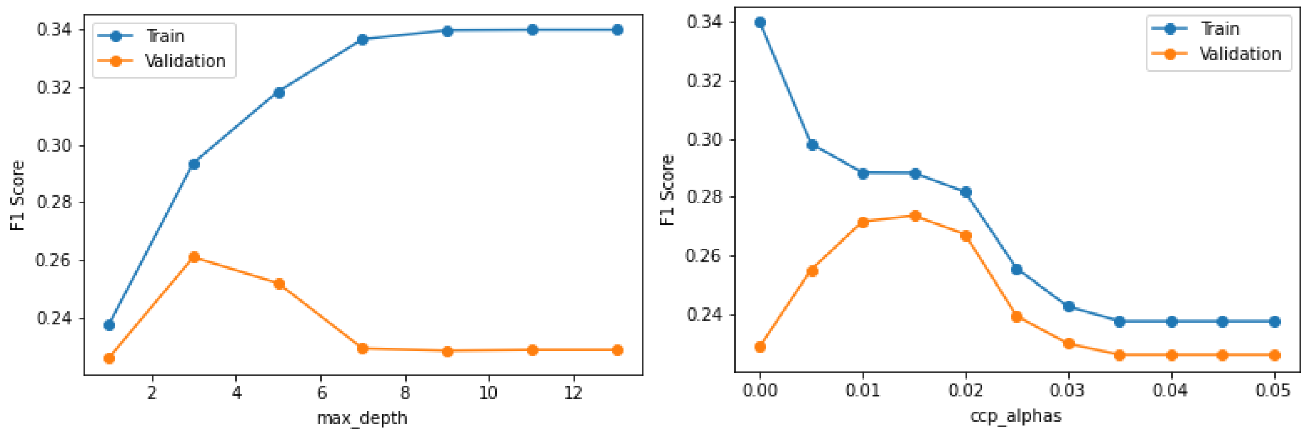


Fig. 10: `max_depth` (left) and `ccp_alpha` (right) tuning. `max_depth` ranges from 0 to 30 and `ccp_alpha` ranges from 0 to 0.05. **Note**: data sampling on the training set is done before tuning (mentioned above).

`max_depth` performs pre-pruning and `ccp_alpha` performs post-pruning. At first glance, it's tempting to choose `max_depth` = 5 or `ccp_alpha` in range (0.01, 0.02) because they make F1 significantly hike up. However, as we plot the data on each `max_depth` and `ccp_alpha` value (see Fig. 11), the smaller `max_depth` and the higher `ccp_alpha` makes prediction more unreal. In case `max_depth` < 5 or `ccp_alpha` > 0.006, we clearly see that CART is far less flexible, almost all examples with `age` > 70 or `avg_glucose_level` > 200 are labeled as stroke. Though the recall could be very high, we have to compromise with precision.

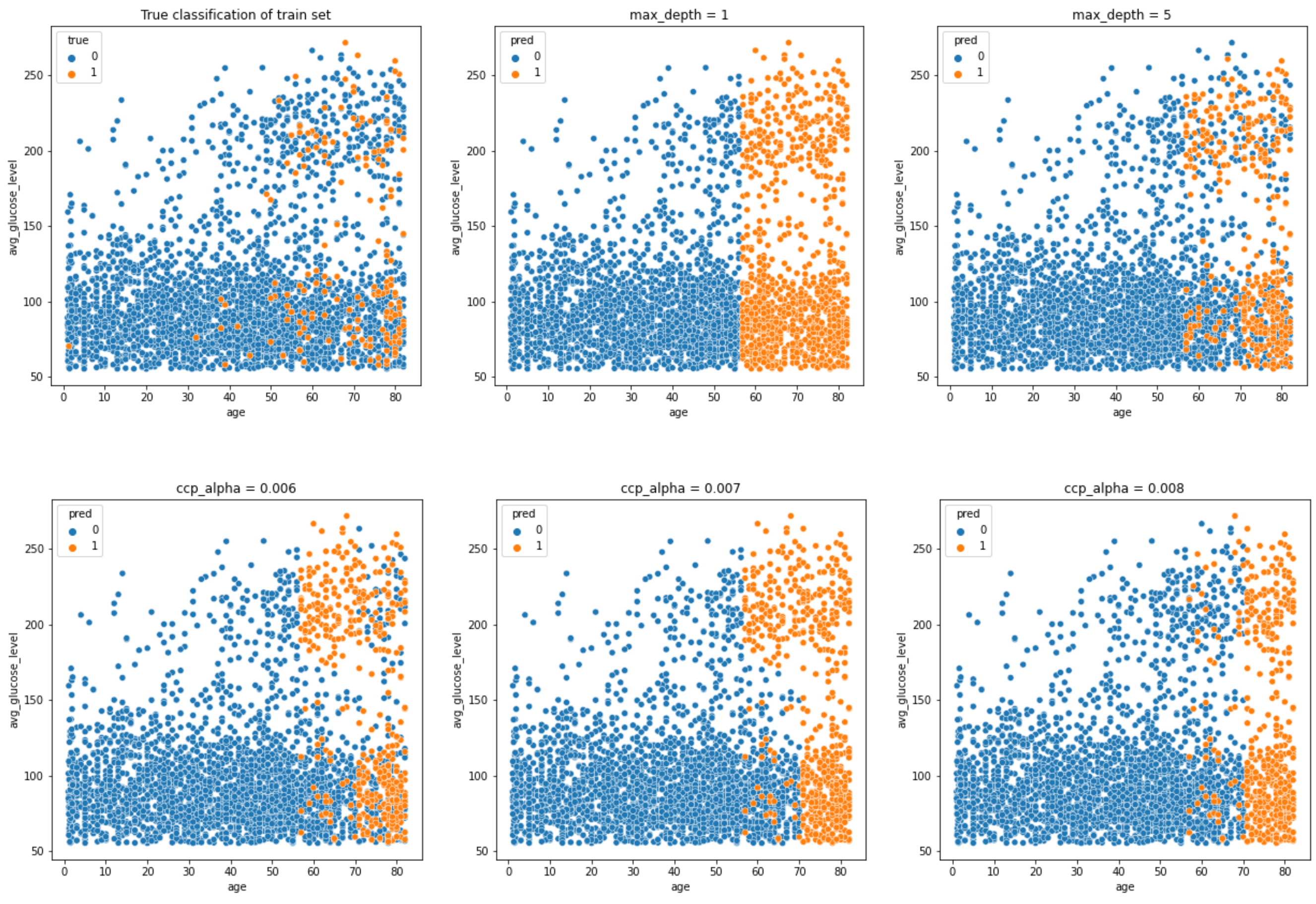


Fig. 11: Scatter plot dataset with `max_depth` (=1 and =5) and with `ccp_alpha` (=0.006, =0.007, =0.008). The top left figure is for true classification of the training set.

With SVM

The implementation is divided into the following steps:

- ❖ Do data preparation as described in the previous section.
- ❖ Test the impact of different data transformation on SVC performance
- ❖ Test the impact of different class weights and comparing them with other sampling methods
- ❖ Tune model and plot the results for the training set and validation set.

As we describe in the data preparation part, for SVM we use a decision tree model to handle missing values. The decision tree model will learn from the known instance of `bmi` with 2 attributes `age` and `gender`, therefore predict the missing values. It's still debatable whether the use of decision trees for imputation is the most effective method, but we still want to try it as the missing values can impact the accuracy and classify ability of SVC.

Our first experiment on SVM is comparing different data transformation techniques. As you know, SVM requires that each data instance is represented as a vector of real numbers. In Fig 11, we had tried 4 combinations of 2 options for each numerical and categorical encoder. The result is not surprising as the Standard-Nominal transformer has the highest score mean with the lowest variance. This can be explained by the characteristics of our dataset with all nominal categorical attributes.

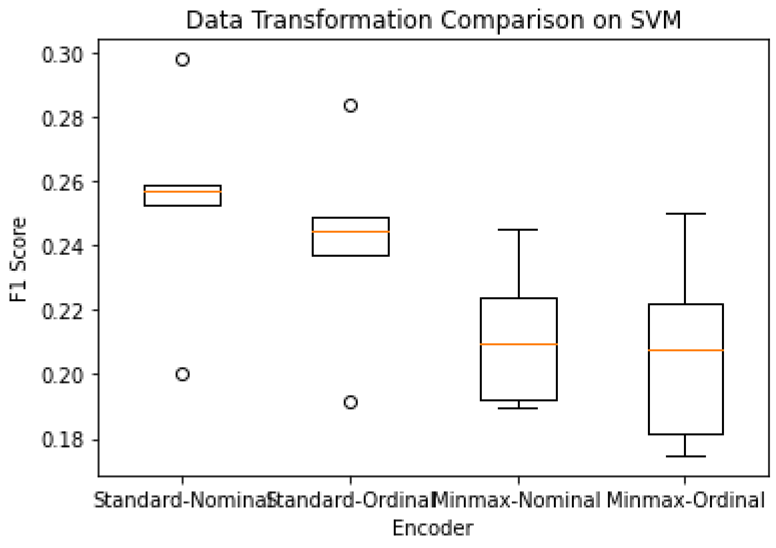
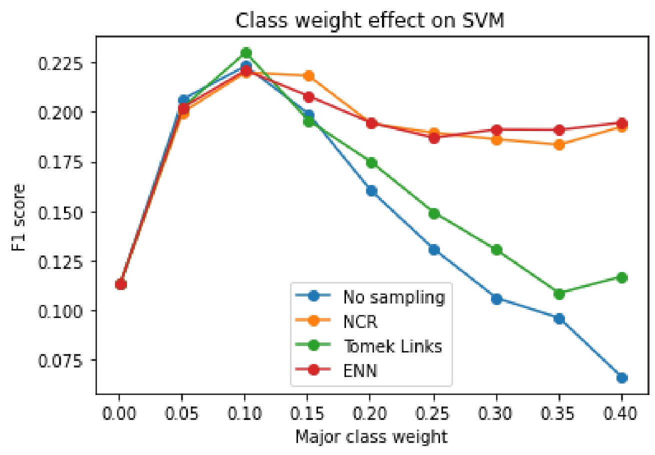


Fig11. Impact of different transformation methods to SVM model. Each pair of encoders is the abbreviated name of a numerical - categorical transformer (Please check out svm.ipynb for more details).

As mentioned in Analysis, our dataset faces a problem of being severely imbalanced. To tackle this issue, we have 2 ways. The first one is sampling the dataset like CART or MLP, we can use some over-, Undersampling techniques, or both of those, like SMOTE ENN in Fig 13. - all of the sampling techniques are better than no sampling. The second one is modifying the class_weight in SVM, which is considered a useful way to give more bias to the minor class. In figure 12., we've tested different class weights and combined them on some under sampling methods.



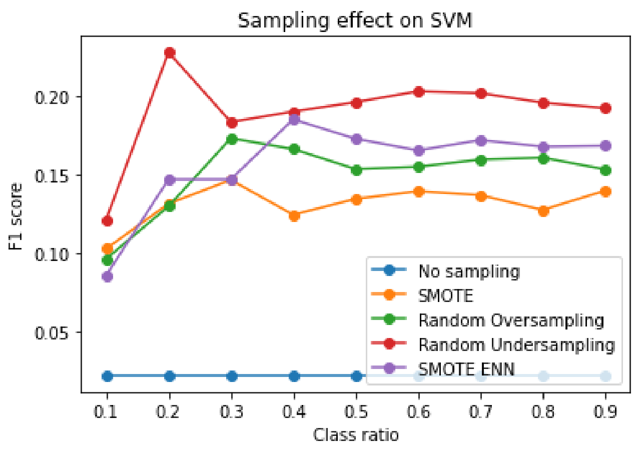


Fig 12. Class_weight effect on SVM with some undersampling methods

Fig 13. Different sampling effect on SVM (class_weight is set to default)

The result is quite surprising that the maximum score of using class_weight is slightly higher than that of sampling methods. And because the sampling methods can cause more computation and information losts, we decided to use class_weight = 0.1 to handle imbalances in SVC for the rest of the project.

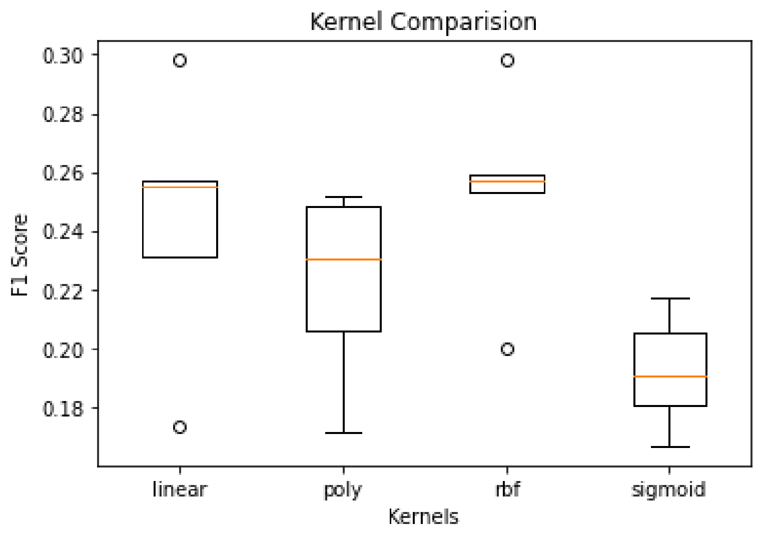


Fig 14. Kernel comparison with data transformation and class_weight =0.1

After experimenting with different data preparation techniques, we're ready for tuning our SVC model. Fig 14. is the comparison between 4 basic kernels, as you can see, the Linear and RBF kernel have the higher f1-score. However, the RBF kernel is more stable with extremely low variance. We will try to tune the parameters for both of those: Linear and RBF

There are 2 parameters for RBF kernel: C and γ. We are recommended to use GridSearch together with k-fold Cross-validation for tuning these parameters, also . The result in Fig 15a makes us very confused when many values are the same. What next? With all the similar results, how can we evaluate which one is the best? To tackle this issue, we've chosen 3 pivot values representing 3 highest areas and compared them by k-fold cross validation (Fig 15b.). After determining the better one, a finer grid searches more precisely on that region to make sure we found the best parameter (Fig 15c.)

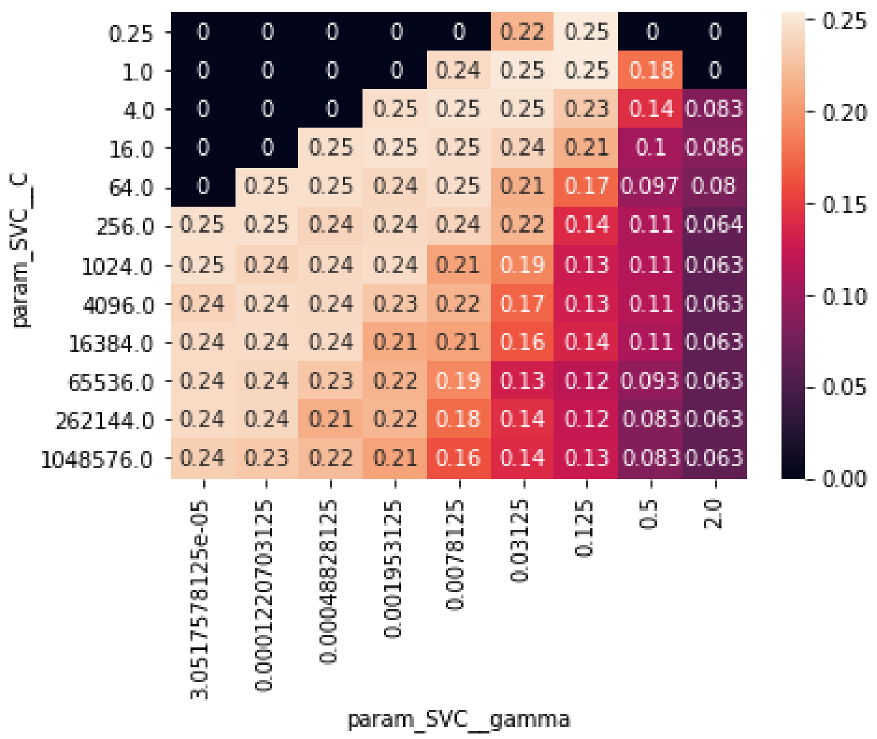
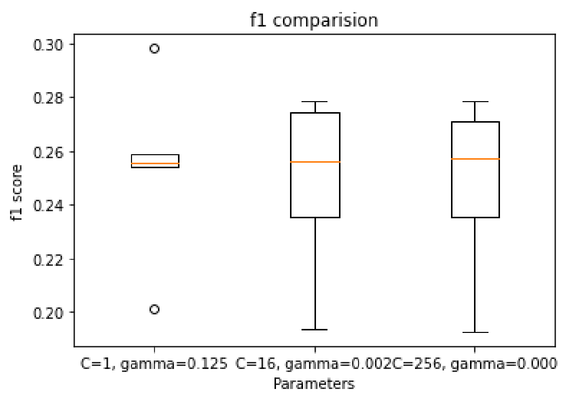


Fig 15a. Grid Search on

C = 2^{-2}, 2^{-4},... 2^{20}

and γ = 2^{-15}, 2^{-13},... 2^{1}



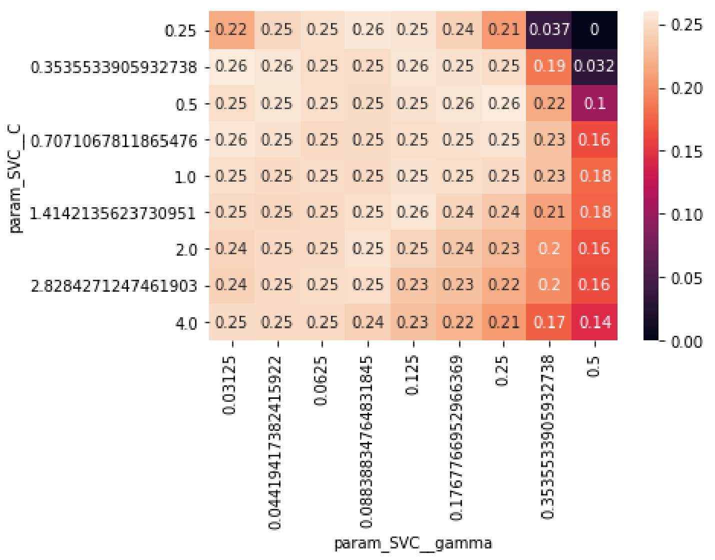


Fig15b. F1 comparison of 3 pivot parameters.

C =1 is chosen for the lowest variance

Fig 15c. Fine grid search with C = 2^{-2}, $2^{-1,5}$,... 2^{2} and $\gamma = 2^{-5}$, $2^{-4,5}$,... 2^{-1}

After tuning for RBF, we continued to do the same thing for the linear kernel (Fig 16.) and compared the result of the two (Fig 17.). The Linear kernel sometimes could result with a pretty high score like 0.3, but sometimes it could be 0,2, which is very unstable. The RBF kernel with its tuned parameters would be our final choice for the SVM model.

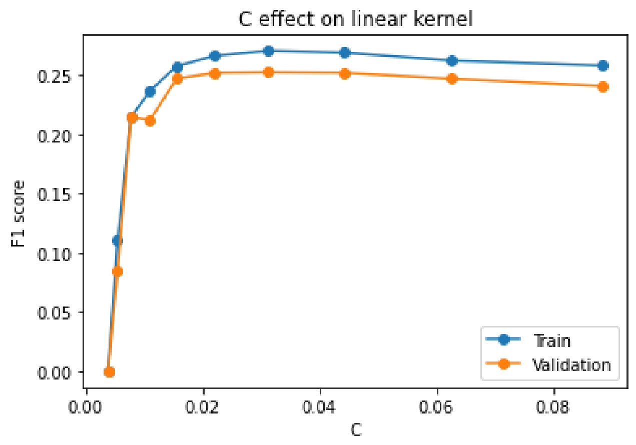


Fig 16. C tuning for Linear kernel

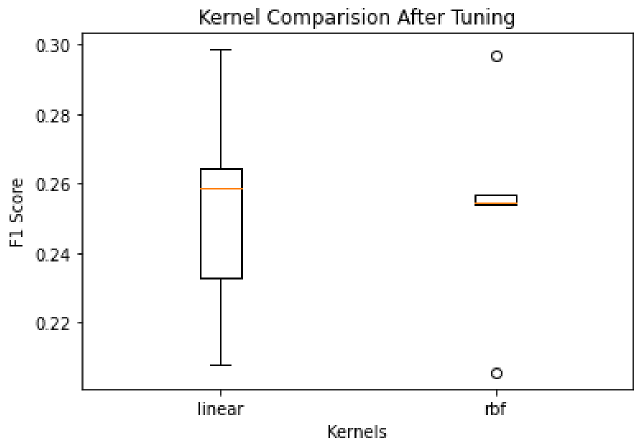


Fig 17. Kernel comparison after tuning. Both both have the same mean but significant different variance

With ANN - MLP

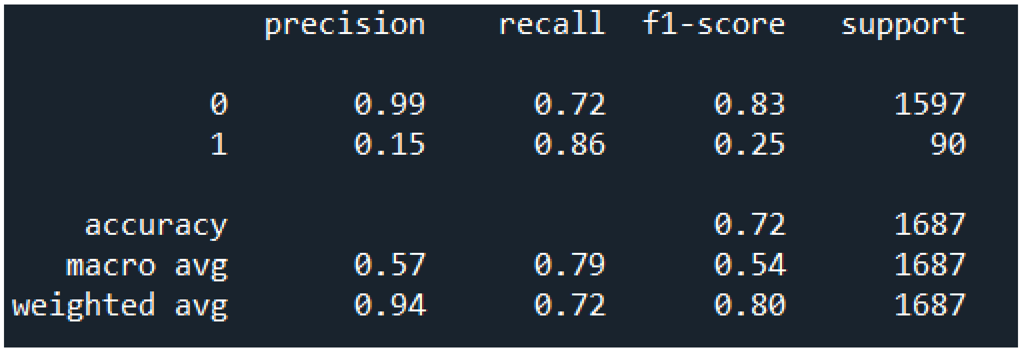
The implementation is divided into the following steps:

- ❖ Do data preparation as described in the previous section.
- ❖ Test the impact of different data sampling techniques as described in the previous section.
- ❖ Tune model and plot the results for the train set and validation set.

Basically, the Data-Preprocessing steps of MLP Classifier are similar to those of CART. However, as there are differences between models, the overall results and performance time are various.

MLPClassifier.1: Summary Results:

Compare with other algorithms, the Classification result of ANN is considerably decent:



```
              precision    recall  f1-score   support

           0       0.99      0.72      0.83      1597
           1       0.15      0.86      0.25        90

    accuracy                           0.72      1687
   macro avg       0.57      0.79      0.54      1687
weighted avg       0.94      0.72      0.80      1687
```

Train time for ANN: 18.270166873931885

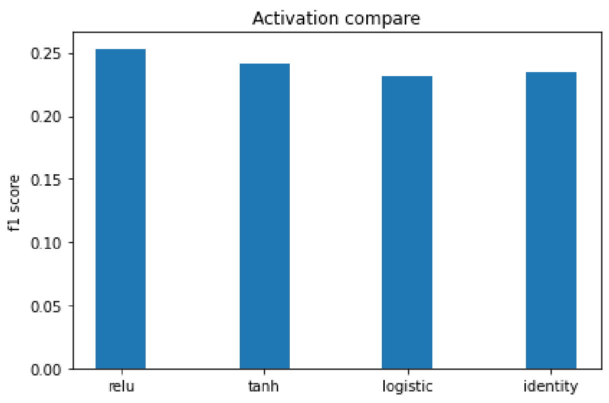
Predict time: 0.12898492813110352

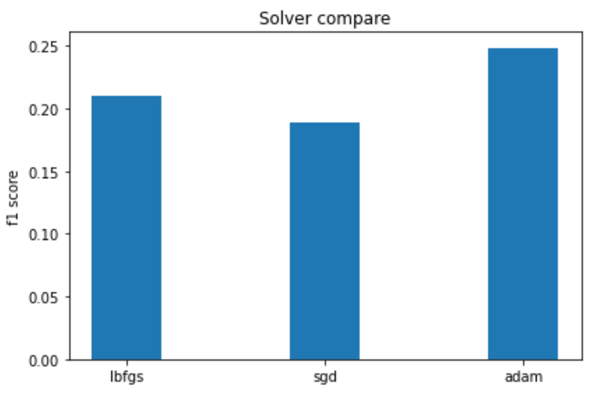
With class "0": The precision reached nearly absolute rate: 0.99, f1_score of this class is niche too. This is expected because the dataset is seriously unbalanced even though Undersampling has been applied.

With class "1": As a consequence of an unbalanced dataset, the minority class has quite a disappointing score. The precision only reached 0.15 when the f1_score - the main metrics measure - only reached 0.25. However, this f1_score is remarkable compared to other models of DecisionTree and SVM.

MLPCLassifier.2: Data Preprocessing and Model Tuning:

1. Data Preprocessing: This step is almost similar to the Decision Tree Model (CART).
   Data is split to train set and test set, then we handle missing values, outliers, normalize and encode related features. Basically, different data processing methods do provide differences in results, but due to the enormous dissimilarity between two classes, those differences caused by methods are insignificant - except for imbalance handling.
   About imbalance handling, those methods include Over-sampling and Undersampling that potentially of making a boost in model result. We combine the observation of those methods with the observation of Model-Tuning for time-saving reason.
2. Model Tuning:
   There are various parameters for ANN to tune. However, many of them don't contribute a considerable change to the model's result. For example: the accuracy given by different activation functions or solver parameters are barely changed. (See diagrams below).





In order to make it focused and intuitive, we will only pay attention to two most efficient parameters: hidden_layer_sizes and max_iter.

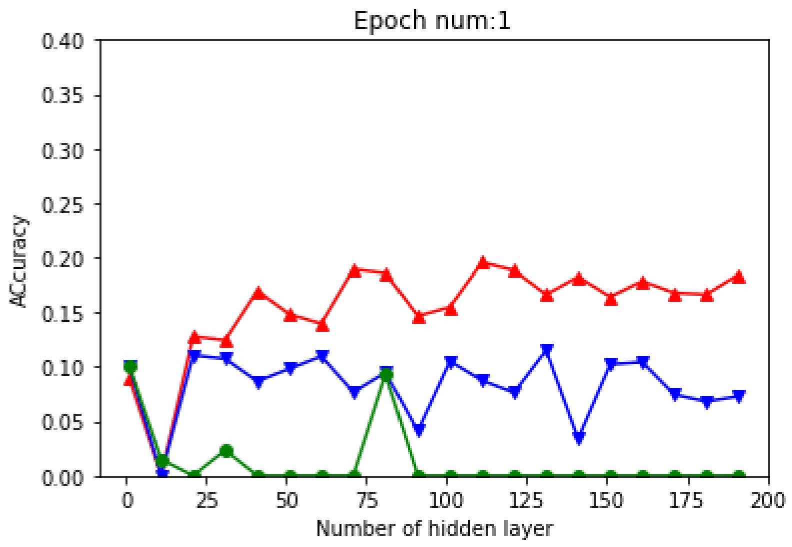
+ What is hidden_layer_sizes and max_iter:
  Hidden_layer_sizes is the number of hidden layers in the structure of MLP Classifier. Theoretically, the larger the number of hidden layers, the better MLP will learn on training set. But too many hidden layers will lead to overfitting which makes the accuracy of model decrease. Our task is to determine the ideal number of hidden layers for our dataset.
  Max_iter is the maximum number of iterations. The solver iterates until convergence or this number of iterations. Similar to hidden layer sizes, MLP will learn better with more iterations but it should stop at some points to prevent overfitting as well as enough to ignore underfitting.

+ How do we tune this model:

We combined the experiment of testing hidden layer sizes, max iteration with testing the effect of Oversampling and Undersampling.

For Over-Sampling, we applied SMOTE (mentioned above in CART), for Undersampling, we applied RandomUnderSampler. Both these methods are implemented from imblearn library.

The graph gif below provided the visualization of SMOTE, RandomUnderSampler and original process over a number of hidden layer sizes and max iterations:

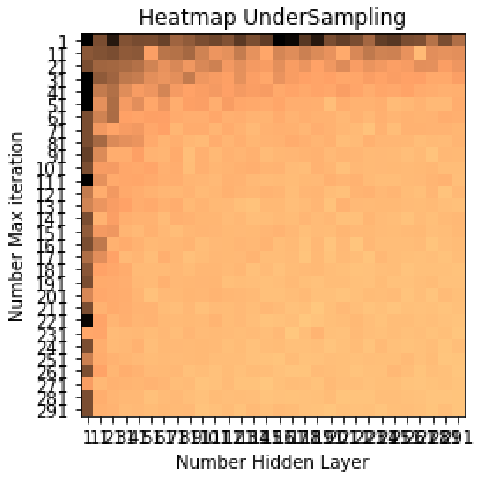


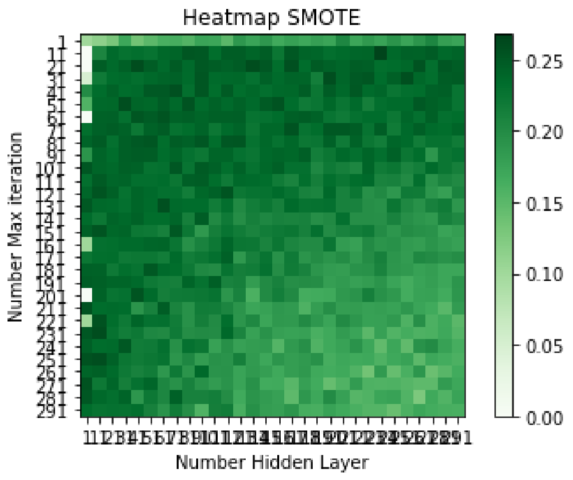
Note: Red-line: SMOTE Oversampling
Blue-line: RandomUnderSampling
Green-Line: Original model

Clearly, handling imbalanced data by Oversampling and Undersampling brings a significant boost in model performance. By default, the model cannot classify almost every instance. But with imbalance handling, the f1-score increases effectively.

Aside from that, along with the growth of max iterations, Undersampling seems to be more adaptive and stable than the Over-Sampling. However, both methods still maintain the performance result at around 0.25 for f1-score.

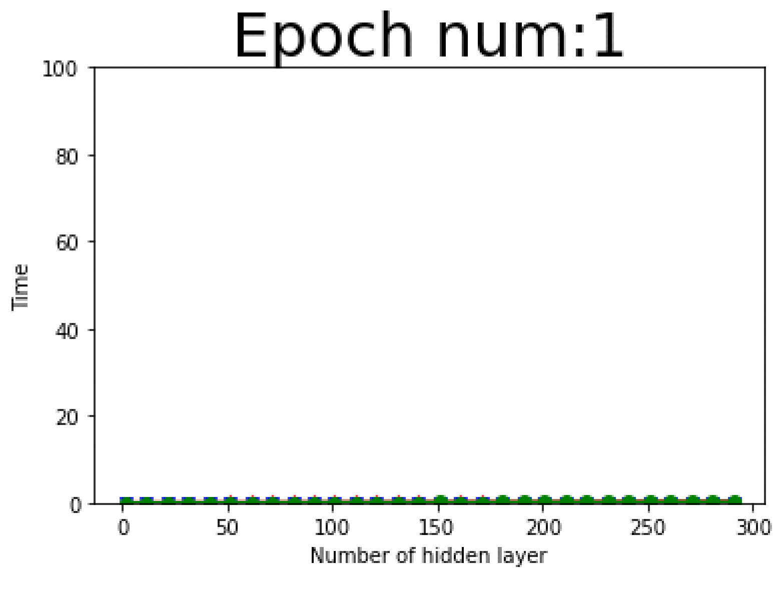
Heatmap for SMOTE and RandomUnderSampler:





The stability of two methods can be seen vividly by the heatmap: with Undersampling, the bottom right corner is more fluid while that region of SMOTE is chaotic.

In spite of having equivalent accuracy, the Undersampling method shows a superior performance in training time.



According to the graphs: At all the number of hidden layers or iteration, Undersampling always runs faster and much faster than both original and SMOTE models. This can be explained as the algorithm of Undersampling reduces training data size by cutting down the amount of majority class. Compared to Over-Sampling, the data size can be reduced by 10 times or more. This difference in training size causes a big change in training time.

In summary, Undersampling is doing great in this problem of classification compared to Over-Sampling and original methods. However, this method of imbalance handling might cause confusion in future learning as its training set is smaller than the actual train set. In the scope of this project, we came to a conclusion that Undersampling is best of 3.
About the MLP model parameters, the ideal range for hidden layer sizes as observed above is around(150-200) along with the max iteration of around (150-170). At this tune, the MLP Model with Undersampling data handling runs with acceptable accuracy and run time.

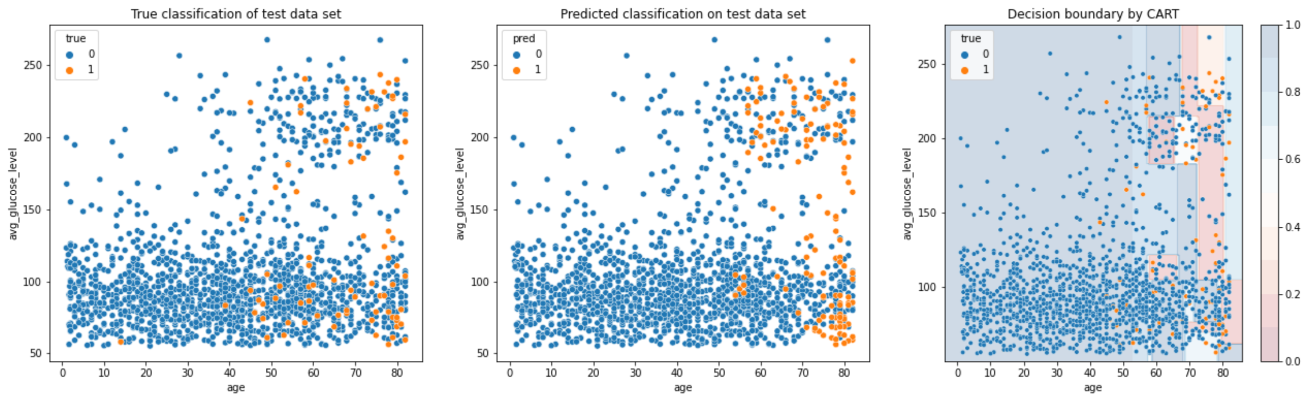
Results

Model Evaluation

With CART, on the final test set, the classification report is described as below:

```
CART training time: 1.12
CART predicting time: 0.03
              precision    recall  f1-score   support

           0       0.96      0.94      0.95      1589
           1       0.20      0.30      0.24        82
```



ANN

```
ANN training time: 3.67

ANN predict time: 0.08

              precision    recall  f1-score   support
```

```
          0        0.99      0.72      0.83      1597
          1        0.15      0.86      0.25        90
```

SVM

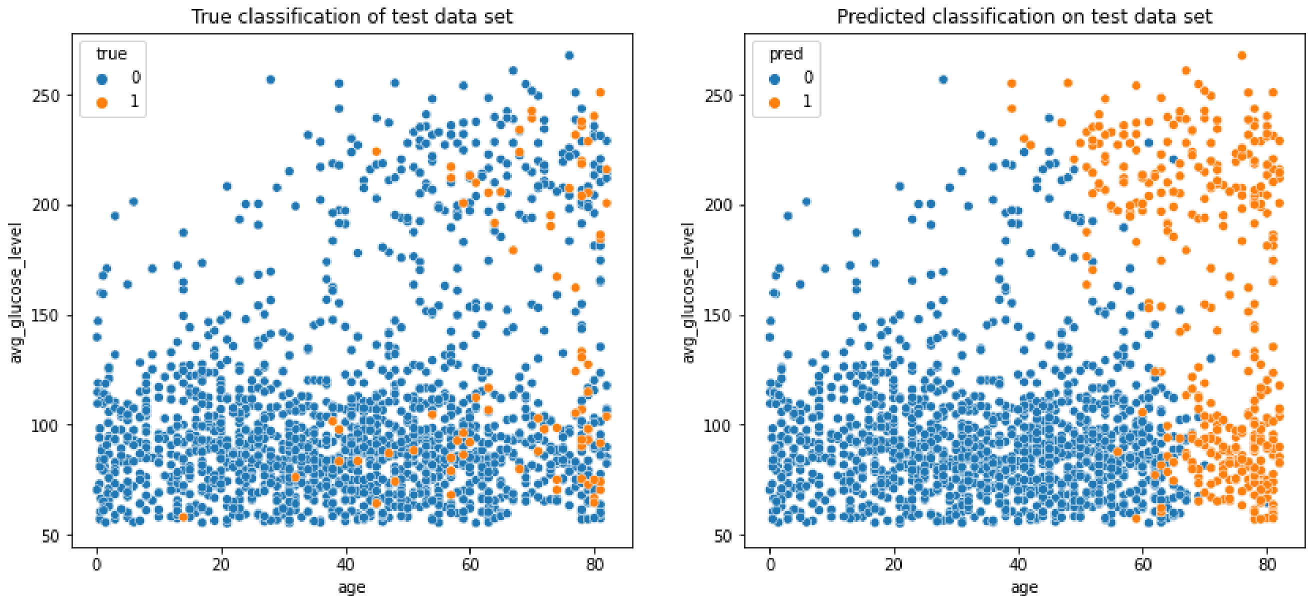
```
SVM training time: 0.48

SVM predicting time: 0.24

              precision    recall  f1-score   support

          0        0.98      0.80      0.88      1604

          1        0.15      0.72      0.25        82
```



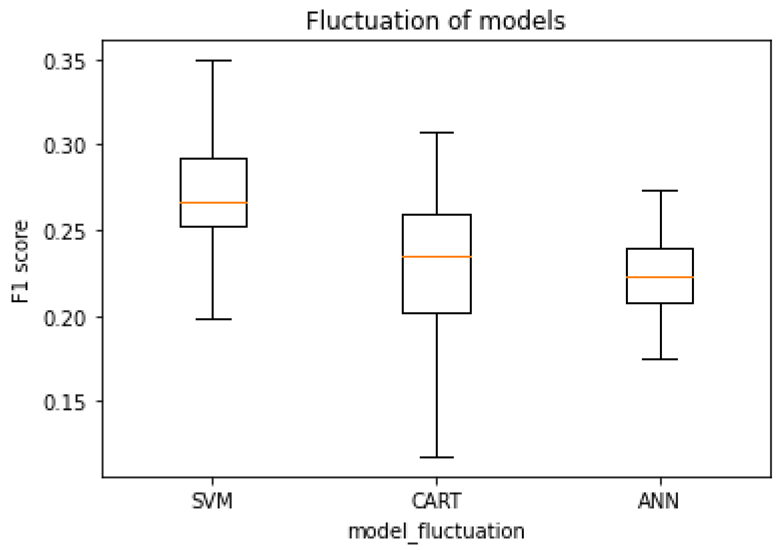
Explanation

Though we exploit many techniques to process imbalance data and configure models, our overall F1 result does not surpass 0.3, which is fairly low. One explanation for this situation could be the reality that the stroke dataset is highly overlapping. Overlapping might also be the reason why oversampling doesn't help in improving model performance. If we use smote alone, the generated minority instances might overlap even more, decision boundary becomes more blur and this might degrade model performance. In the Fig. 1, we can clearly see that oversampling shows some sign of degrading model performance as we increase the sampling_strategy ratio

Conclusion

We have conducted 3 approaches: CART, SVM and ANN. The basic results of all 3 approaches are quite poor: f1 score lower than 0.3.

Among 3 models of CART, SVM and ANN, the SVM model provide the highest f1 score at around 0.26. However, the 3 models have performed similarly as there's no model that can totally overtake others, even SVM. To point out one best model for this problem, the SVM might be the most suitable with both f1 score, recall and precision at a nice cap. On the other hand, CART model has quite a fast prediction time compared to SVM and ANN. This information might be useful in future use when we need to make a massive number of decisions. With ANN, one of its strengths is stability: while SVM provides some good scores, its standard deviation when considering stability is 0.033 - higher than that of 0.022 of ANN, and this situation is even worse with CART with 0.045. Nevertheless, ANN and CART models can potentially be improved in the future where more aspects of learning and parameters are understood better.



Our aim is to achieve Recall as high as possible, i.e the estimator would not ignore any positive patients. However, with regard to the data distribution and the fairly low performance of the estimator, it might be the case that there is no way to know if a patient is likely to suffer from stroke or not. Results attained by classification models don't seem to be reliable even if we had more data.

The dataset has one other problem about age distribution: the age feature focuses mostly on old people that leads to considering young people as noise when actually the situation of stroke on young people has been increasing noticeably recently. This is quite unfair when the young people are actually the ones who are more necessary to be predicted for preventing stroke. Old people are important too, but in fact they have more time for traditional medical methods and personal health care. This point of view empowers the larger conclusion that <u>this dataset is not yet ready</u> for practical use.

Another comment about the dataset is that its output format of 0-1 barely brings reliance to users. For example, some notices to patients that are "You might suffer a stroke in the future" or "You might not suffer a stroke in the future" have fairly low insurance and are not reliable enough. Instead, if the output is continuous, for example: a float number between 0 and 1- the rate of having a stroke in the future, the situation might get better: The model gives a predict of stroke probability on users, and let them consider themselves if that probability is high enough - or dangerous enough. This change in output type might push the dataset closer to practical use.

After trying 3 different approaches but still the results haven't been improved nicely, we have come to a conclusion that <u>based on this dataset</u>, Stroke prediction in practical use might be not reliable and precise enough, specially in this case when making a prediction of medical status is very sensitive. However, if seriously in need, the dataset and some models can provide some useful information but still limited since the recall scores of SVM and ANN are quite acceptable.

Look at the bright side, even when the models are not ready to use, we have obtained some good experiences during project time.

- ➢ Firstly, when dealing with seriously imbalanced data, we have to search for the solution to this problem and find the Imbalance Handling technique. And clearly this technique has significantly improved all of our models. This experience in handling unexpected situations is absolutely useful for us in the future.
- ➢ One other benefit we received during the project is the skill of visualizing data and experimental results as well as the ability to do researches on other articles, libraries and documents.
- ➢ We also have to draw comments on our dataset, our practical problem to make decisions during the project (Using f1 score for measurements instead of accuracy, focus on recall rather than precision rate).

Along with some useful skills and experiences, we still struggle with some hardships:

➢ Most of us haven't got used to the workflow and team working on building a model. The difficulties in handling unexpected problems make us confused for a while, also this is our first project so we aren't sure if the results are good enough, or if the data is handled correctly.
➢ Another thing is that our report and experimental results are not in a good technical writing and can lead to misunderstanding for readers.

Improvement

Due to limited time and experience, we've just explored the basic fields of machine learning. There are areas that can be considered to improve prediction results. If we have more time, we'll try ensemble learning to get not only a more precise prediction result, but also a more stable estimation. In addition, ANN has a lot more to explore and we haven't researched batchsize, learning rate, etc yet. With the surprisingly high Recall, we believe if we study more on ANN, we can improve precision to get a higher overall F1 score.

We are promoted by the topic of EDA, where the procedure includes discovering patterns, spotting anomalies in the data with the help of summary statistics and graphical representations. In the future, we'll dig deeper into this area because none of our features does a good job of separating classes.

References

Stroke Prediction Dataset | Kaggle

sklearn.neural_network.MLPClassifier — scikit-learn 0.24.2 documentation

3. Undersampling — Version 0.8.0 (imbalanced-learn.org)

https://imbalanced-learn.org/stable/over_sampling.html

https://imbalanced-learn.org/stable/combine.html

https://towardsdatascience.com/one-hot-encoding-is-making-your-tree-based-ensembles-worse-heres-why-d64b282b5769

__ The End __