COMP9417 PROJECT REPORT

Life Insurance Evaluation

Group No. 29

Bingyang Li, z5275077

Hans Quan, z5276952

Zicheng Qu, z5092597

Zirui Yan, z5234037

Table of Contents

COMP9417 Group Project Report 1
1. Introduction 1
2. Implementation..... 2
   2.1. Tools and environments 2
   2.2. Comparison of models 2
   2.3. The final project structure and model 5
   2.4. Method Overview 5
3. Experimentation 6
   3.1. Feature Engineering 6
   3.2. Linear Regression, Lasso, Ridge 6
   3.3. Decision Tree 8
   3.4. Naive Bayes 8
   3.5. Logistic Regression 9

3.6. SVM and LinearSVC 11

3.7. KNeighbors Classifier 14

3.8. MLP Classifier 15

4. Conclusion and future work 16

5. References 16

6. Appendix 17

COMP9417 Group Project Report

1. Introduction

Insurance as a method of risk management is generally accepted in society. We regard insurance as reasonable risk avoidance for ourselves and our financial. For individual insurance, insurance is roughly divided into property insurance, personal insurance, and financial insurance. In this project, we make prediction model around life insurance.

Regarding life insurance, the application process is complicated, and the application cycle is long. For instance, customers need to provide a large amount of information to assess eligibility and classify risk. Medical examination is required during the application process. The whole process will take about a month. With the popularity of online quick shopping and more and more detailed types of insurance, manual audit and assessment of classified risks to find suitable insurance types for customers can no longer adapt to the fast-paced society.

On the Kaggle platform, Prudential Life Insurance Assessment provides one idea to use machine learning to predict which types of insurance customers would be eligible for based on their information. The goal of the challenge is to achieve a classification model that accurately classifies risks. Kaggle provides us with more than 50,000 customer desensitization information as training set. Our task is to build models that predict risk types and to find an optimal model to ensure higher prediction accuracy. In this project, our risk categories(Response) are expressed as an integer from 1 to 8. We will try different classification models (Linear Regression, Decision Trees, Logistic Regression, Naïve Bayes, Neural Networks, etc.) and our evaluation is combined accuracy_score, precision_score,recall_score and f1_score and quadratic weighted kappa provided by Kaggle together to evaluate the quality of models.

2. Implementation

2.1. Tools and environments

The project we chose was to do a quick assessment for the insurance, which required reasonable predictions between a total of eight different levels, so it was a multi-classification machine learning problem. We discussed mane machine learning models based on sklearn, or neural networks or deep learning based on PyTorch or TensorFlow. In the end, we decided to use sklearn as our main model implementation approach. The primary reason is that sklearn contains many models learned in this course, which can facilitate team members to learn more effectively, compare the characteristics of various of models, and deepen their understanding of models.
The packages used in our insurance prediction project include NumPy, Pandas, Matplotlib, Seaborn, and sklearn. The programming environment is Anaconda integrated environment based on Python 3.8.

2.2. Comparison of models

The learning algorithms (models) we used in this project are listed below:

Table 1: 2.2 Learning algorithms overview

Models	Brief Introduction	Parameter	Role	Evaluation
LinearRegression, Lasso, and Ridge	LinearRegression: The output of **wx** + b is continuous. Mainly used for regression problems. Lasso and Ridge: L1 and L2 Regularization are used for Lasso and Ridge, respectively.	The regularization strength alpha.	Alpha can impose penalties on the confidence interval, reduce the confidence interval, make some features ignored, and increase the generalization ability of the model.	The accuracy rate is around 35%.

DecisionTree	Suitable for regression and classification. Split the features as different partitions according to Gini or Entropy. Ultimately, it can be understood as a set of if, else if, and else.	min_samples_split	Can ensure each decision has at least the target leaves to do the majority vote. It can effectively prevent overfitting.	The accuracy rate is around 42%.
LogisticRegression	Generally used for binary classification. Can also set multi_class for multi-classification.	penalty	1 or 2 norm penalizations.	The accuracy is around 46%.
		solver	Different algorithms used in our model including liblinear, saga, lbfgs, newton-cg, in order to optimize the problem.	
BernoulliNB and MultinomialNB	Mainly used for classification. This works well for data with Bernoulli or Multinomial distributions. Variously used in the text data set.	smoothing parameter alpha	Add smoothing to prevent something that should happen, but didn't occur in the given dataset, and cannot be counted, and result in a 0 result.	BernoulliNB: The accuracy is around 33%.
				MultinomialNB: The accuracy is around 37%.
KNN	K training samples closest to the testing instance are found in the training data set and do the majority voting as the label for the testing instance.	weights	Whether use different weights depending on the distance or treat them equally as long as they are in the K range.	The accuracy rate is around 36%.
		n_neighbors	The larger n_neighbors, the more stable the model, the smaller its variance and the larger its bias. The larger the N_neighbors, the stronger the generalization ability of the model, but the balance between overfitting	

			and underfitting needs to be considered.	
		p	Whether to use manhattan or euclidean distance.	
LinearSVC	LinearSVC is similar to SVC, but it only supports linear kernel functions and cannot be used to linearly indivisible data sets.	penalty	1 or 2 norm penalizations.	The accuracy rate is around 46%.
		dual	For the optimization of dual and primal.	
		C	It is an inverse of the regularization strength.	
SVC	SVC is an algorithm of SVM, which can adapt to more situations than LinearSVC, but is a little slower to train than LinearSVC.	kernal	rbf kernel used in this model. Mapping from low dimensions to high dimensions, greatly reduces math operations.	The accuracy rate is around 49%.
		C	It is an inverse of the regularization strength.	
MLPClassifier	It is a multi-layer perceptron model. Its main principle is to adjust the weight step by step through backpropagation for gradient descent.	activation	An activation function tanh that turns linear to nonlinear, and compresses the value to (-1, 1).	The accuracy rate is around 49%.
		hidden_layer_sizes	Set the number of layers and the number of perception neurons in each layer.	
BaggingClassifier	Bagging uses MLPClassifier and samples the data set with replacement. Reduce variance without affecting the bias.	n_estimators	The number of MLPClassifier used in this model.	The accuracy rate is around 51%.

2.3. The final project structure and model

By comparing the models above, we finally choose Bagging with MLPClassifier as our final model. The "model_MLP.py" can be run directly and it can automatically call preprocessing.py to acquire the data sets for training and prediction. Similarly, python files for other models can also be run directly.

The final project structure is preprocessing.py to do the feature engineering, model_MLP.py to train and predict the mode, figure_plot.py to plot figures, and test_grid.py is used for GridSearchCV. The generated model and CSV file are saved in the "./Submission" folder, and the figures are saved in "./figures" folder.

2.4. Method Overview

Code modularization can better reuse code, debug, and maintain reliability (Mirzoyan, 2020), so we decided to divide the code into different modules according to different functions. It is mainly divided into the following aspects: data processing, plotting, GridSearchCV, different models (one model in one file).

For each model, each team member conducts analysis and experiments on it according to different situations. On the premise of not overfitting, the model and parameters are optimized from many aspects to improve prediction accuracy as far as possible.
The main methods involved but are not limited to are cross-validation, Grid Search, bagging, etc. In terms of model evaluation, used accuracy_score, recall, precise, f1, and some other aspects for analysis. Since the test standard adopted by Kaggle in this competition is the score, we mainly adopt the score as our evaluation method, which is consistent with the evaluation standard of this competition in Kaggle.
See Experimentation in Part 3 for feature engineering and each model's implementation process and the specific methods involved.

3. Experimentation

3.1. Feature Engineering

- Data Set initialization

According to the information provided by Kaggle, the features are classified into categorical, continuous, discrete, and dummy.

- Split or reconstruct some features

BMI and age are generally taken into account in the rating of insurance eligibility or insurance premium (Croll, 2021). Therefore, we multiplied BMI and age together and removed the original two ones.
By observing the format of each data, it is found that there is a categorical feature, the first part is letters, the second part is numbers. Therefore, the feature is split into two parts and the letters are mapped to numbers.

- Processing missing value

It is often better to fill missing values with mode than with mean for classifications(Badr, 2019). Therefore, we decided to delete those with a large missing proportion, and fill in those with a small missing proportion with the mode of each feature.

- Removes features with high-frequency single value occurrence.
- Identify features with high positive or negative correlations via pairplot, and remove some of them to reduce the correlations.
- Verify the correlation again with the heatmap, decrease the correlation. Since some features are not suitable to be analyzed via pairplot.

All figures related to 3.1 Feature Engineering are placed in appendix.

3.2. Linear Regression, Lasso, Ridge

When we use linear regression, we need to consider whether there is a linear relationship between the dependent variable and the independent variable. When we constructed the linear regression model, the score of model was 0.34. But the test set predicted accuracy is only 0.13. We can infer that Linear Regression does not apply to this dataset. When we extract the features, there are both discrete features and continuous features. And there are many missing values. We discard the column with a large miss ratio, leave the column with a small miss ratio and fill the missing area with the mean value of that column. It's worth mentioning that we also have feature that are strings. As you can see from the composition of the feature, there is no linear relationship here.

As for Lasso and Ridge, Lasso which means Least absolute shrinkage and selection operator. This method is a type of compression estimation. We often need to scale these features before building the model. That is, sum up absolute values of the force coefficient is less than a fixed value; And set some regression coefficients to zero. When we used Lasso to test the data set, we got a predictive accuracy of 0.11 at most. Ridge Regression is a biased estimation regression method for linear data analysis. It is essentially an improved least squares estimation method. Also, in order for the model to work better, we theoretically need to stretch the feature to a certain extent to amplify the characteristics of each value. When we used Ridge Regression to test the data set, we got a predictive accuracy of 0.13 at most. Both Lasso and Ridge, we apply different 14 tuning parameter λ for these two models. Their accuracy didn't change much. Shown as table 3.2.1 and 3.2.2, these models do not predict the results of test dataset very well.

Table 2: 3.2.1 The result of Ridge and Lasso with different lambda value (part 1)

lambda	**0.01**	**0.1**	**0.5**	**1**	**1.5**	**2**	**5**
Ridge	0.1303	0.1302	0.1301	0.13	0.1297	0.1295	0.1288
Lasso	0.1129	0.0952	0.0968	0.0946	0.0935	0.0924	0.0915

Table 3: 3.2.2 The result of Ridge and Lasso with different lambda value (part 2)

lambda	**10**	**20**	**30**	**50**	**100**	**200**	**300**
Ridge	0.1284	0.1275	0.1264	0.1254	0.1238	0.1218	0.1198
Lasso	0.0915	0.0915	0.0915	0.0915	0.0915	0.0915	0.0915

3.3. Decision Tree

Decision Tree model is more in line with the way people think about classification problem. Decision tree will make a judgment on the characteristics of samples from different aspect of the problem, and then conclude the category of sample. When using decision tree, one should consider avoiding over-fitting problems as much as possible. First we have an experiment for cross-validation(shown as table 3.3.1).

Table 4: 3.3.1 The result of the experiment for cross-validation

fold	10	20	40	60	80	100
CV score	0.4056	0.4063	0.4049	0.4091	0.4058	0.4063

For Decision Tree, we set stopping criterion to 14 different values and criterion='entropy' and random_state=0. According to different stopping condition, we get prediction accuracy and plot it by legend. **The figure are placed in appendix named figure 3.3.1.**

We note that in the process of decision tree growth, it is easy to occur over-fitting, resulting in low generalization ability.

3.4. Naive Bayes

Regarding the naive Bayes model, we tested two of them: Multinomial distribution Bayes and Bernoulli naive Bayes. If the distribution of the actual data conforms to the multinomial distribution or Bernoulli distribution, the effect of using the corresponding classifier will be better. We first used the default parameters to train the model, and the results are as table 3.4.1:

Table 5: 3.4.1 The results of naive Bayes regression

classifier	train accruacy	test accruacy	recall	precision
MultinomialNB	0.37246	0.36339	0.17841	0.21601
BernoulliNB	0.33768	0.3328	0.13778	0.10594

From the results, we can conclude that the data in our training set may not conform to Gaussian distribution or multinomial distribution, because the accuracy rate is much lower than that of other models. We continue to take the range of alpha and want to find the most suitable value. After filtering, we got the corresponding data shown in as table 3.4.2:

Table 6: 3.4.2 The results of the best alpha value chosen

classifier	**train accruacy**	**test accruacy**	**recall**	**precision**	**best alpha**
MultinomialNB	0.37512	0.3699	0.18176	0.2075	41.98707
BernoulliNB	0.33762	0.33291	0.13791	0.23095	8.30994

In this process, it is worth noting that the naive Bayes model runs much faster than logistic regression, which reflects the characteristics of small time and space overhead in the classification process.
We took 200 numbers in the range of 10^-2 to 10^5 as the value range of a. The result displayed by the code shows that even if the highest accuracy a value is taken, its accuracy still does not exceed 0.4. At the same time, we can know from the results that the test data may be closer to the polynomial distribution rather than the Bernoulli distribution, because the result of the polynomial distribution is better than that of the Bernoulli distribution regardless of the default or the highest accuracy. Good. But this method still does not meet our needs.

3.5. Logistic Regression

For logistic regression, we first tested the following five cases, as shown in the table 3.5.1:

Table 7: 3.5.1 The results of logistic regression under five different parameter settings (including train accuracy, test accuracy, recall and precision)

classifier	**parameter**	**train accruacy**	**test accruacy**	**recall**	**precision**
LogisticRegression	penalty='l1', solver='liblinear'	0.46896	0.45795	0.27073	0.33036
	penalty='l2', solver='liblinear'	0.46734	0.45739	0.26977	0.33424

	penalty='l2', solver='saga', max_iter=10000	0.45896	0.45251	0.26634	0.32092
	penalty='l2', solver='lbfgs', max_iter=10000	0.47238	0.46029	0.28070	0.33477
	penalty='l2', solver='newton-cg', max_iter=10000	0.47618	0.46285	0.28439	0.35464

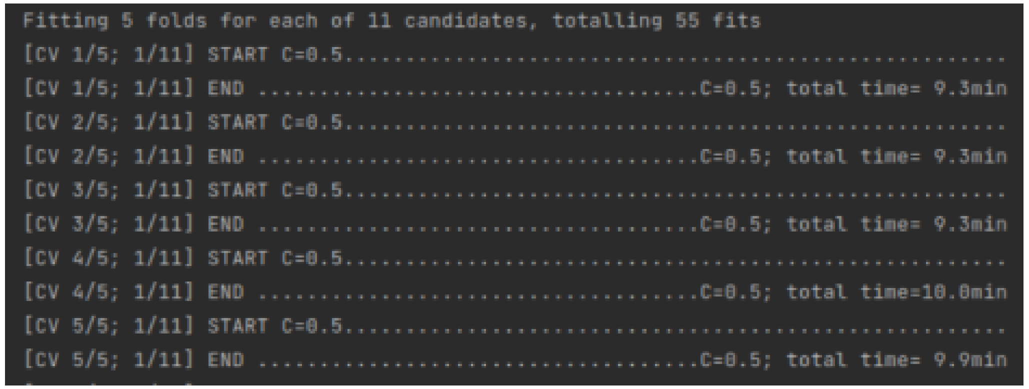
In the program operation, except for the parameters mentioned in the table, we use the default parameters. Before the parameter 'max_iter' is set by us, the program will report "ConvergenceWarning: The max_iter was reached which means the coef_ did not converge "the coef_ did not converge", ConvergenceWarning)" error, indicating that the number of iterations we set is not enough and the parameter is not Convergence, by setting a reasonable max_iter attribute, the model can be converged, so after several attempts we set 'max_iter' to 10000. We found that after the value of the 'max_iter' attribute is increased, the time required for the operation also increases significantly.

From the data, we can see that when penalty='l2', solver='newton-cg', max_iter=10000, we get the highest accuracy, recall and prediction rate, but compared with other models There are still shortcomings, so we continue the experimental process to find more suitable parameters through GridsearchCV. For the case of solver='liblinear', the running time of the program is relatively reasonable, but this is not the case for the other situations. In the left three parameter settings, each fit takes about ten minutes, and it takes a lot of time in the case of using GridsearchCV, so we did not obtain the test results of the remaining three cases, just tested part of their operation time(shown in Picture 3.5.1). The optimal parameters under the tested two parameter settings are as the table 3.5.2:

Table 8: 3.5.2 the results of the best c value chosen by GridsearchCV

classifier	parameter	train accruacy	test accruacy	best c value
LogisticRegression	penalty='l1', solver='liblinear'	0.46882	0.45837	0.6667
	penalty='l2', solver='liblinear'	0.46884	0.45837	1.7778

For these two cases, we adopted a step size of 0.1111 and tested two models in the range of 0.0001 to 1. For the case of penalty='l2', solver='liblinear', we found that the range of c is not large enough, so we tested the range of 1 to 2 again and reached the conclusion in the table. After analysis, we came to the following conclusion: Logistic regression is more inclined to solve the binary classification problem, and the y value of our experiment data has 8 classifications, so the accuracy rate is not particularly high.



```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
[CV 1/5; 1/11] START C=0.5..................................................
[CV 1/5; 1/11] END ...................................C=0.5; total time= 9.3min
[CV 2/5; 1/11] START C=0.5..................................................
[CV 2/5; 1/11] END ...................................C=0.5; total time= 9.3min
[CV 3/5; 1/11] START C=0.5..................................................
[CV 3/5; 1/11] END ...................................C=0.5; total time= 9.3min
[CV 4/5; 1/11] START C=0.5..................................................
[CV 4/5; 1/11] END ...................................C=0.5; total time=10.0min
[CV 5/5; 1/11] START C=0.5..................................................
[CV 5/5; 1/11] END ...................................C=0.5; total time= 9.9min
```

Figure 1: 3.5.1 part of the output of LogisticRegression(penalty='l2', solver='lbfgs', max_iter=10000), costing about 10 min each fit

3.6. SVM and LinearSVC

Compared with the logistic regression model, the SVM model is more complex, SVM uses hinge loss, and the logistic regression is used to logistical loss. The loss function of both models is to increase the weight of data points with a greater impact on classification, reduce the weight of the smaller point of classification. They also can add different regularization terms, l1, l2, etc., so the results may be similar.
“C”, “kernel”, “gamma”are the three important parameters of the SVM model, and the accuracy of different models is obtained by using GridSearchCV. I also use train_test_split to split the training set for model training and prediction.

Table 9: 3.6.1 the accuracy of the different kernels

kernel	C	mean_fit_time	mean_test_score
rbf	5	230	0.47
linear	5	1385	0.48
sigmod	5	105	0.39

You can see that Gauss and linear kernel can get a better accuracy, but in training time, linear kernel is much larger than Gauss kernel, I choose Gauss and linear kernel for further training.
I tried to train Gauss and linear kernel with different C from 0.5 to 20.

Table 10: 3.6.2 Evaluation of Gauss kernel with different C

kernel	C	mean_fit_time	mean_test_score
rbf	0.5	168	0.48
rbf	0.7	175	0.484
rbf	0.9	178	0.485
rbf	1	169	0.485
rbf	5	229	0.473
rbf	10	275	0.46
rbf	15	314	0.453
rbf	20	338	0.449

Table 11: 3.6.3 Evaluation of linear kernel with different C

kernel	C	mean_fit_time	mean_test_score
linear	0.7	328	0.482
linear	1	398	0.482
linear	5	2528	0.482

It can be seen from the table that the linear kernel has little effect on the accuracy of the change of C value, and Gauss kernel has this best accuracy at the time of C = 1. In addition, it is obvious that the training time of linear kernel is significantly higher than that of Gaussian kernel, because linear kernel is suitable for data sets with more features and fewer samples, while Gaussian kernel performs better when the number of features is small and the sample number is large.
Next we compare the two kernels key evaluation score.

Table 12: 3.6.4 Compare the evaluation of these two kernels

kernel	C	train_score	test_score	precision	recall	f1_score	submission_score
rbf	1	0.629	0.491	0.604	0.491	0.529	0.49
linear	1	0.489	0.486	0.608	0.486	0.528	0.47

In general, Gauss kernel are more accurate and efficient than linear kernel.

Next we tried to solve the problem with LinearSVC. Compared to SVM, the loss function of LinearSVC is squared hinge loss, and SVM is hinge loss. LinearSVC is based on liblinear and converges faster than SVM in large amounts of data.
When the sample size is much larger than the number of features, the parameter 'dual' preferred to 'False', while trying different parameters C, as well as the penalty. There is no restriction on max_iter, so that the training is fully convergent.

Table 13: 3.6.5 Test set accuracy under different parameters in LinearSVC model

mean_fit_time	C	dual	penalty	mean_test_score
50.49	0.5	FALSE	l1	0.469
5.06	0.5	FALSE	l2	0.469
48.71	0.7	FALSE	l1	0.469
5	0.7	FALSE	l2	0.469
50.17	1	FALSE	l1	0.469
5.14	1	FALSE	l2	0.469
62.93	5	FALSE	l1	0.469
5.46	5	FALSE	l2	0.469
62.25	10	FALSE	l1	0.469
5.78	10	FALSE	l2	0.469

It is obvious that LinearSVC is more efficient than SVM, and that the penalty = l1 is significantly more efficient than l2, but different penalty and C values have no significant effect on test set accuracy.

Table 14: 3.6.6 Evaluation of LinearSVC

	train_score	test_score	precision	recall	f1_score	submission_score
LinearSVC	0.474	0.468	0.626	0.467	0.518	0.454

3.7. KNeighbors Classifier

KNN model is less complex than SVM, but the amount of calculation is large when the number of features is high. Basically, it does not do train learning, so compared to logic regression and other algorithms, the predicted process are slower.
The accuracy score of the model changed little when using and not using normalization. The accuracy of the training set was 0.44, the accuracy of the test set was 0.34, the training speed was very fast and the accuracy was not high.
After using MaxMinScaler and StandardScaler transforms training set and test set, accuracy improved slightly, but training and predicting time increased significantly. Using StandardScaler is better than using MaxMinScaler, but the difference is small.
I use train_test_split to split the training set for model training and prediction. The main parameter weight is uniform or distance. When weight = 'distance', the training set accuracy reaches 0.99, while the test set accuracy is only 0.37, I do not solve the overfitting problem by increasing the k value.
I found the uniform is clearly better. I tried different k-values and their performance didn't make much difference.

Table 15: 3.7.1 The evaluation result in different k values with transforming by StandardScaler

k	train_score	test_score	precision	recall	f1_score	submission_score
8	0.48	0.39	0.47	0.39	0.42	0.38
25	0.45	0.42	0.61	0.42	0.48	0.38
27	0.45	0.42	0.62	0.42	0.48	0.36

Although a better accuracy can be obtained in this case, the accuracy is still too low. The KNN model is not an ideal choice.

3.8. MLP Classifier

Compared with SVM, MLPClassifier uses the local optimal algorithm, while SVM adopts to the global optimal algorithm, but MLP is suitable for large-scale data sets, but the requirement of computing power are high.
The MLPClassifier model is a combination of multiple perceptrons to achieve nonlinear classification. The 'solver' parameter selects the default 'adam', because it works better on larger datasets and 'lbfgs' is faster on smaller data sets. The ideal activation function was selected by GridSearchCV as 'tanh', and a combination of hidden layers was designed to select an optimal set of parameters.

Table 16: 3.8.1 Major params of MLPClassifier

	max_iter	activation	early_stopping	hidden_layer_sizes	shuffle	solver
MLPClassifier	500	tanh	FALSE	(16,64,128,64,32,8)	TRUE	adam

To further improve model accuracy and reduce variance, I combined BaggingClassifier and MLPClassifier and sampled from train set with putting back to train 10 models, and thus calculated the average values.

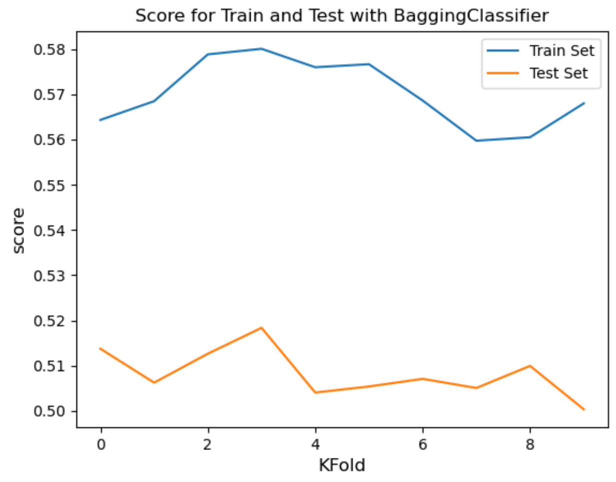


Figure 2: 3.8.2 The train score and test score of 10 bagging models

The accuracy of final submission is 0.51364, which is the best result so far.

4. Conclusion and future work

In this project, we used the training set provided by Kaggle and built a model to predict the type of risk. There is still a certain gap between our model and the best model on Kaggle, but we have hired the knowledge in this course, and selected the best model from the models we have learned, and the results obtained are also within a reasonable range. However, the model we selected and the model with the highest accuracy rate on the website still cannot be used in reality to completely rely on this model to determine whether the customer can be insured, because their accuracy rate is not high enough. Moreover, once the conditions for determining whether a customer can be insured are changed or increased, the models trained with various classifiers require a large amount of data to retrain the model, making it impossible to replace the manual operation with a model at this stage.

In this experiment, the training set and test set we used have been processed, so when we process the data again, we may process some of the processed features again, resulting in inaccurate test data. At the same time, using a single model may also produce some errors. In the future, we can explore more methods to improve the accuracy of prediction, including but not limited to using multiple models to predict together, processing the original data differently, adding some features, etc.

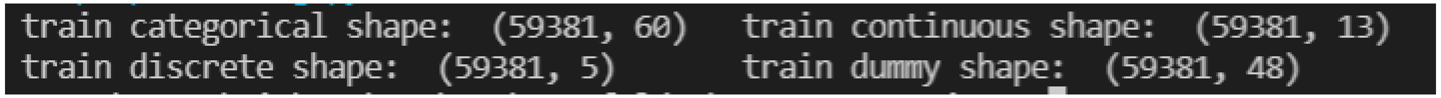
5. References

Badr, W., 2019. *6 Different Ways to Compensate for Missing Data (Data Imputation with examples)*. [online] Towards Data Science. Available at: <https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779> [Accessed 15 July 2021].

Croll, M., 2021. *Life Insurance for Overweight and Obese People: How Your BMI Affects Life Insurance Rates*. [online] ValuePenguin. Available at: <https://www.valuepenguin.com/life-insurance-overweight-obese> [Accessed 12 July 2021].

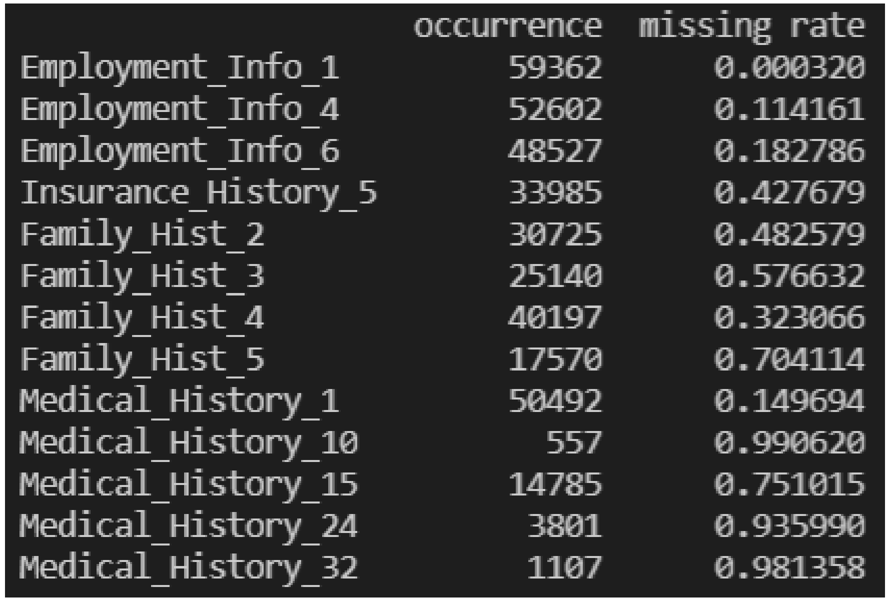
Mirzoyan, V., 2020. *The Importance of Modular Programming*. [online] Aist.global. Available at: <https://aist.global/en/importance-of-modular-programming> [Accessed 8 July 2021].

6. Appendix



```
train categorical shape:  (59381, 60)   train continuous shape:  (59381, 13)
train discrete shape:  (59381, 5)       train dummy shape:  (59381, 48)
```

Figure 3: 3.1.1 Data Set Initialization



```
                     occurrence  missing rate
Employment_Info_1         59362      0.000320
Employment_Info_4         52602      0.114161
Employment_Info_6         48527      0.182786
Insurance_History_5       33985      0.427679
Family_Hist_2             30725      0.482579
Family_Hist_3             25140      0.576632
Family_Hist_4             40197      0.323066
Family_Hist_5             17570      0.704114
Medical_History_1         50492      0.149694
Medical_History_10          557      0.990620
Medical_History_15        14785      0.751015
Medical_History_24         3801      0.935990
Medical_History_32         1107      0.981358
```

Figure 4: 3.1.2 Processing missing value

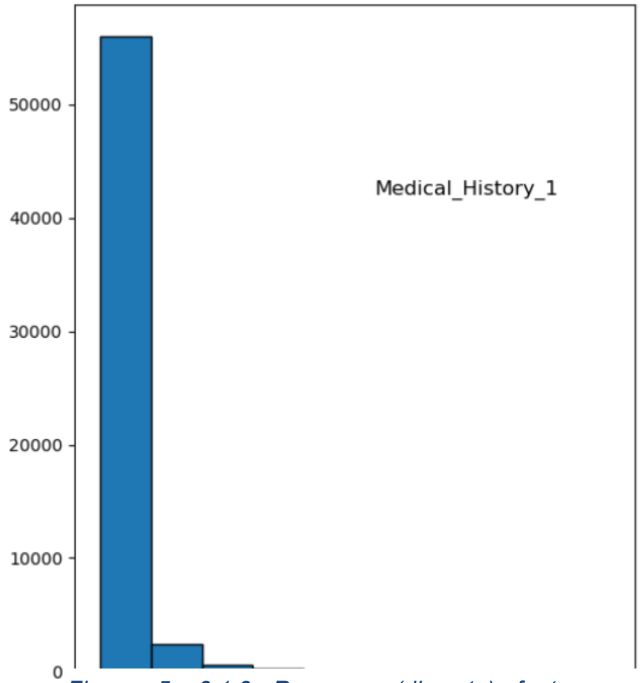


Figure 5: 3.1.3 Removes (discrete) features with high-frequency single value occurrence.

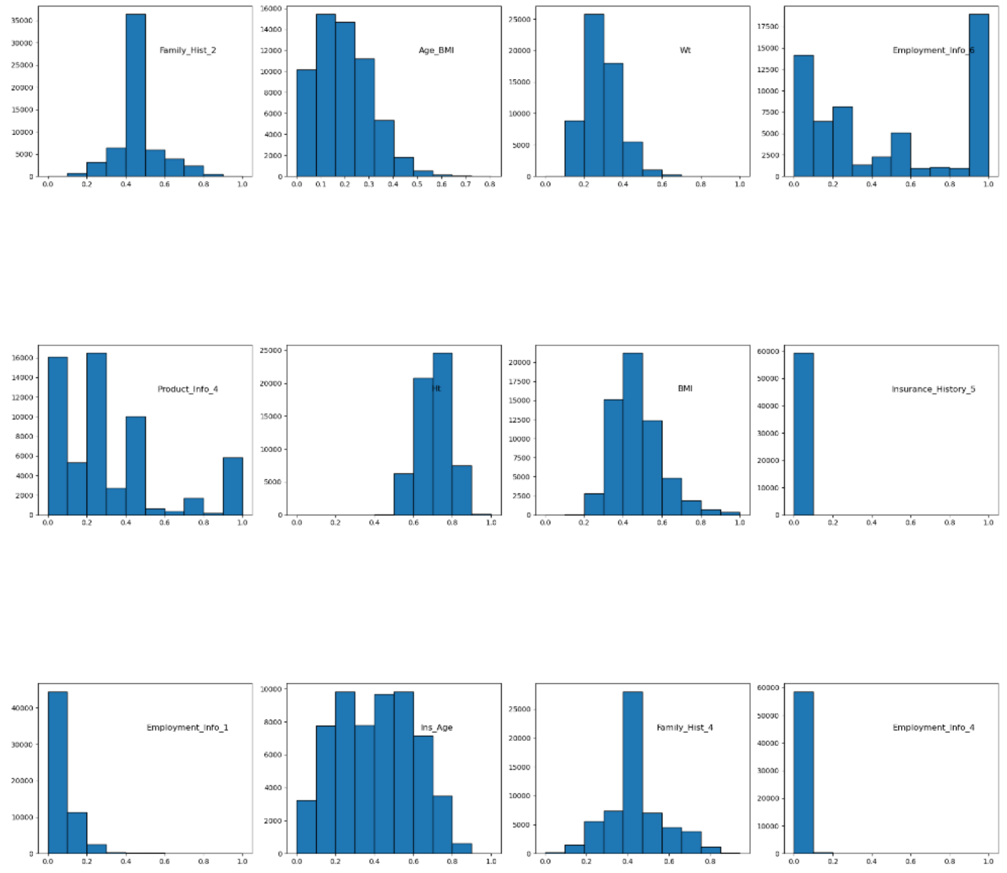


Figure 6: 3.1.3 Removes (discrete) features with high-frequency single value occurrence.

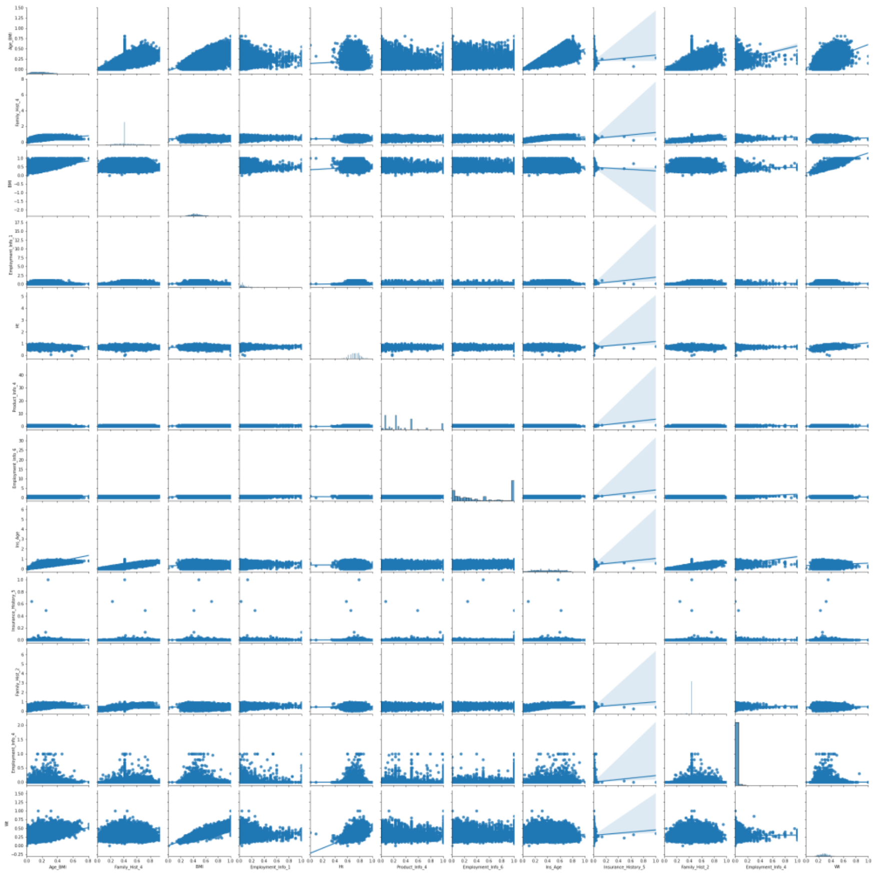
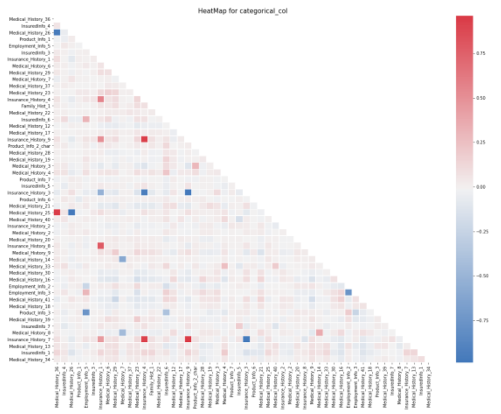
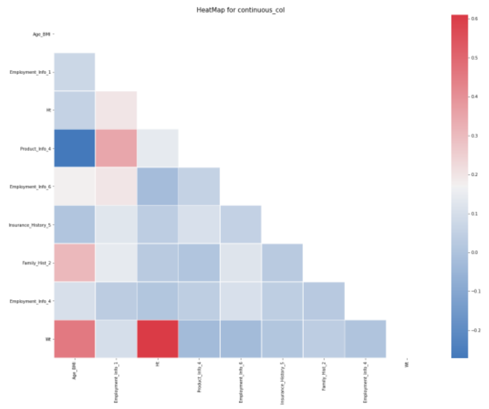


Figure 7: 3.1.4 Correlations via pairplot for continuous features





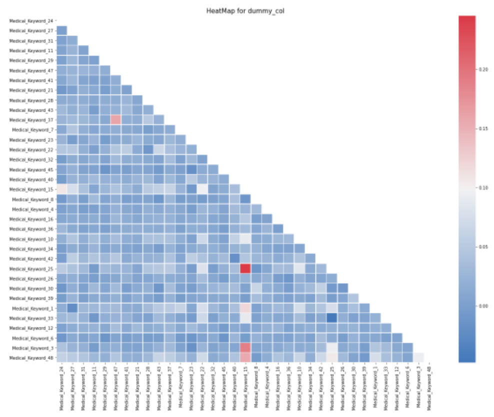


Figure 8: 3.1.5 Correlations via heatmap for categorical. Figure 9: 3.1.5 Correlations via heatmap for continuous. Figure 10: 3.1.5 Correlations via heatmap for dummy features

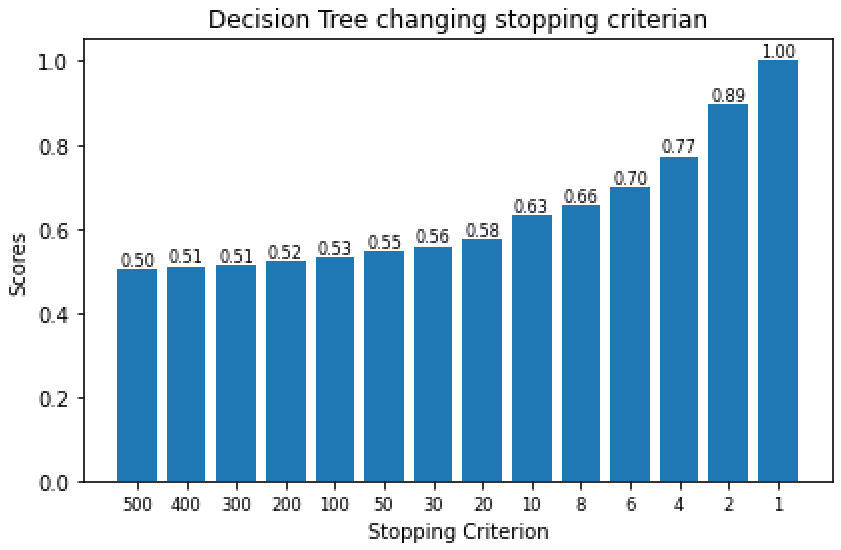


Figure 11: 3.3.1 The plot of prediction accuracy of decision tree