Machine Learning Report

Team: tf pandas*

*Jie Zhou 31384579 jz5n19@soton.ac.uk
*Mingjun Xie 30971586 mx4n19@soton.ac.uk
*Zixuan Cai 31303528 zc1g19@soton.ac.uk
*Chenguang Zhu 31369251 cz1g19@soton.ac.uk

***Abstract*— The data set we are going to explore is cardiovascular disease data set[1] with the basic information below:**

# id	ID number		
# age	Age	Objective Feature	int (days)
# gender	Gender	Objective Feature	1 - women, 2 - men
# height	Height	Objective Feature	int (cm)
# weight	Weight	Objective Feature	float (kg)
# ap_hi	Systolic blood pressure	Examination Feature	int
# ap_lo	Diastolic blood pressure	Examination Feature	int
# cholesterol	Cholesterol	Examination Feature	1: normal, 2: above normal, 3: well above normal
# gluc	Glucose	Examination Feature	1: normal, 2: above normal, 3: well above normal
# smoke	Smoking	Subjective Feature	binary
# alco	Alcohol intake	Subjective Feature	binary
# active	Physical activity	Subjective Feature	binary
# cardio	Presence or absence of cardiovascular disease	Target Variable	binary

Fig. 1. Data set information

After developing the exploratory data analysis, in this report, we implement different models on the cardiovascular data set and make some adjustment on the data and model to increase the accuracy.

I. Introduction

In the last report, we got the cardio data set and developed the exploratory data analysis on it. The report is mainly divided into two parts. First, we use many effective method to deal with the data and show the improvement of prediction accuracy on a specific learning model. Second, we implement many different learning model on the training data and adjust the parameter based on the model principle and data characteristics to make the model perform better.

II. Methods on data set

We will apply the methods described below step by step on the basis of the original training set from the last report. And we prove the effectiveness of these methods by recording the impact of these methods on the accuracy of the KNN model whose parameter k is 3. The accuracy of the KNN model learning original training set is 71.20%(accuracy on training set) and 55.05%(accuracy on test set). It ls clearly that there is a serious overfitting and the test accuracy is low.

A. Remove duplicate incidences in the training set

In the last report, we divide the whole data set into a training set and a test set according to an eight-to-two ratio. The number of incidences in training set is 56000 with 13 attributes. After implementing removal of duplicate values, the incidences of training set reduces to 55984, which has 16 fewer incidences than before. This indicates that there are 17 incidences are totally same at any of 13 attributes. After clearing duplicate incidences, the accuracy of the KNN model is 71.21%(accuracy on training set) and 55.05%(accuracy on test set), which is slightly better than the one of original training set.

B. Attribute combination and outlier removal

From the table shown in Fig.2, we can find that there are some outliers in weight, height, systolic blood pressure(ap_hi) and diastolic blood pressure(ap_lo).

	id	age	gender	height	weight	ap_hi	ap_lo
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000
mean	49972.419900	19468.865814	1.349571	164.359229	74.205690	128.817286	96.630414
std	28851.302323	2467.251667	0.476838	8.210126	14.395757	154.011419	188.472530
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.000000	-70.000000
25%	25006.750000	17664.000000	1.000000	159.000000	65.000000	120.000000	80.000000
50%	50001.500000	19703.000000	1.000000	165.000000	72.000000	120.000000	80.000000
75%	74889.250000	21327.000000	2.000000	170.000000	82.000000	140.000000	90.000000
max	99999.000000	23713.000000	2.000000	250.000000	200.000000	16020.000000	11000.000000

Fig. 2. Describe of the data set

We need to remove the outliers to improve the model performance. But there is a case that some incidences only have one outlier in all its attributes. If we simply delete the incidence, it is wasteful. For the incidence only have one outlier, it is ok to replace the outlier with the median. But we think the better way is that combining the two characteristics of height and weight into BMI as well as combining the systolic blood pressure and diastolic blood pressure into mean arterial pressure(MAP), set the reasonable range of these two attributes accroding to the professional knowledge and replace the outliers with upper limit and lower limit of the range. After implementing these issues, the accuracy of the KNN model is 76.88%(accuracy on training set) and 66.24%(accuracy on test set). We can find that the accuracy of both training and test set increase a lot and the degree of overfitting is reduced. Thus, attribute combination and outlier removal is effective.

C. Standardization

Standardization is a measure that adjust the distribution of feature data to a standard normal distribution, also known as Gaussian distribution, that is, make the data mean 0 and variance 1. The reason for standardization is that if the variance of some features is too large, it will dominate the objective function so that the parameter estimator cannot

correctly learn other features. The standardization process consists of two steps: decentralization of the mean (mean becomes 0); scale of variance (variance becomes 1). Implementing standardization, the accuracy of the KNN model is 78.30%(accuracy on training set) and 67.11%(accuracy on test set), which are slightly better than above.

III. Implementing diffenrent models

In this section, the training set put into the model has been processed through all the methods in section II. We implement different models and adjust the model parameters based on the model and data set characteristics.

A. k-Nearest Neighbor

The core idea of the K-Nearest Neighbor (KNN) classification algorithm is that if most of the k most similar samples in a feature space (that is, the closest neighbors in the feature space) belong to a certain category, then the the sample also belongs to this category. The only parameter of this model is k. When k equals 3, the accuracy is 81.75% for training set and 66.93% for test set. The model performance on test set is not satisfactory and the overfitting is also very serious. I think k equals 3 is too small which could cause overfitting. I increase k and the accuracy is 75.08%/70.81%(k=10), 73.19%/71.77%(k=20) and 73.32%/72.07%(k=30). Before the slash is the accuracy of the training set, and after the slash is the accuracy of the test set. From this result, we can draw a conclusion that increasing k can reduce overfitting effectively and improve the model generalization performance while the processing time will increase a little.

B. Decision Tree

Decision Tree is a supervised-learning algorithm, the principle of this algorithm is to build binary tree to partition the data into the leaves of the tree. Each node of the tree is a chosen rule that maximise the "purity" of the leaf. There are two indexes to measure the purity, Gini index and Cross-entropy. In this project, we import the built-in Decision tree model from Scikit-learn library' and then tune the parameters of this model to make better prediction for our data set. There are two main parameters: criterion and max_depth. Criterion decide whether Gini index or Cross-entropy is used and Max_depth decide the max depth of the tree. To set a proper depth for the tree is the core task of this model since too shallow will reduce the accuracy and too high will also cause problems such as overfitting.

After many attempts, we conclude that criterion 'Gini index' and 'Cross-entropy' are similar in performance while 'Gini index' do better in lower depth and 'Cross-entropy' perform well in higher depth. We can see the accuracy of different tree depth in Fig.3. These four figures show the accuracy of four max depth: 2, 5, 16 and 20. The results show that the accuracy increase with the tree depth at first but turn to decrease when the max depth is too big and the best tree depth is about 6 with accuracy around 73.5%. The accuracy is calculated by 10-folds cross validation.

```
[k=2]  DecisionTree_cross_validation_accuracy: 70.15%
       Accuracy of test set: 70.52%
[k=5]  DecisionTree_cross_validation_accuracy: 73.20%
       Accuracy of test set: 73.42%
[k=16] DecisionTree_cross_validation_accuracy: 67.28%
       Accuracy of test set: 66.89%
[k=20] DecisionTree_cross_validation_accuracy: 65.92%
       Accuracy of test set: 64.95%
```

Fig. 3. The accuracy of different tree depth

C. Random Forest

Random Forest is ensemble algorithms, it utilizes the principle of decision tree but average the results from several different decision trees. The advantage of Random Forest is: It can deal with high dimensional dataset and always performs better than single decision tree. It is hard to overfit either. However, the disadvantage of Random Forest is it requires more computing resources and has poor performance on low dimensional dataset. The parameters of Random Forest are similar to decision tree and the most importance parameters are n_estimators and max_features. n_estimators represent the number of trees and max_features is the maximum number of features that allowed to use in the tree.

```
n_estimators = 10  max_features = 4
RandomForest_cross_validation_accuracy: 72.35%
Accuracy of test set: 72.55%
n_estimators = 10  max_features = 8
RandomForest_cross_validation_accuracy: 72.78%
Accuracy of test set: 72.95%
n_estimators = 50  max_features = 4
RandomForest_cross_validation_accuracy: 72.82%
Accuracy of test set: 72.89%
n_estimators = 50  max_features = 8
RandomForest_cross_validation_accuracy: 73.48%
Accuracy of test set: 73.69%
```

Fig. 4. The accuracy of different combination of n_estimators and max_features

In theory, the accuracy will increase with more tree and larger maximum number of features, that is the same with the results of our attempts shown in these figures.

Random Forest is the ensemble method of Decision tree and the performance of Random Forest model is usually better than that of Decision tree model.

D. Naive Bayes

At first, we just applied a Gaussian Naive Bayes classifier without optimization, the accuracies got from training set and test set are both 0.66. Later, we quickly found that not all features are suitable for the Gaussian Naive Bayes classifiers. Because not all the features are numerical or continuous.

Specifically, all features in this dataset can be divided into three types: numerical (or continuous), binary, and categorical. In the level of fields:

age in days /numerical

BMI (from height and weight) /numerical

MAP (from ap_hi and ap_lo) /numerical

cholesterol 1: normal, 2: above normal, 3: well above normal /categorical

gluc 1: normal, 2: above normal, 3: well above normal /categorical

gender 1 - women, 2 - men /binary

smoke whether patient smokes or not /binary

alco Binary feature /binary

active Binary feature /binary

Given these 3 types of features, we can make different assumptions regarding the distribution of the likelihood, i.e., Gaussian, Bernoulli, and Categorical distribution. And we fit 3 Naïve Bayesian classifiers to the training datasets sliced by the corresponding fields as sub-classifiers:

```
import sklearn.naive_bayes as nb
# fit 3 kinds of NB classifiers for 3 groups of features
clf1 = nb.GaussianNB().fit(cardio[['age','BMI','MAP']],cardio_targets)
clf2 = nb.BernoulliNB().fit(cardio[['gender','smoke','alco','active']],cardio_targets)
clf3 = nb.CategoricalNB().fit(cardio[['cholesterol','gluc']],cardio_targets)
```

Fig. 5. Sub-classifiers

In order to assemble these three sub-classifiers into the final classifier, we calculated the probabilities estimate for the sliced training vectors. Then the probabilities were stacked horizontally for the fitting of a new Gaussian Naïve Bayesian classifier, since the stacked probabilities are all numerical fields. And we got the final Naïve Bayesian classifier after fitting it on the stacked probabilities and the original target field.

```
#return probability estimates for the test vector X with diffrent feature groups
clf1_prob=clf1.predict_proba(cardio[['age','BMI','MAP']])
clf2_prob=clf2.predict_proba(cardio[['gender','smoke','alco','active']])
clf3_prob=clf3.predict_proba(cardio[['cholesterol','gluc']])
#stack the probs horizontally
clf_prob=np.hstack((clf1_prob, clf2_prob, clf3_prob))
#fit a new NB classifier based on that
clf= nb.GaussianNB().fit(clf_prob,cardio_targets)
```

Fig. 6. Final classifier

We calculated the accuracies for the training and test set. They are both improved to 0.68. It is worth mentioning that the accuracy of the test set is also calculated by stacking the probabilities returned by the three sub-classifiers:

```
#the accuracy on the test data
print(clf.score(np.hstack((clf1.predict_proba(cardio_test[['age','BMI','MAP']]),
                           clf2.predict_proba(cardio_test[['gender','smoke','alco','active']]),
                           clf3.predict_proba(cardio_test[['cholesterol','gluc']])))),cardio_test_targets))
```

Fig. 7. Test accuracy

Naive Bayes learners and classifiers can be relative fast compared to more sophisticated methods. On the other hand, naïve Bayes is known to be a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously. Additionally, there is no 'zero probability problem' in this binary classed classification problem because both classes appeared in the training data.

E. SVM

In SVM, we mainly adjust the penalty parameter C and Gamma. C tells the algorithm how much I care about misclassified points while Gamma defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. I used linear-kernel SVM to run the model with default parameters at first, the accuracy of the training set and testing set are 50.1% and 49.8%. Then I tried to finetune the parameters in this model, but the accuracy did change to much. I also implemented the RBF-kernel SVM, but the interesting thing is, the accuracy is the same as that of linear-kernel SVM, I tried many times and I am not sure if there is something wrong in my code. After I finetune the RBF-kernel SVM I can finally get a better performance, with 50.4% and 50.2% accuracy in training set and testing set when I set Gamma=5, C=2. In order to get an even better performance, I used Grid Search algorithm to try to find best parameters. The main idea behind it is to create a grid of hyper-parameters and just try all of their combinations. I wrote the code shown as follow.

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
model = SVC(kernel='rbf', probability=True)
param_grid = {'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000], 'gamma': [0.001, 0.0001]}
grid_search = GridSearchCV(model, param_grid, n_jobs = 8, verbose=1)
grid_search.fit(cardio, cardio_targets)
best_parameters = grid_search.best_estimator_.get_params()
model = SVC(kernel='rbf', C=best_parameters['C'], gamma=best_parameters['gamma'], probability=True)
model.fit(cardio, cardio_targets)
score9 = model.score(cardio,cardio_targets)
score10 = model.score(cardio_test,cardio_test_targets)
print(score9)
print(score10)
```

Fig. 8. Code of SVM

Although we ran the code for an hour, it still couldn't find the best parameters to improve the performance of RBF-kernel SVM. We change the parameter of Grid Search couples of times, I still fail to get the results.

REFERENCES

[1] https://www.kaggle.com/sulianova/cardiovascular-disease-dataset