London School of Economics and Political Science

ST451

Stochastic Volatility

And

Pima Indians Diabetes Database

Candidate Number:

14974

March 4, 2019

1 Stochastic Volatility (SV)

1.1 Introduction

Analysis of stochastic volatility model is very important in the financial market. For example, estimation of volatility is helpful to predict option-pricing, and some "event" style studies want to explore relationships between some specific event and changes in stochastic volatility[4]. However, the distribution of high-frequency stochastic process data are highly abnormal, capturing its change is not easy. In this report, I applied Hoffman model from [2], and used Markov chain Monte Carlo (MCMC) method to draw the conclusion from the model (i.e. unknown parameters). Because of highly abnormal distribution, using Metropolis-Hastins method to do sampling will result in high autocorrelation results [3], so I used a more advanced method No-U-Turn Sampler (NUTS) instead, which is the default method in the package PyMC3, and then the empirical distribution of those samples was used to approximate the target distribution.

1.2 Data

The Standard & Poor's 500 indexes (S&P 500) is an American stock market index based on the market capitalizations of 500 large companies having common stock listed on the NYSE, NASDAQ, or the Cboe BZX Exchange [9]. I chose a period of time from it, which was from 22 March 2018 to 22 March 2019, and there are 252 daily returns in total. The source of dataset is from the Yahoo Fiance [10]. The data was downloaded through Yahoo's Application programming interface (API), and the original data was transformed to the daily percent change to see daily volatility, as:

$$y_i = \frac{r_i - r_{i-1}}{r_{i-1}},$$

where r_i is daily return of SP 500, and y_i denotes daily percent change.

Then trace-plot of y_i is shown as figure 1, intuitively, the most violent fluctuations were around January, 2019.

1.3 Method

1.3.1 Model

According to Hoffman[2], the report assumed observed values S&P 500 index is generated by the following generative process:

$$\begin{aligned} \sigma^2, \nu, s_1 &\sim \mathit{Exponential}(100), \\ \log(s_{i>1}) &\sim \mathcal{N}(\log(s_{i-1}), \sigma^{-2}), \\ \log(y_i) &\sim t(\nu, 0, \exp(-2s_i)); \end{aligned} \tag{1}$$

where y_i denotes the observed daily percent change, and s is the 'Hidden Variables", denoting underlying values of daily volatility, s_i is drawn from the Gaussian Random Walk process, the value of s_i is only determined by previous step s_{i-1}, and s_i will determine the value of y_i; σ is a precision parameter of the normal distribution of $\log s_i$, and ν is a parameter relevant

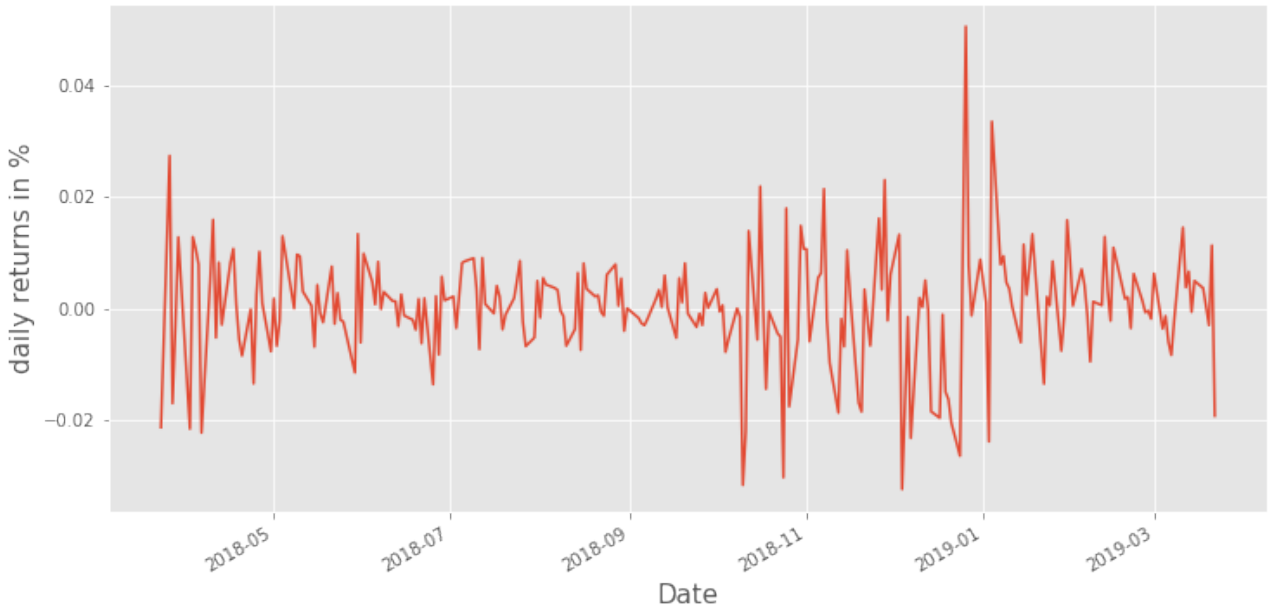


Figure 1: The trace-plot of daily percent change of S&P500 from March 2018 to March 2019.

to Student-T distribution of daily percent change y; s_1, ν and σ are drawn prior distribution $Exponential(100)$. According to Bayes theorem, the joint posteriors distribution is:

$$\begin{aligned} p(s, \nu|y) \propto & \exp(-0.01\nu)\exp(-0.01 s_1) \\ & (\prod_{i=1}^{252} t_\nu(s_i^{-1}(\log y_i - \log y_{i-1}))) \\ & (0.01 + 0.5\sum_{i=2}^{251}(\log s_i - \log s_{i-1})^2)^{-126} \end{aligned} .$$

Initialize, drew $sigma, \nu$ and $_1$ from $Exponential(100)$, then further s_i was drew daily from Gaussian Random Walk, and $\log y_i$ from Student-T distribution. Then, sufficient times was run (i.e. 100000) to make sure that Markov chain Monte Carlo(MCMC) process was converged.

1.3.2 Algorithm

According to lecture notes, we learned the Metropolis-Hasting algorithm and Gibbs Sampler, and apparently, Gibbs sampling is inappropriate here. In Python, package "PyMC3" introduces a more efficient method, named No-U-Turn Sampler (NUTS), which is able to converge much quicker from high-dimensional target distributions comparing to Metropolis Hasting and Gibbs[8]. In our project, we have 252 discrete hidden state s, and NUTS was shown a greater advantage comparing to Metropolis Hasting. During the sampling process, there are 4 independent chains running simultaneously, which are expected to have similar results if the chains are converged.

1.4 Result

1.4.1 Convergence and Mixing

As for MCMC, the most important part of diagnostics is to check whether the chains were converged, so I started with trace plots firstly. After 100000 times sampling, as figure 2 is shown, 4 parallel chains (i.e. in 4 different colors) nearly overlapped, therefore, results in multiple sampling are nearly the same. Also, Gelman-Rubin diagnostic $\hat{R}$ (i.e. it tests for lack of convergence[1]) of σ and ν was nearly 1, indicating this 4 chains were converged. Also, posterior distributions of ν and σ were very different from their prior distribution, and both density plots showed a clear single uni-modal, implying the MCMC process was effectively learned information from data. In addition, the trace plots also showed MCMC samplers mix well. The summary of parameters σ and ν were shown in table 1, effective sample sizes were 9936 and 219006 respectively for σ and ν. Although effective sample sizes were relatively small, comparing to 100000 times sampling, but there were sufficient quantities to make reliable estimation of parameters. Finally, $\hat{R}$ are nearly 1, implying MCMC chains might have converged.

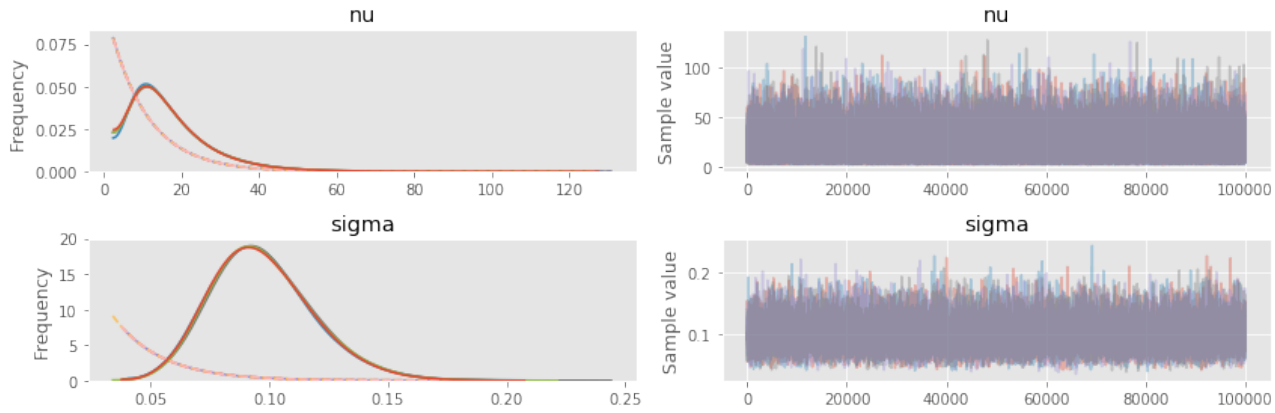


Figure 2: The left two plots are density plots, the solid lines are posterior distribution and dashed lines are prior distribution; the right two plots are trace plots, because of multiprocess sampling (i.e. 4 chains), there are 4 colors lines here.

Table 1: Table of summary statistics of σ and ν, s.d. is standard deviation, mc_error is Monte Carlo Error, hpd_2.5 and hpd_97.5 are 2.5 % and 97.5% Highest Posterior Density interval, n_eff is number of effective sample size, Rhat is the result of the Gelman-Rubin diagnostic tests.

	Mean	s.d.	mc_error	hpd_2.5	hpd_97.5	n_eff	Rhat
σ	0.0968	0.0214	0.0002	0.0574	0.14	9936.0247	1.0002
ν	17.2952	10.4317	0.0237	3.7662	38.1069	219005.935	1.0

1.4.2 Autocorrelation function (ACF) plots

The ACF plot is used to inspect the serial correlation of draws, the figure 3 showed that autocorrelation was already very small at the beginning, and it soon decreased to 0, and the figure 4 shows autocorrelation was large at short lags, but went to 0 in the first 100 lags.

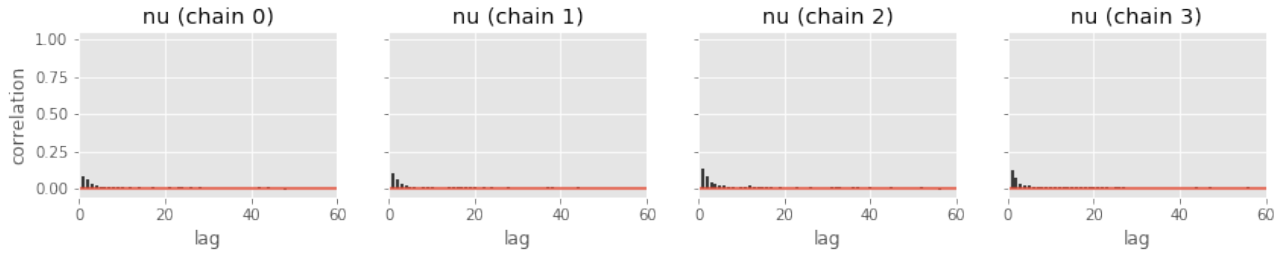


Figure 3: The autocorrelation plot of ν in 4 chains.

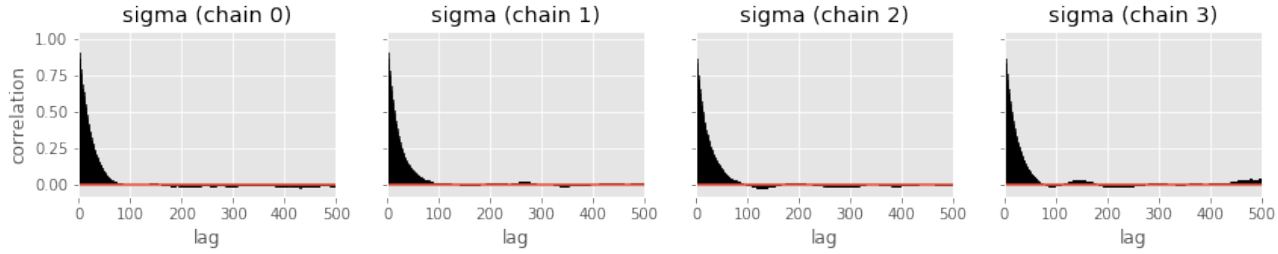


Figure 4: The autocorrelation plot of σ in 4 chains.

The figure 5 showed highest posterior density distribution of ν was right skewed and highest posterior density distribution of σ was symmetric. The figure 6 showed 10000 times sampling of volatility s, most samplers were captured by the 95% credible interval. There were two strong volatility, the first was due to my prior exponential distribution was far the posterior, and the second strong volatility was around January 2019, caused by "the Worst Christmas Eve" (i.e. a sharp decline of the American stock market in December 2018).

1.5 Conclusion

Overall, the performance of MCMC is quite well, according to 7, overlapped area of daily returns drawn from posterior roughly included the observed values, and there is no obvious problem to suspect the convergence of chains and the posterior distributions reasonable too. In section 1.3.1, the model was constructed, and substituting into point estimation of parameter (i.e. mean) in table 1, as:

$$\log(s_{i>1}) \sim \mathcal{N}(\log(s_{i-1}), 0.0968^{-2}),$$
$$\log(y_i) \sim t(17.2952, 0, \exp(-2s_i));$$

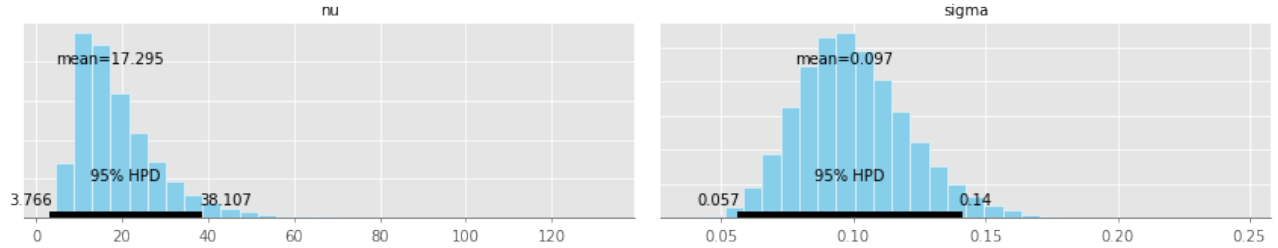


Figure 5: The posterior plots of ν and σ.

Based on this, further stochastic volatility could be sampled when we keep updating the random walk process s_i. In section 1.4.1, I used various methods to test convergence diagnostics and model validation, all results indicating that the chains were highly likely to be converged. One strange phenomenon is a relatively small number of effective sample size, this may be caused by the strong autocorrelation among daily returns, which may influence the efficiency of the algorithm. Also, the model does not consider the potential possibility of some big events (e.g. financial crisis), so the model still need further perfect.

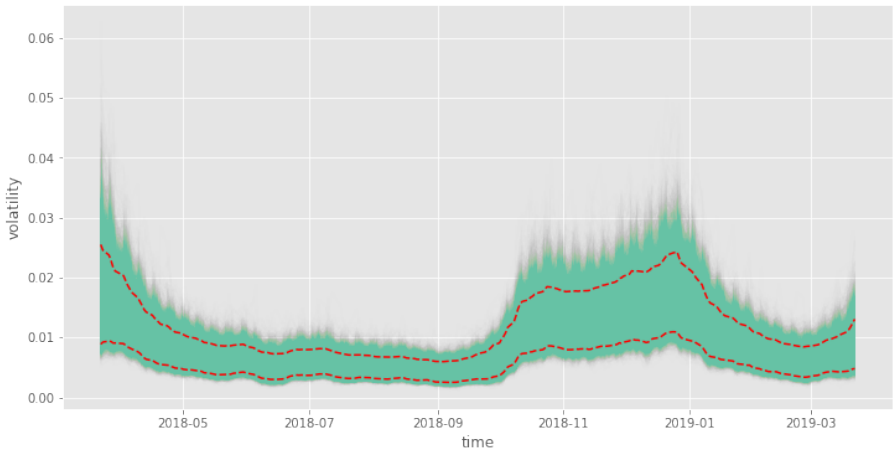


Figure 6: The red dashed line are 95 % credible intervals, and green areas are superposition of 100000 sampled traces, so the more overlapping part of the deeper the color.

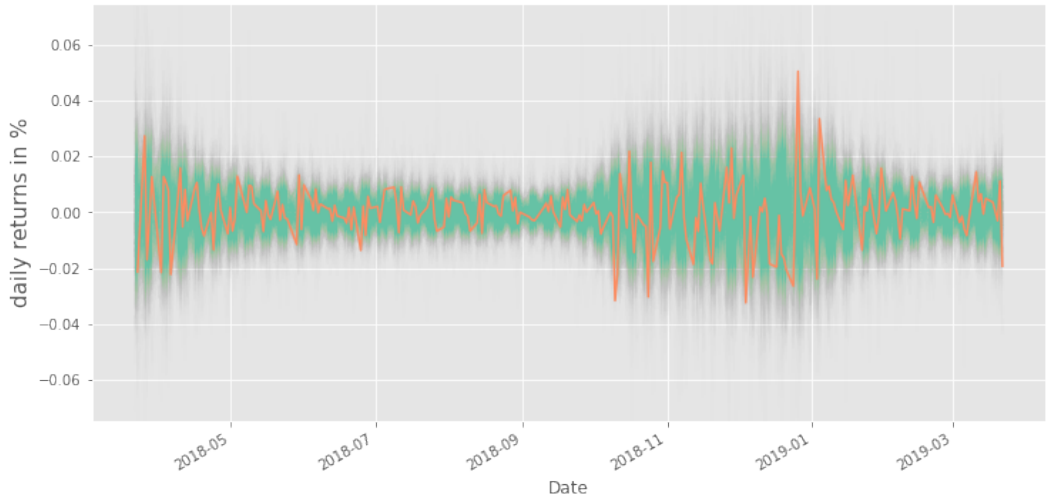


Figure 7: The orange line is observed daily percent change from 22 March 2018 to 22 March 2019, the green overlapped lines were 1000 independent realizations of daily return for this time period base on the posterior distribution.

2 Pima Indians Diabetes Database

2.1 Introduction

Forecasting and discriminating disease is a topic of increasing concern, many diseases may exist some hidden functional relationships with features of patients (e.g. blood type, etc.), so it is quite interesting to apply statistical techniques to explore the potential relationship. In the second part of the report, I used Naive Bayes and K-Nearest Neighbors classification(KNN) to do classification for the Pima Indians Diabetes Database. I start with variable selection, then standardization, and finally model fitting in Naive Bayes and KNN respectively.

2.2 Data

The source of dataset is from the National Institute of Diabetes and Digestive and Kidney Diseases, the target population in the dataset was the Pima Indian population near Phoenix, Arizona [5]. There are 768 patients in the dataset, including 8 diagnostic measurements and 1 target variable (i.e. whether the patient has diabetes), see full description in table**??**. All patients in this dataset are female and over 21 year old. The aim of dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. In order to test the performance of classification, the whole dataset is divided into training (67%) and test sets (33%) by using a random holdout split, so there are 514 records in training part and 254 in test part.

Table 2: Full description of variables in dataset.

Name	Description
Pregnancies	number of times pregnant
Glucose Plasma	glucose concentration a 2 hours in an oral glucose tolerance test
Blood Pressure	diastolic blood pressure (mm Hg)
Skin Thickness Triceps	skin fold thickness (mm)
Insulin	2-Hour serum insulin (mu U/ml)
BMI	body mass index (weight in kg/(height in m)^2)
Diabetes Pedigree Function	diabetes pedigree function
Age	Age (years)
Outcome	Class variable (0 or 1) 268 of 768 are 1, the others are 0

2.3 Method

2.3.1 Naive Bayes

Firstly, I use naive Bayes to do classification, because of the property of naive Bayes, I assume all features are independent of each other, for example, under our assumption, patients' BMI should be irrelevant to age, though they may have a potential connection, independent assumption is made here.

The equation of Naive Bayes could be written as:

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times ... \times P(x_8|c) \times P(c); \quad (2)$$

x_i denotes variables (i.e. pregnancies, glucose plasma and etc.), c denotes outcome (i.e. 0 or 1); $P(x_i|c)$ is the probability of variables x_i when class c happens, $P(c)$ is the prior probability of class c, $P(c|X)$ is the posterior probability of class given variable. This dataset has two classes, 0 or 1, so I could calculate posterior probability $P(c = 0|X)$ and $P(c = 1|X)$, if $P(c = 0|X) > P(c = 1|X)$, the patient is classified as "0", otherwise is classified as "1".

2.3.2 K-Nearest Neighbors classification(KNN)

In order to do the comparison, I also use non-Bayes method KNN to do classification. KNN assumed that distances among individuals in the same class are sufficiently smaller than different classes, in other words, similar things are near to each other. Therefore, for any individual in dataset, I find the K nearest individuals around the current individuals, and mode of those individual's' classification is predicted classification of the current individual. The choice of values of K is decided by the cross-validation, which means KNN models with different K are fitted, and chooses the K with minimal error. In addition, variables are rescaled by $z = \frac{x_i - \mu}{\sigma}$ so as to fix the skewness of original data, where μ is mean and σ is variance.

2.3.3 Evaluation

In order to evaluate the performance of two methods, this report uses accuracy, confusion matrix and Receiver Operating Characteristic (ROC) curve to evaluate the performance of the model. Confusion matrix is always used in for summarizing the performance of a classification algorithm, it including precision, recall and F1 score, the closer to 1 the better the performance. The dataset is imbalance(i.e. number of patients with diabetes are higher then number of patients without.), considering many imbalance situations have high accuracy with low precision or recall, so it is quiet necessary to calculate the confusion matrix. ROC curve is also a very important criterion, as it may balance trade-off of Type I error and Type II error, and larger AUC (Area under the curve) is, the better the performance.

2.4 Result

2.4.1 Variable Selection

Some variables with strong multicollinearity could be eliminated, so I used correlation matrix to check any potential correlations, as the correlation matrix 8 was shown, there was no strong correlation among variables, as Pearson's Correlation Coefficients were all lower than 0.2, so there was no need to eliminate any variables.

2.4.2 Choice of Parameters

For Naive Bayes, I don't need to choose the value of the parameter, but the value of K in KNN is necessary, and cross-validation is appropriate method here, so I fitted the KNN models with different K, and selected K with maximum accuracy. According to cross-validation figure 9, for value of K from 1 to 14, when $K = 11$, the accuracy maintained the maximal, which is 74.61%.

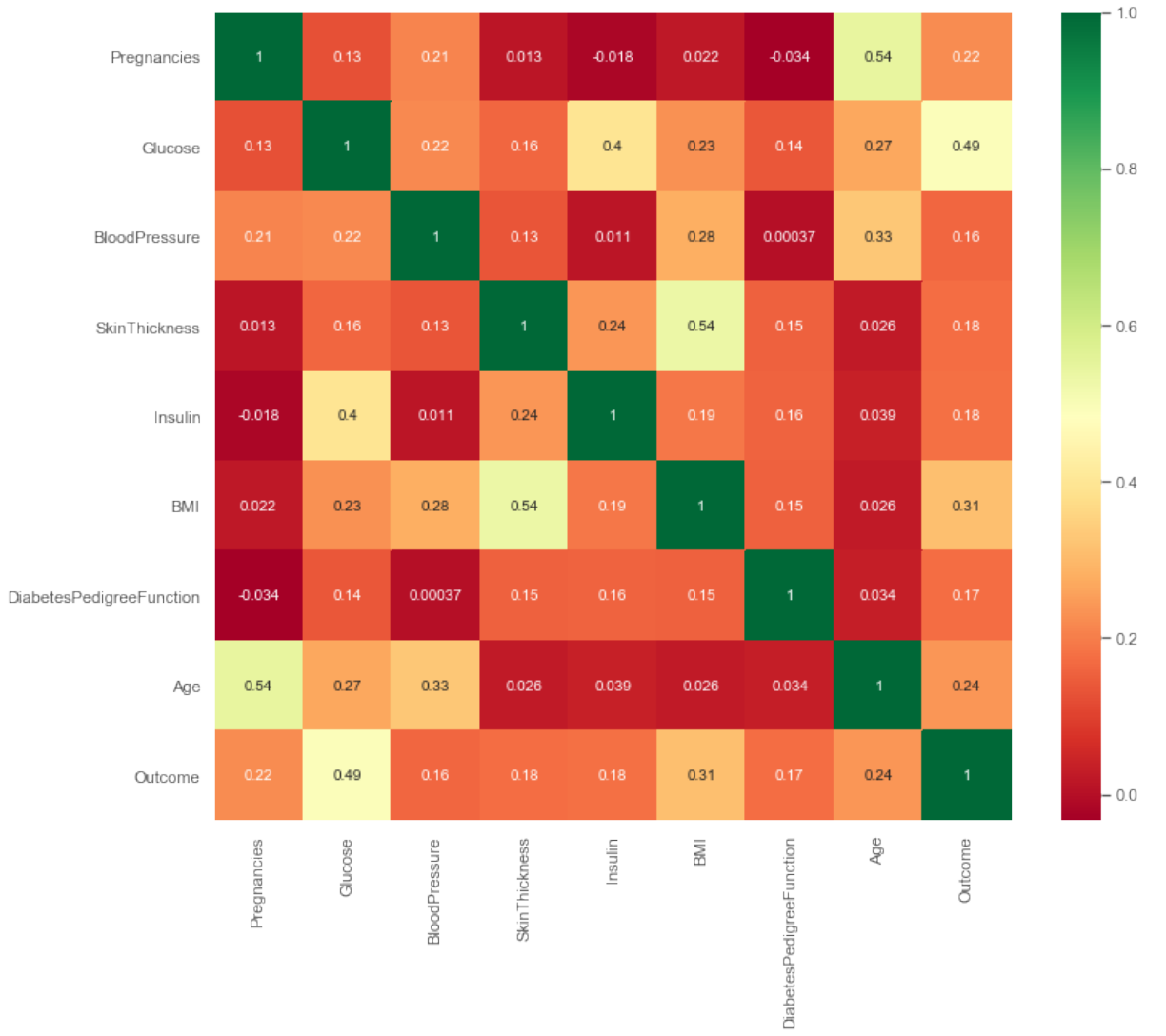


Figure 8: The correlation matrix plot of Pearson's Correlation Coefficients among variables.

2.4.3 Model Performance Analysis

The accuracy of Naive Bayes was 72.66% and the accuracy of KNN was 74.61%, the performance of KNN was slightly better than Naive Bayes, but considering our data was highly unbalanced, confusion matrix was very necessary here, and confusion matrices of two methods were shown in table 3, all results seem reasonable, indicating Naive Bayes and KNN are proper methods here, too. Similarly, the confusion matrix figures were shown in 10(a) and 10(b), most non-diabetes patients were classified as "0" and most diabetes patients were classified as "1", so performance of classification was good.

As figure 10(c) and 10(d) were shown, the area under curve (AUC) for Naive Bayes and KNN are 0.7 and 0.8 respectively, indicating both models are fair, though KNN was slightly better.

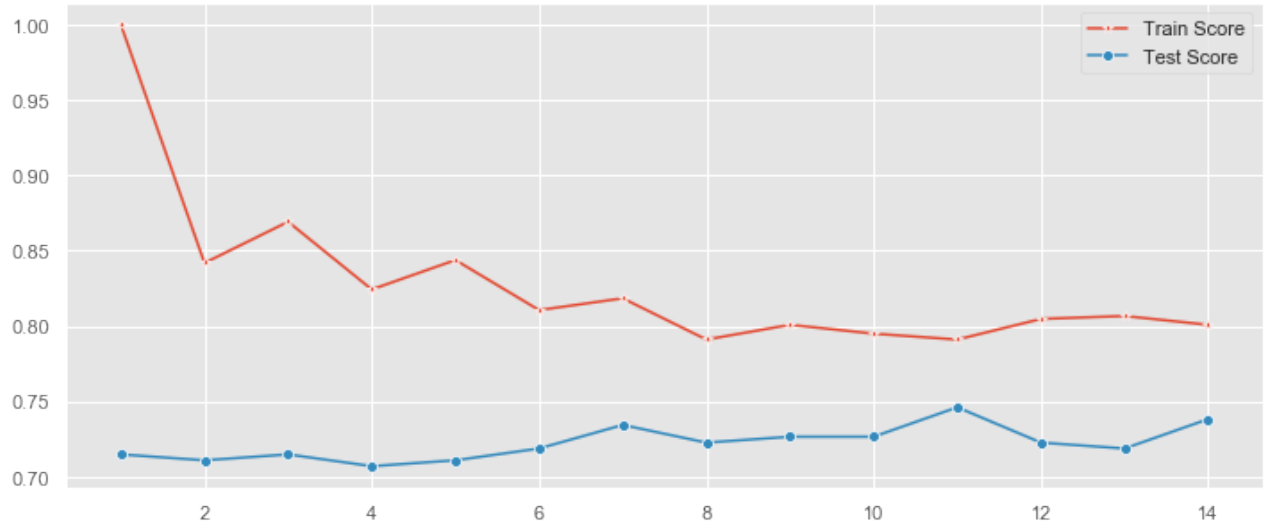


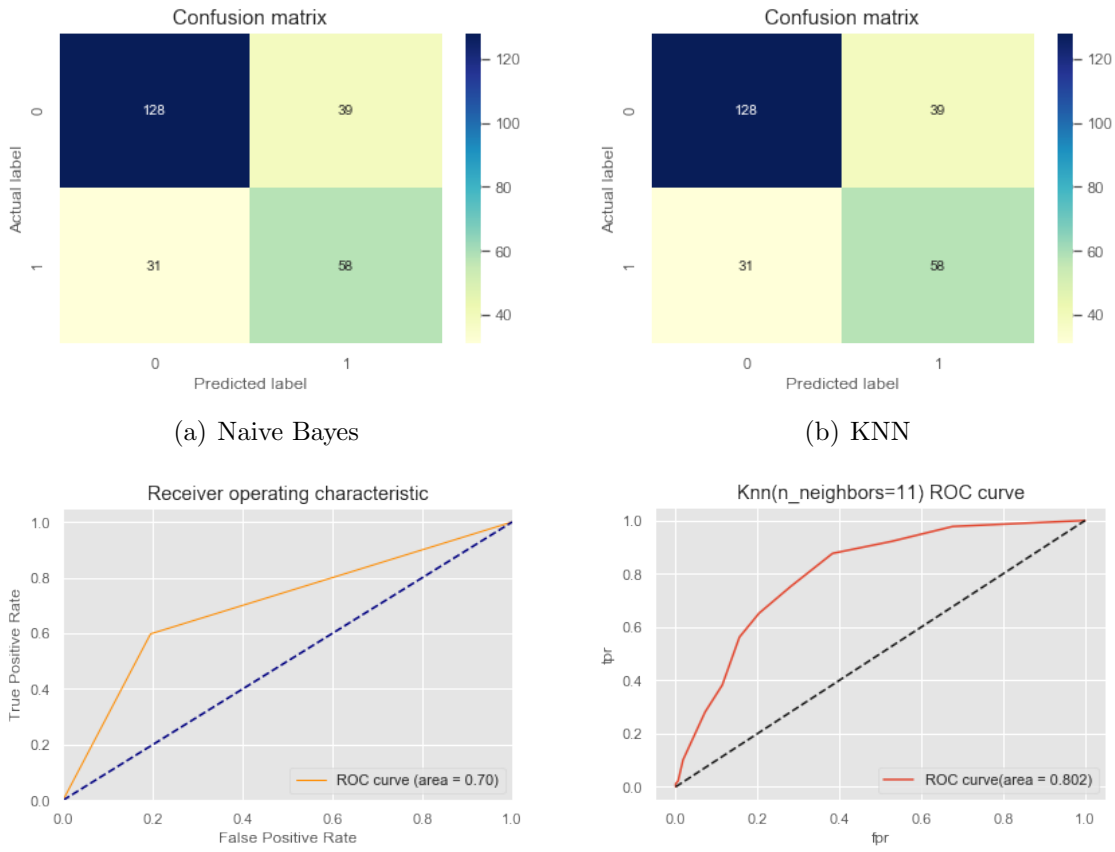
Figure 9: The cross validation plot for KNN classification.

Table 3: Precision, recall and F1 score for Naive Bayes and KNN.

Class(Naive Bayes)	Precision	Recall	F1 score	Support
Non-diabetes("0")	0.81	0.77	0.79	167
Diabetes("1")	0.60	0.65	0.62	89
Class(KNN)	Precision	Recall	F1 score	Support
Non-diabetes("0")	0.78	0.84	0.81	167
Diabetes("1")	0.66	0.56	0.61	89

2.5 Conclusion

In conclusion, I find both models have quite good performance (e.g. accuracy are higher than 70%), though KNN performed slightly better than Bayes. Naive Bayes is easy to interpret and only requires a small number of training data to estimate the parameters necessary for classification[7], and KNN is more accurate but more calculation complexity. One main limitation of Naive Bayes is the assumption of independent predictors, this may explain why Naive Bayes performs slightly worse. In real life, it is almost impossible to get a set of predictors which are completely independent, and those variables in this dataset cannot be regarded as completely independent, for instance, the elderly are more likely have high blood pressure. Therefore, some modified method, like Semi-naive Bayesian classifier [6], was introduced, which could slightly relax restrictions of independent assumption.



(a) Naive Bayes

(b) KNN

(c) Naive Bayes

(d) KNN

Figure 10: Confusion matrix and ROC plots for for Naive Bayes and KNN.

References

[1] Steve Brooks et al. *Handbook of markov chain monte carlo*. CRC press, 2011.

[2] Matthew D Hoffman and Andrew Gelman. "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1593–1623.

[3] *Introduction of PyMC3*. `https://https://blog.csdn.net/jackxu8/article/details/71080865`. Accessed: 2017-05-02.

[4] Eric Jacquier, Nicholas G Polson, and Peter E Rossi. "Bayesian analysis of stochastic volatility models". In: *Journal of Business & Economic Statistics* 20.1 (2002), pp. 69–87.

[5] R S Johannes. "Using the ADAP learning algorithm to forecast the onset of diabetes mellitus". In: *Johns Hopkins APL Technical Digest* 10 (1988), pp. 262–266.

[6] Igor Kononenko. "Semi-naive Bayesian classifier". In: *European Working Session on Learning*. Springer. 1991, pp. 206–219.

[7] *Naive Bayes classifier*. `https://en.wikipedia.org/wiki/Naive_Bayes_classifier#Semi-supervised_parameter_estimation`. Accessed: 2019-04-30.

[8] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. "Probabilistic programming in Python using PyMC3". In: *PeerJ Computer Science* 2 (2016), e55.

[9] *SP 500 Index*. `https://en.wikipedia.org/wiki/S%26P_500_Index`. Accessed: 2019-04-15.

[10] *SPDR SP 500 ETF (SPY)*. `https://finance.yahoo.com/quote/SPY`. Accessed: 2019.