University College London

Masters Thesis

Gaussian Process Prediction of Fine Wine Prices

Author:
Michelle Yeo

Supervisors:
Dr. Tristan Fletcher,
Prof. John Shawe-Taylor

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in

Computational Statistics and Machine Learning
Department of Computer Science

August 2014

This report is submitted as part requirement for the MSc Degree in Machine Learning/C-SML at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

"I applied my Gaussian processes algorithm to this nasty regression problem. It sucked its thumb, cried, then spat out a line of best fit."

@$ML\_Hipster$ (2013)

UNIVERSITY COLLEGE LONDON

Abstract

Faculty of Engineering
Department of Computer Science

Master of Science

Gaussian Process Prediction of Fine Wine Prices

by Michelle Yeo

The present thesis aims to introduce advanced machine learning techniques like Gaussian process regression and multi-task learning to predict the returns of fine wine, using only previous wine return values as input. These methods are novel in the area of wine price prediction as previous research in this area have only utilised linear regression to predict wine prices. Working with the 100 wines in the Liv-Ex 100 index, the main contributions of this paper to the field are, firstly, a clustering of the wines into two distinct clusters based on autocorrelation features. Secondly, an implementation of Gaussian process regression on these wines with MSE results surpassing both the trivial prediction and simple ARMA and GARCH time series prediction benchmarks. Lastly, we present and implement an algorithm which performs multi-task feature learning with kernels on the wine returns as an extension to our optimal Gaussian process regression model. Using the optimal covariance kernel from Gaussian process regression, we achieve MSE results which are comparable to that of Gaussian process regression. Altogether, our research suggests that there is potential in using these advanced machine learning techniques on fine wine price prediction.

Acknowledgements

First, and certainly foremost, I would like to thank my external supervisor Dr. Tristan Fletcher for introducing me to this fascinating topic as well as providing me with constant mentorship throughout the thesis writing process. His careful direction in the analysis helped me develop a deeper intuition for the machine learning techniques in this paper and the way they interact with the data.

Secondly, I would like to thank my internal supervisor Prof. John Shawe-Taylor for very patiently guiding me with the more theoretical aspects of the project. His lucid explanations of the more challenging and abstract concepts are simply first class.

Thirdly, I would like to thank my family for constantly supporting me throughout the thesis writing process and patiently enduring my numerous midnight Skype calls home. Sorry!

Fourthly, I would like to thank Ethan Koh for enthusiastically pretending to understand my project (fair enough, since I pretend to understand what he does too), brightening my days with song and dance, and making sure I am healthy and well-fed.

In addition (because I was told its bad to begin a sentence with "fifthly"), I would like to thank Li Chen and LeeMey Goh for proofreading this paper and suffering through pages of careless grammar.

Also, I would like to thank the UCL computer science department and the UCL Gatsby Computational Neuroscience Unit for providing me with two excellent courses (Supervised Learning and Probabilistic and Unsupervised Learning) which provided me with the tools to understand and implement the methodology outlined in this paper, and, of course, subsidised software.

Lastly, I would like to thank Invinio Ltd for giving me access to their wine data without which this project would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The existing research on predicting wine prices is scarce and the few papers on this topic mainly employ the technique of multiple linear regression to predict wine prices [1] [2] [3] [4] [5]. This is not unreasonable, as using linear regression we get a simple but clear explanation of the forces that drive the prices of wine just by examining the optimal set of predictor variables (eg. weather, number of medals the vineyard won) which best explain the variance in wine prices as well as the goodness of the model fit. We note here, however, that the primary focus of these papers is on explanation rather than prediction. Thus, a hypothesised relationship between certain predictor variables (eg. weather) and wine prices is made, usually based on intuition or reasoning derived from market forces, and linear regression is subsequently carried out to test the strength of that relationship.

This paper will move away from the explanatory direction which dominates the existing literature, and instead will focus primarily on wine price prediction. We thus eschew searching for and hypothesising about the effects of certain predictor variables and wine prices, focusing instead on treating wine returns (i.e. price difference between time t and time t-1) as a stochastic process and employing machine learning techniques, specifically Gaussian processes, to model the returns. We start with fitting the basic ARMA and GARCH models used in most of the time series and financial literature to a smaller set of 100 wines. We then go on to fit Gaussian process regression models with different kernel functions. Following that, we look at making use of the relationships between the wines by incorporating common underlying features among the wines into the Gaussian process regression framework using multi-task feature learning. These machine learning techniques were chosen mainly due to their compatibility with the data and the natural and reasonable way they build upon one another which lends an overall coherence to the project.

Along the way, the main research question we seek to explore and ultimately hope to answer is the following: to what extent do more complicated machine learning techniques like Gaussian process regression and multi-task feature learning perform better than simpler ones like ARMA and GARCH models? A satisfactory answer to this problem would not only be beneficial to the area of wine price prediction in general, but also shed some light on the overall utility of using advanced machine learning techniques in realistic, practical forecasting situations, especially when taking into account the overall complexity (both computationally and theoretically) of such methods.

Interestingly, we note that such machine learning techniques (in fact, anything other than linear regression), although commonly used in the forecasting literature of other financial instruments [6] [7] [8] [9], are completely unprecedented when it comes to wine price forecasting. We also note that wine differs from the majority of other financial instruments in three main ways: 1) it does not have a yield, 2) the supply of wine from a given vintage year is always monotonically decreasing, 3) with most wines, the global supply is unknown and often has to be estimated. The rationale behind these differences is due to the fact that wine is mainly bought for consumption, a process which is both irreversible and generates no internal rate of return. Together, these differences are arguably sufficient to put wine into a separate class of financial instruments altogether and thus this research on using machine learning techniques in wine price prediction is, in a certain sense, quite novel.

We assume a basic familiarity with matrix algebra, statistical concepts and distributions, and the MATLAB programming language. The rest of the paper will be organised as follows: chapter 2 gives an overview of the wine data we will be working with so as to better understand the choice of prediction techniques we choose to focus on. Simple ARMA and GARCH model fitting and forecasting techniques will also be implemented on the data. Chapter 3 explores Gaussian process regression as a method of predicting wine prices. Following that, we look at leveraging common, latent features among the wines by using multi-task learning in chapter 4. At the end of each chapter, a comparison and discussion of the forecasting techniques will be carried out. Finally, an overall discussion of the significance of our research and a consideration of future work in this area will be covered in the concluding chapter.

Chapter 2

Exploratory data analysis

2.1 Origins of wine data

The wine database we are given contains 10458 wines from various regions with wine prices dating from 2007 all the way to the present day. Apart from wine prices, the database also includes other information about the wines like vintage, appellation, and colour. All of these information come from the wine-searcher website ($http://www.wine-searcher.com/$) through their API. Because of the sheer size of the database, however, we will focus the preliminary data analysis on smaller subset of 100 wines, namely the 100 wines in the Liv-Ex 100 wines index (April 2014 composition), which represents the price movement of 100 of the most sought-after fine wines for which there is a strong secondary market. The Liv-Ex 100 wines come from 5 regions (Bordeaux, Burgundy, Rhone, Champagne and Bolgheri) and 14 vintage years. Throughout this paper, we will only refer to the wines by their ID number and not their names since we feel that the length and similarity of some of the wine names would serve as nothing but an unnecessary distraction to the overall goal of this paper.

2.2 Data cleaning and processing

Data cleaning consists of first converting wine prices of these 100 wines into GBP and then standardising the frequency of the wine prices. Here, it is important to note that although we have price information for most of the Liv-Ex 100 wines dating all the way until 2007, the frequency of the prices are rather irregular. For instance, from January 2007 to August 2013, we only get wine prices on the first of the month with some months skipped over. From mid-August to the current date today, however, we start to get pretty frequent wine prices with gaps of 2-3 days on average separating these wine

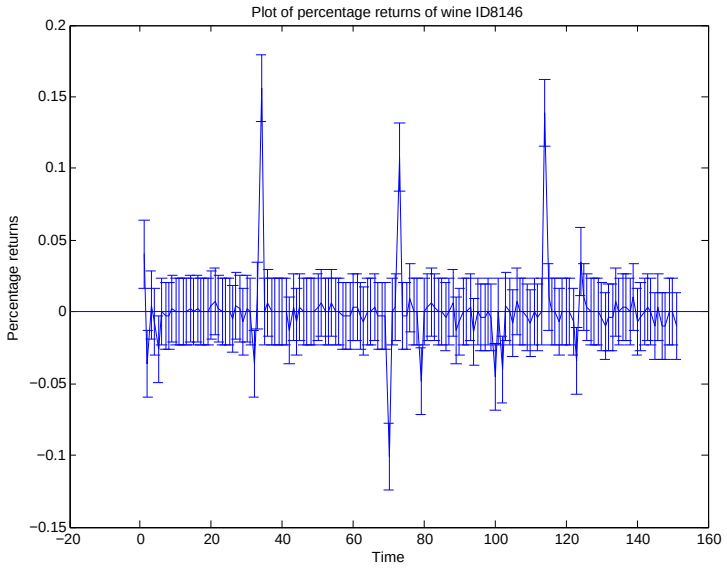
prices. As such, we decided to ignore the relative variability in the gap between wine prices from mid-August 2013 to the current date and just use all the wine prices from mid August 2013 to the current date so as to maximise the amount of price data we have on each wine. We note that an alternative treatment of the variability in the time interval separating the wine prices would be to average all the wine prices in a given week so we get weekly price data. Nevertheless, to average the price points over a week would give us a mere 29 weeks of price data to work with which could severely hamper trend detection, and thus we chose to go ahead with the former option instead. Finally, we chose to work with percentage returns ($:= \frac{returns_t}{returns_{t-1}} - 1$) instead of wine prices due to the fact that returns are generally more stationary.

Before going on to describe the nature of the wine percentage returns, we first introduce some notation which will be used throughout this chapter. For each wine, we denote the observed percentage returns at the most recent time point as Y_T, with the percentage returns at the previous time points as Y_{T-1}, Y_{T-2} and so on. We also assume that the returns are generated by a stochastic process defined as $Y_t + \epsilon_t$, ϵ_t being the value of a purely mean 0 random process at time t which is independent of Y_t so that at time t, $\mathbb{E}(Y_t + \epsilon_t) = Y_t$, for t = 1 ... T.

2.3 Nature of wine data

To get a better understanding of the nature of the wine returns, we first plot the percentage returns of each wine for each time point. For most of the wines, these values tend to be constantly close to the mean of the series across time, with occasional spikes and dips which deviate significantly from the mean. We note that is slight evidence of volatility clustering in the plots, that is, large fluctuations are usually succeeded by large fluctuations and vice versa. There is also no significant upward or downward trend for percentage returns of the wines. In addition, we note that the plot of percentage returns values across time for most of the wines show some evidence of autocorrelation, where values above the mean, especially values which are very much higher than the mean, tend to be followed by values above the mean and vice versa. Figure 2.1 below illustrates this phenomenon of autocorrelation with the horizontal line signifying the mean value of the series and the 1 standard deviation error bars around the percentage returns at each time point.

FIGURE 2.1: Returns with error bars of 1 standard deviation for wine ID 8146



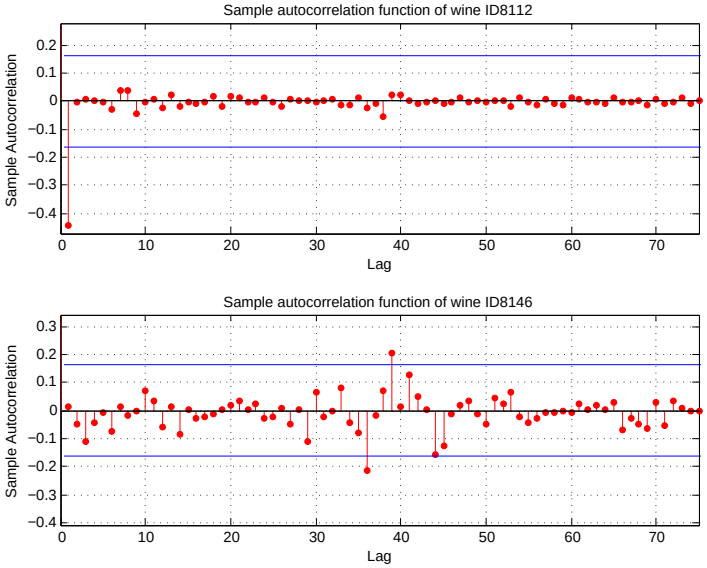
As we can see from this particular plot, the returns tend to stay close to the mean, which is around 0, and which also does not seem to change over time. Some values in the plot are clearly very much larger or smaller than the mean. These values do not seem that unreasonable, however, especially when we convert the returns back to prices, and since we notice that when looking at all the wines in general these values persistently occur at around the same period, we decide to just keep these values rather than drop them in the hope of perhaps finding more information eventually which can shed light on these extreme values.

To check for possible dependence between the percentage returns as well as the random process across all time points, that is, dependence between $Y_T, Y_{T-1}, ...$, and dependence between $\epsilon_T, \epsilon_{T-1}, ...$, we looked at the plot of the sample autocorrelation function (acf) of percentage returns, that is, $corr(Y_T, Y_{t-k})$, for all the wines against lags (k) for all the lags up to the end of the each time series. Here, we found something quite interesting: the plots of the acf of roughly half of the wines follow a clear exponential decay with signs of strong negative autocorrelation between the returns at the current time point and the previous time point. An example of which is seen in the first plot of figure 2.2 which shows the plot of the sample acf for wine ID 8112. We also plotted the confidence bounds of $\pm$ 2 standard errors of the sample autocorrelation coefficients ($\pm 2/\sqrt{n}$), which is not only commonly used in the time series literature but also, assuming independent percentage returns, would detect the 5% of autocorrelation coefficients which fall outside 2 standard errors of their sample mean, which we believe to be evidence of significant temporal relationship. In the first plot of figure 2.2, we see the acf value at lag 1 is clearly

significantly lower than the lower confidence bound which strongly suggests a significant lag 1 temporal relationship between either the returns, the random process, or both. We test this significance further by using the Ljung-Box Q test for autocorrelation at significance level 0.05 for wine ID 8112 which returns a p-value of 2.81×10^{-6} for the autocorrelation coefficient at lag 1. Similar Ljung-Box Q tests conducted for the other wines in this group also yielded similarly small p-values at lag 1 which reinforces our suspicion of a significant lag 1 temporal relationship for the wines in this group.

For the other half of the wines though, the acf plots do not give a clear indication of a strong temporal relationship whatsoever as seen in the second plot in figure 2.2 of the sample acf for wine ID 8146 where we have most acf values falling below the confidence bounds with random spikes protruding above the bounds at infrequent time intervals. Yet to dismiss this as evidence of a purely random noise process seems rather tenuous as the acf values for most wines in this group, though not high or low enough to exceed the bounds, still remain pretty close to and even exceeding the bounds at certain lags. Ljung-Box Q tests for the significance of the acf coefficients for wines in this group at the same significance level also displayed high p-values across the wines, again reinforcing insignificance of acf coefficients of the returns for the wines in this group.

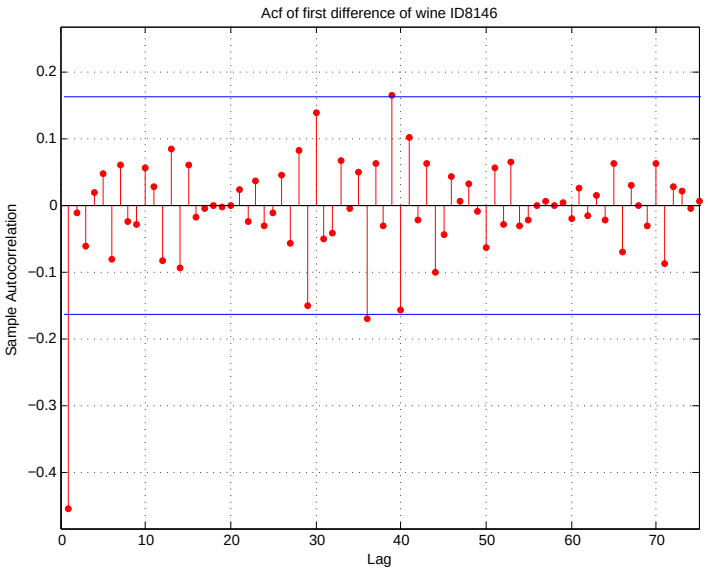
FIGURE 2.2: Comparing the acf of wines from the 2 groups



To check for evidence of a more complicated autocorrelation structure, we took the first difference of the percentage returns of the wines in the second group and plotted the acf values against lag like before. For most of these wines, the plot of the acf of the first difference shows a similar structure to that of the acf of the wines in the first group, namely, significant acf values at lag 1. A plot of the acf values for the transformed

returns of the same wine in the previous plot is shown in figure 2.3. The acf value at lag 1 clearly dips below the confidence bounds, although there are still random spikes near the confidence bound at infrequent time intervals. The Ljung-Box Q test for this wine gave us tested significant at lag 1 with a p-value of ≈ 0 and similar Ljung-Box Q tests conducted on the other wines in the group also gave significant results and small p-values at lag 1.

FIGURE 2.3: Acf of first difference of wine ID 8146



In addition, we looked at box plots of acf coefficients firstly over the wines in each group over all lags to determine the lag beyond which the acf coefficients tail off into noise. The reason for doing so is to see if, beyond using significance tests and looking at confidence bounds like above, we can get an more intuitive feel of when we pick up correlations and when we just pick up noise, by observing the lag beyond which the range and mean of acf values are consistently near 0. Moreover, using this methodology simultaneously with the above would also fortify our confidence our above observation of the two different types of autocorrelation structures.

Using this methodology, we get lag 1 as the optimal lag beyond which the acf coefficients tail off into noise for the wines in the first group. We see this in figure 2.4 where, at lag 1, the whiskers of the plots (chosen to cover 99.7% of the points, that is, 3 standard deviations, assuming the distribution of autocorrelation function values were normal) start from around -0.39 and end at around -0.5 and we get a mean acf value of -0.437. Beyond lag 1, the whisker range and mean acf values abruptly shifts to centre fairly consistently around 0. Altogether, this reinforces our hypothesis that we have a

significant lag 1 temporal relationship among the wine returns for the wines in the first group.

For the returns of the wines in the second group, this optimal point is not as clear as the that in the first as the distribution of the acf values of percentage returns across the lags display a persistently large whisker range which continue until around lag 115 beyond which we can safely say the range and mean of the acf values consistently cluster around 0 as seen in figure 2.5 below. To say that this is the optimal point beyond which the acf coefficients tail off into noise seems rather artificial, however, as unlike the box plots of the acf coefficients of returns for the wines in the first group, there is no sharp change in the mean or in the whisker range of the acf coefficients across all lags. Looking at box plots of the acf coefficients of the differenced returns across all lags as evident in figure 2.6, we get a better sign of where we get a significant relationship and where we just get noise. Thus from figure 2.6, the optimal lag point appears at around 1, beyond which the values seem to be fairly uniformly clustered around 0, though we still note a few lag points with more deviant acf values and ranges. Altogether though, it appears that the distribution of acf values of the differenced returns across lags seems to support our above observation that taking the first difference of the wines in the second group reveals an autocorrelation structure similar to the lag 1 relationship in the first group of wines.

FIGURE 2.4: Box plots of acf of returns for all lags for wines of first group

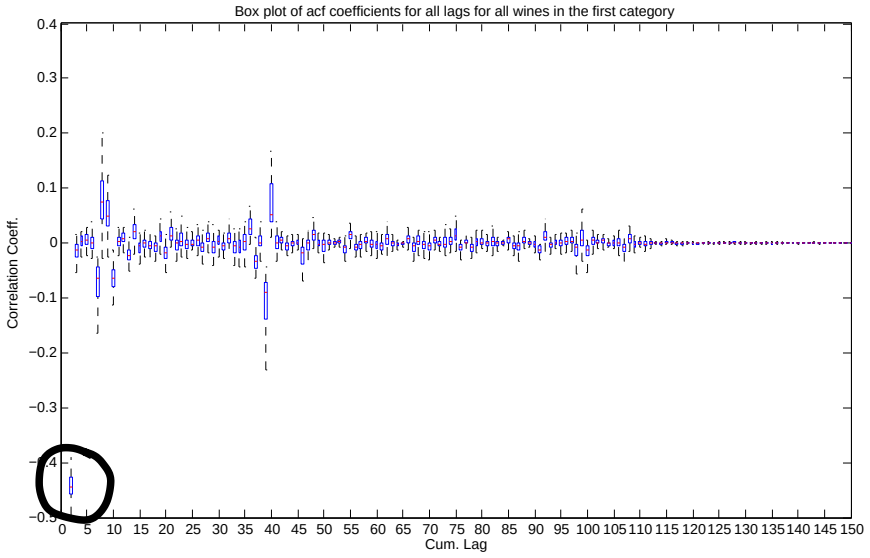


FIGURE 2.5: Box plots of acf of returns for all lags wines from second group

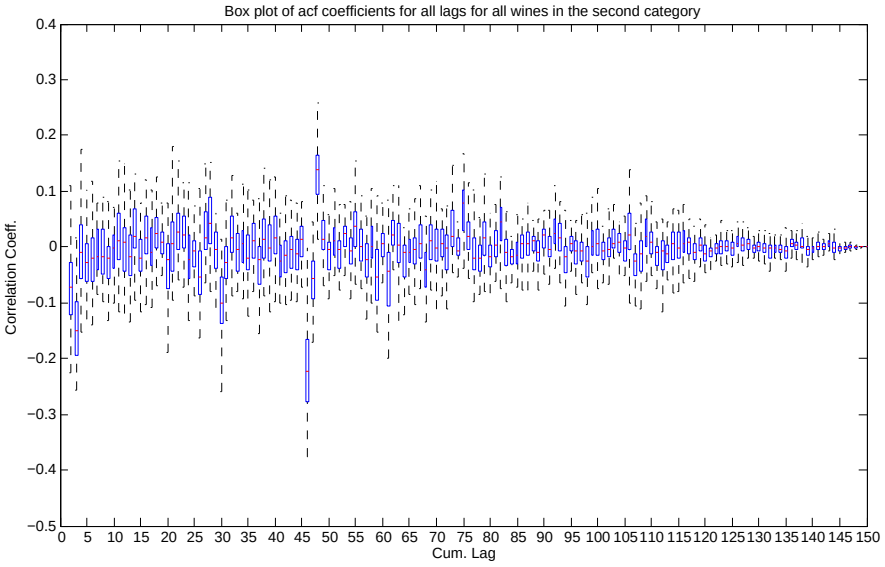
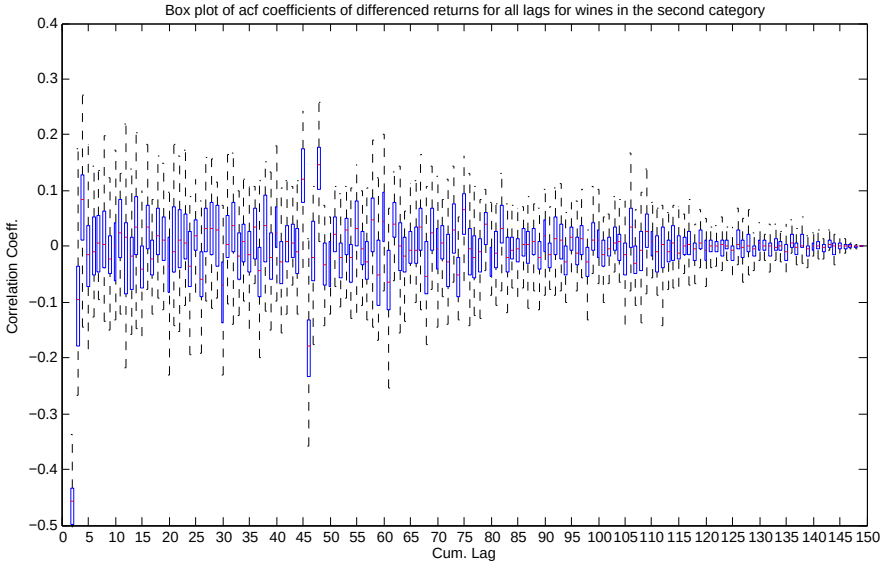


FIGURE 2.6: Box plots of acf of differenced returns for all lags for wines from second group

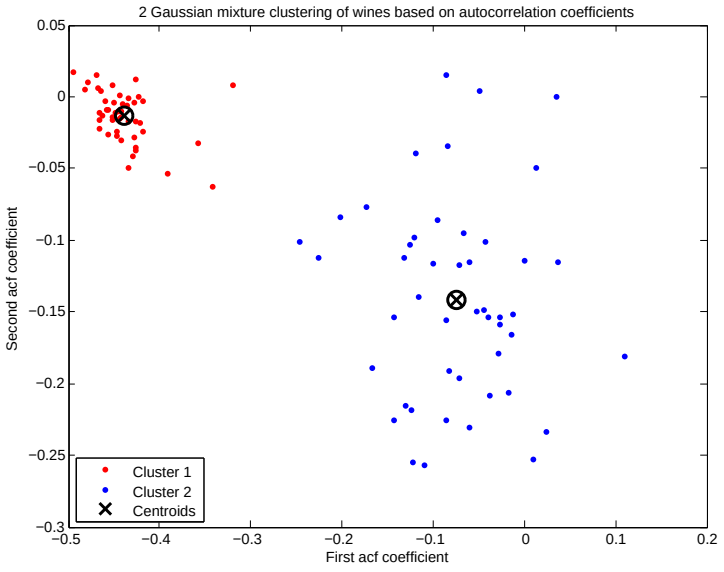


Altogether, the distribution of acf values across lags supports our above observation of the two different types of autocorrelation structures. This points to a natural classification of the wines into 2 categories: the former with a clear lag 1 temporal autocorrelation structure, and the latter with perhaps a more complicated autocorrelation structure. For the wines in the first category, a possible explanation for the autocorrelation structure of a strong negative lag 1 temporal relationship is perhaps simply regression of wine prices to the mean, that is, large positive percentage returns tend to be followed by negative

percentage returns and vice versa so that the overall prices for the wines in this category tends towards the mean wine price for each wine. As for the wines in the second category, It is slightly more difficult to venture an explanation for the autocorrelation structure observed since it appears to be more complicated than the first but a possible explanation of the negative autocorrelation observed in the first difference of the returns could be perhaps a regression of the rate of change of the returns (i.e. the gradient) to the mean rate of change, that is, larger positive rates of change of the returns tend to be followed by negative rates of change of the returns and vice versa so that overall, the rate of change or the gradient of the returns tends towards the mean rate of change for each wine in this category.

Finally, to further confirm and accentuate the fact that we have a clear separation of the wines into 2 categories based on autocorrelation structure, we decide to take features from the autocorrelation structure of all the wines to see if we can cluster them into 2 distinct clusters. We used the autocorrelation coefficients of the wines up to 10 lags as features and using a mixture of Gaussians clustering with a 2 Gaussian mixture model together with the expectation maximisation algorithm to find the maximum likelihood estimates of the 2 Gaussian mixture model, we noticed that there is clearly two distinct clusters as can be seen in figure 2.7 below.

FIGURE 2.7: k=2 clustering of wines

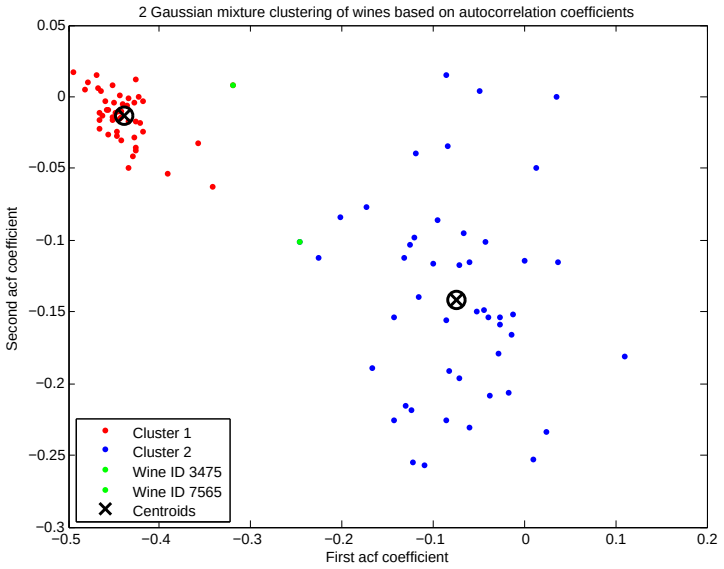


In addition, we note that the clustering of the wines into the two clusters using the 2 Gaussian mixture model corresponds very closely to our initial hypothesised classification of the wines into 2 categories, which we did based on just observing the acf coefficients of the wines. From figure 2.7 above, we note that for the red cluster we have

an average first acf coefficient of -0.45 and an average second acf coefficient very near 0. This corresponds very closely to our initial observation of the wines which we classify as being in the first category as having a very strong lag 1 autocorrelation structure. In addition, we note that the red cluster is very tightly clustered around the cluster mean which implies the wines classified as being in that cluster are all quite similar to one another. This also corresponds closely to our observation that the wines in the first category share very similar plots of their acf coefficients. The blue cluster, on the other hand, has an average first acf coefficient of -0.07 and an average second acf coefficient of -0.15 which supports our observation that there seems to be no interesting temporal autocorrelation structure when we just look at the wine returns of the wines in the second category. Unlike the red cluster, the blue cluster is not as tightly clustered around the cluster mean which suggests that there might be more variation among the wines in this cluster in terms of autocorrelation structure. This again corresponds to our observation of the general trend across the wines in category 2 (persistently large acf coefficient ranges across all lags), but they are not as similar as those in category 1 in terms of the overall shape of their acf plots.

In fact, using our observed classification as the "true" classification, we note that the clustering results only differs from our "true" observation by an average of 2 data points over 10 different clustering attempts. We plot the 2 clusters obtained below in figure 2.8, with the red dots representing the wines which the k-means algorithm classified as belonging to category 1, the blue dots representing the wines which the algorithm classified as belonging to category 2, and the green dots highlighting the 2 misclassified wines. We note that it is mainly the same 2 wines (wine ID 3475 and wine ID 7565) which are being misclassified in each run of the mixture of 2 Gaussians clustering algorithm.

FIGURE 2.8: Clustering wines using autocorrelation coefficients



As such, from the 2 Gaussian mixture model clustering of the data, we see that it not only highlights a distinct separation of the wines into 2 main classes based on autocorrelation structure, with the first cluster being comprised of wines which share a much more similar autocorrelation structure as compared to the second, but also that the clustering corresponds very closely closely to our observations above.

2.4 Regional similarities

A natural question arising from our classification of the wines into two categories, one with a simpler, more similar autocorrelation structure and the other perhaps with a more complicated one, is whether there exist some features of the wines which play a strong role in determining which category they fall into. Interestingly, from the acf vs lag plots, we note that the Burgundy, Bolgheri, and Rhone Liv-Ex 100 wines all fall into the second category. To check for possible relationships between region and autocorrelation structure, we also looked at box plots of acf coefficients across all the wines as a whole as well as grouped according to regions to see if we could discern any interesting differences in autocorrelation structure. For all the wines, we see the optimal lag beyond which we get noise appears to be around the 5th lag mark as seen in figure 2.9 where after the 5th lag point the whisker range and mean acf values seem to be very closely and consistently clustered around 0. Grouping the wines according to region, we note that we get an optimal lag of 3 for the Champagne wines as seen in figure 2.10 below. For the Rhone wines, the range and mean of acf values appears to be consistently near 0

after lag 130 as evident in figure 2.11 but again like in the case of wines in the second category above, this lag point seems pretty artificial as there is no sharp change in the mean or in the whisker range of the acf coefficients across all lags. When we observe the distribution of the acf values of the differenced returns for Rhone wines however, we see that the optimal lag is clearly lag 1 as evidenced in the drastic change in whisker range and the consistency of whisker range and acf mean values around 0 beyond lag 1 as seen in figure 2.12. For Bordeaux wines, we notice an optimal lag of around 4 as seen in figure 2.13 below. For the Bolgheri and Burgundy wines as seen in figures 2.14 and 2.15 respectively, we get an "optimal" lag of 115 and 140 respectively but again like the Rhone wines, the optimal lag appears artificial for similar reasons. Looking at the distribution of the acf values of the differenced returns for the Bolgheri and Burgundy wines, we see that the optimal lag is clearly lag 1 for similar reasons to the Rhone wines as seen in figure 2.16 and figure 2.17 respectively.

FIGURE 2.9: Box plots of acf over all lags of all wines

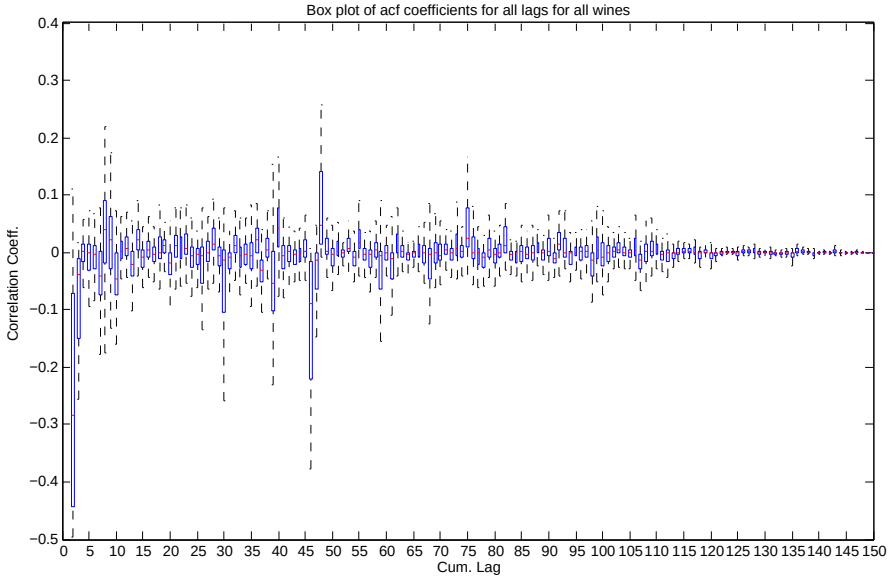


FIGURE 2.10: Box plots of acf of returns over all lags for Champagne wines

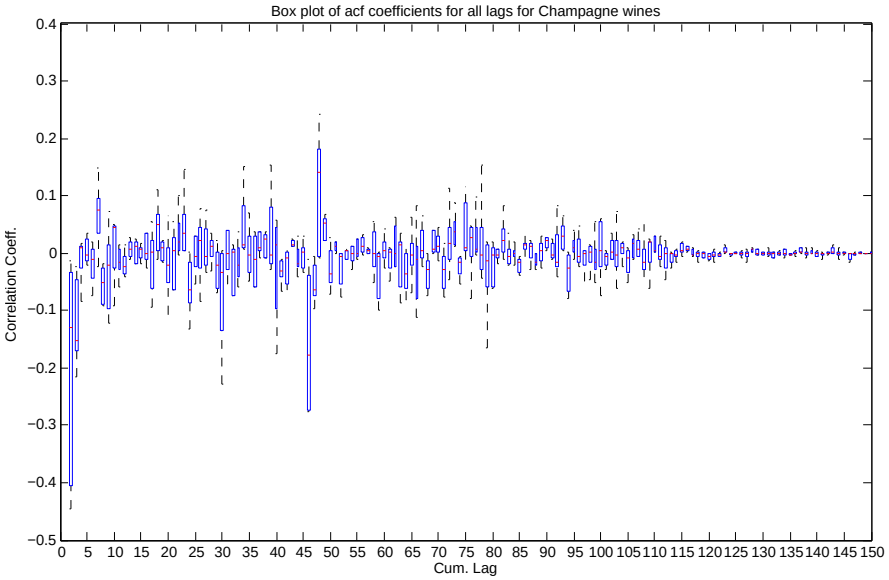


FIGURE 2.11: Box plots of acf of returns over all lags for Rhone wines

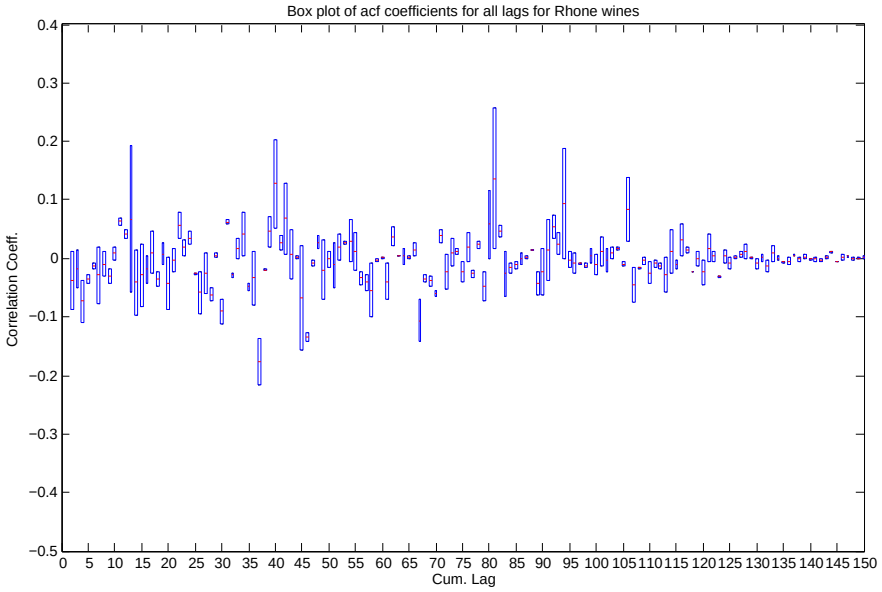


FIGURE 2.12: Box plots of acf of differenced returns over all lags for Rhone wines

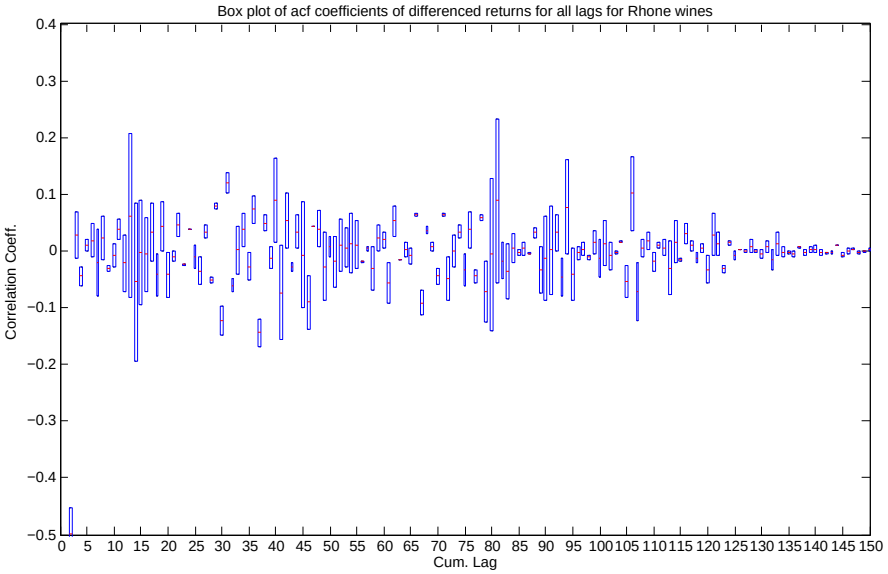


FIGURE 2.13: Box plots of acf of returns over all lags for Bordeaux wines

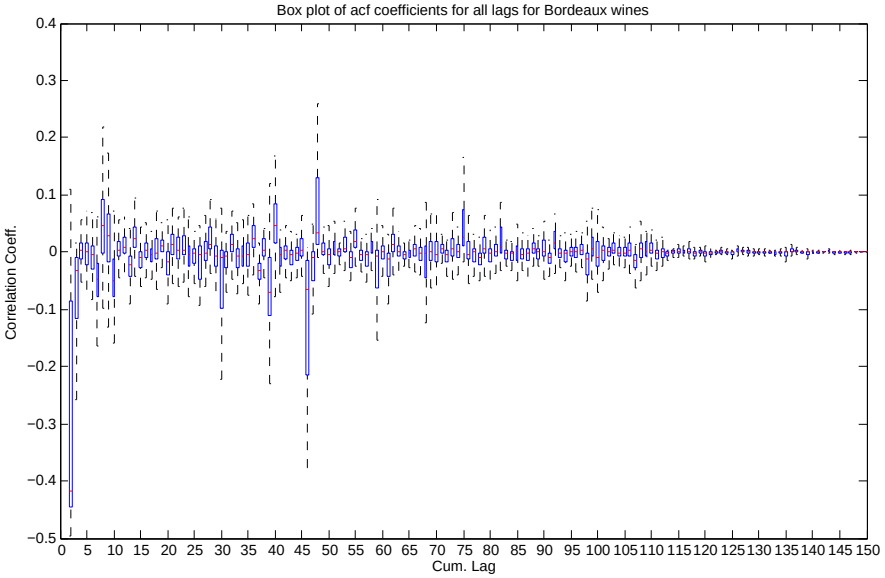


FIGURE 2.14: Box plots of acf of returns over all lags for Bolgheri wines

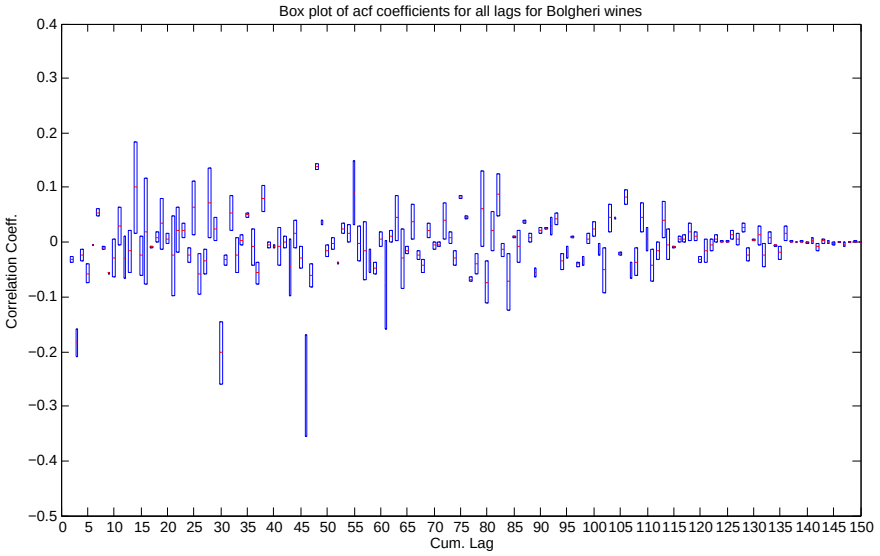


FIGURE 2.15: Box plots of acf of returns over all lags for Burgundy wines

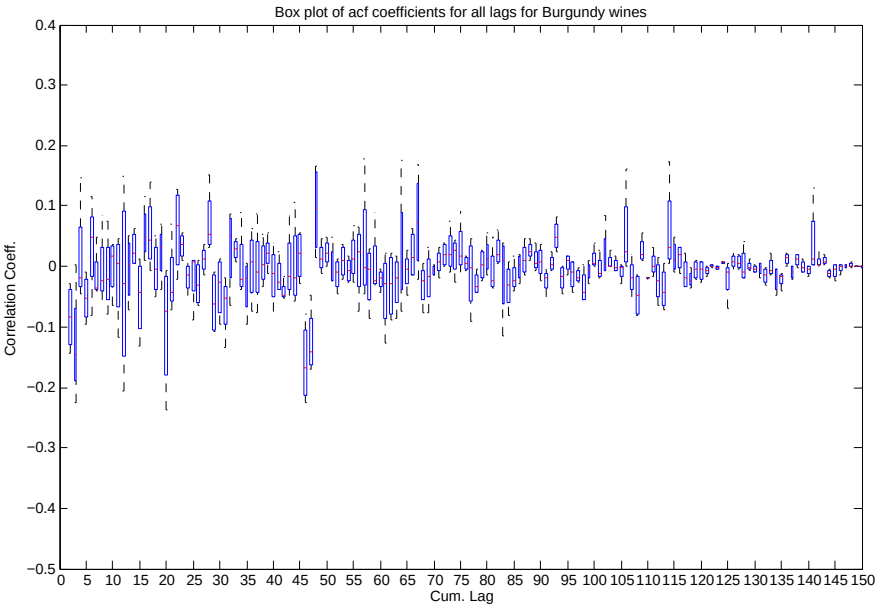


FIGURE 2.16: Box plots of acf of differenced returns over all lags for Bolgheri wines

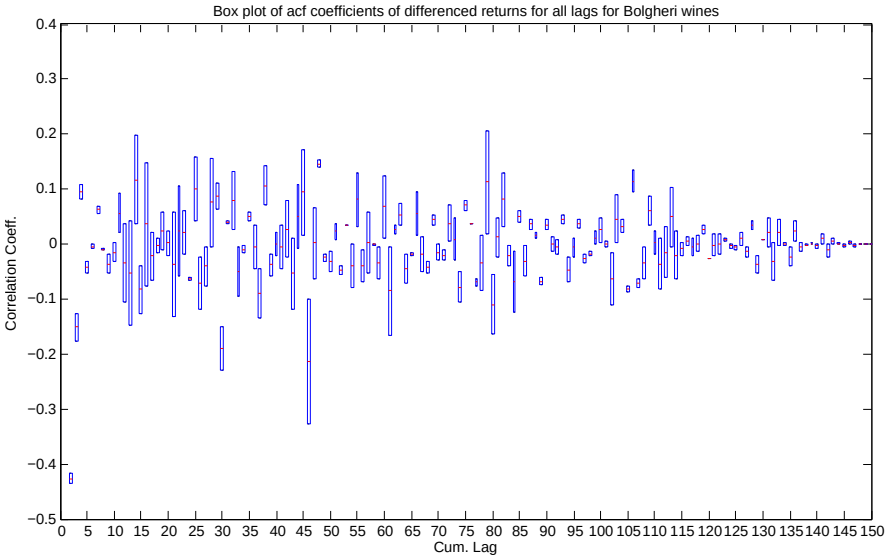
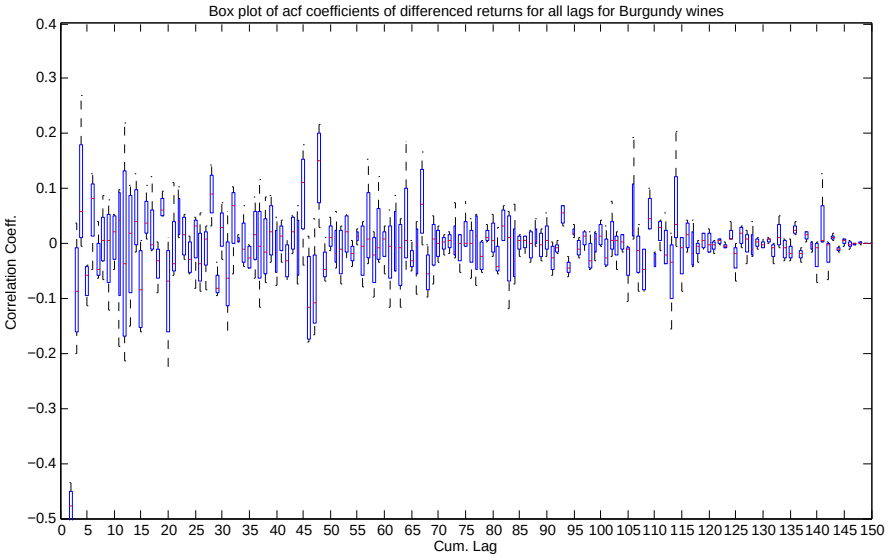


FIGURE 2.17: Box plots of acf of differenced returns over all lags for Burgundy wines



The box plots of acf values across lags for the wines grouped according to region thus supports the idea that the returns of Rhone, Bolgheri and Burgundy wines have autocorrelation structures which do not exhibit any significant relationship but have first differenced returns with a significant lag 1 temporal relationship. Apart from this, no other interesting relationships between region and autocorrelation structure could be discerned. A similar attempt to find interesting relationships between autocorrelation structure and vintage was made by grouping the wines into vintages and looking at box

plots of the acf values for all time lags but no interesting difference between vintage and autocorrelation structure that was discerned. We note, however, that since the Liv-Ex 100 wines mainly come from Bordeaux with only a few coming from the other regions (e.g. we only have 2 Bolgheri wines) and also since we only have a few Liv-Ex 100 wines for each vintage year, it is difficult to conclude if the observed difference between regions and autocorrelation structure is actually significant or just due to chance. Future work should perhaps draw on more region and vintage data from the rest of the other wines in the database to see if such a generalisation based on either region or vintage and autocorrelation structure appears plausible.

2.5 Simple model fitting

2.5.1 ARMA and ARIMA

The exponential decay observed in the plots of the acf of percentage returns of the first group of Liv-Ex 100 wines as well as that of the first difference of the second suggests dependence in either observed returns or the random process or both across time. In this subsection, we will first attempt to fit simple autoregressive moving average (ARMA) as well as autoregressive integrated moving average (ARIMA) processes to the wine returns. We begin by defining an ARMA(p,q) process as follows:

$$Y_t = \phi_1 Y_{t-1} + ... + \phi_p Y_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - ... - \theta_q \epsilon_{t-q} \tag{2.1}$$

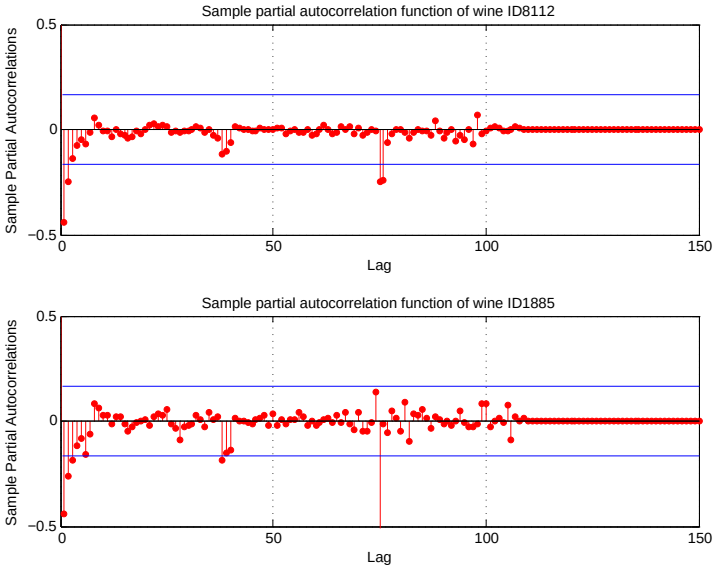
where $\phi_1 ... \phi_p$ and $\theta_1 ... \theta_q$ are some constants which can be estimated by maximum likelihood. Essentially, an ARMA(p,q) process is a process in which the current value in the time series depends linearly on p of its past values and q past values of the random term. An ARIMA(p,d,q) process is simply an process in which the d-th difference of the series is an ARMA(p,q) process. Here it is noteworthy that a key assumption in the ARMA models is that of weak stationarity, that is, the mean and variance of the distribution of the random variables is constant across time [10]. This is a strong assumption, but we will just assume it holds for now, especially since we note from the plots of percentage returns that the mean percentage returns of the wines does not seem to change over time. We will consider models which are more flexible in the next subsection.

For the first group of wines, we used the Box-Jenkins method outlined in [10] to estimate the orders of the ARMA process which gives us q = 1 from the plots of the acf. To estimate p, we look at the partial autocorrelation function (pacf) of the wine returns.

Unlike the acf which measures the correlation between Y_t and Y_{t-k} for a given lag k, the pacf measures the correlation for a given lag k between Y_t and Y_{t-k} after removing the effect of the intervening variables $Y_{t-1}, Y_{t-2}, ..., Y_{t-k+1}$. Essentially, what this means is that after taking into account how both Y_t and Y_{t-k} are related to $Y_{t-1}, Y_{t-2}, ..., Y_{t-k+1}$, the correlation between Y_t and Y_{t-k} then becomes the partial correlation coefficient of Y_t and Y_{t-k}.

The plots of the pacf of the wines in this group suggest p=1 or 2 since for most wines the pacf values at lags 1 and 2 consistently spike above the $\pm$ 2 standard error confidence bounds, with the second spike much closer to the bounds than the first. An example of this is seen below in the pacf plot of wine ID 8112 in figure 2.18. We note that there are a few spikes in the pacf values in the middle of the series (around lag 75) which perhaps suggests either non-constant variance in the wine returns or possible outliers and points to a possible limitation of ARMA modelling for these wines. This is most acutely evident in the second pacf plot of wine ID 8112 in figure 2.18 where we see a huge dip in the sample pacf value at around lag 75.

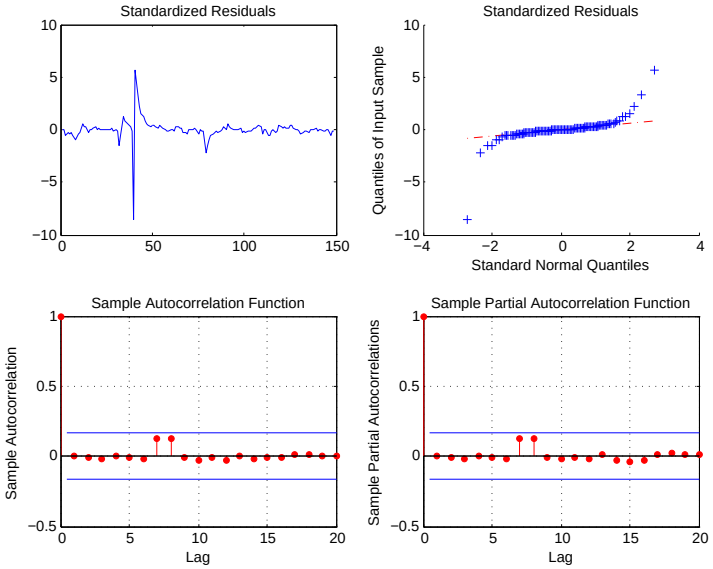
FIGURE 2.18: Pacf of wine ID 1885 and wine ID 8112



We first fit an ARMA(1,1) process to the wines in the first category. The residual diagnostics for most of the wines show evidence of non-normality, with QQplots highlighting thicker tails and slight dependence in standardised residuals which again suggests non-constant variance in the returns series. Acf and pacf plots of the residuals, however, seem alright. Figure 2.19 below shows an example of residual diagnostics of fitting the ARMA(1,1) model to wine ID 1885. We also tried fitting an ARMA(2,1) process but

found no improvement in residual diagnostics and thus, keeping in line with model simplicity, we will stick to using the ARMA(1,1) process to fit the wine returns for the wines in the first category.

FIGURE 2.19: Residual diagnostics of ARMA(1,1) fit for wine ID 1885



Finally, we fit the ARMA(1,1) model on the returns up to January 2014, leaving the returns data in February, March, April 2014 to see how well the forecasts from the model perform in comparison to the actual observed returns for those months and we got an average mean square error (MSE) value 3.92×10^{-5} overall the wines in the category.

The Box-Jenkins method for ARMA model fitting was also performed to the wines on the second group, with the optimal result based on looking at acf and pacf plots of the differenced returns being an ARIMA(2,1,1) model, as evident from the significant dips in the first 2 lags of the pacf function for most of the wines in this group. Again we note as before that there seems to be some non-constant variance in the data as we have random spikes and dips in the pacf plot for the wines. A plot of this phenomena is shown in figure 2.20 as well as the residual diagnostics for the same wine in figure 2.21 below.

FIGURE 2.20: Acf and pacf plot of wine ID 346

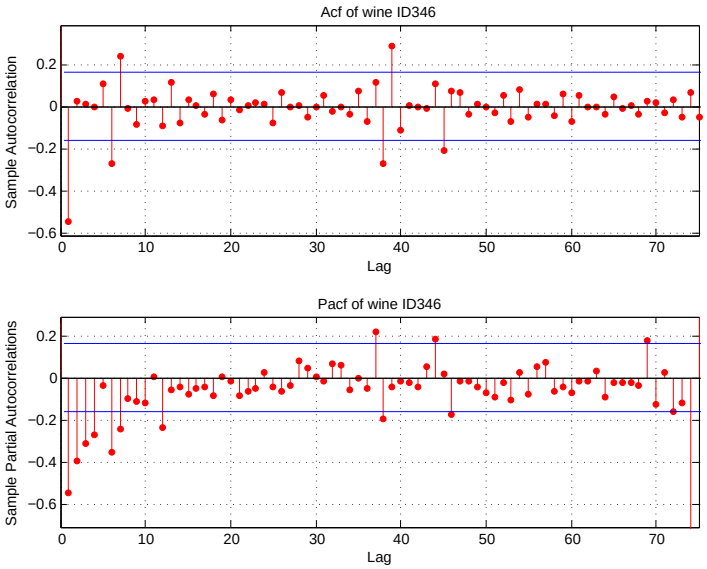
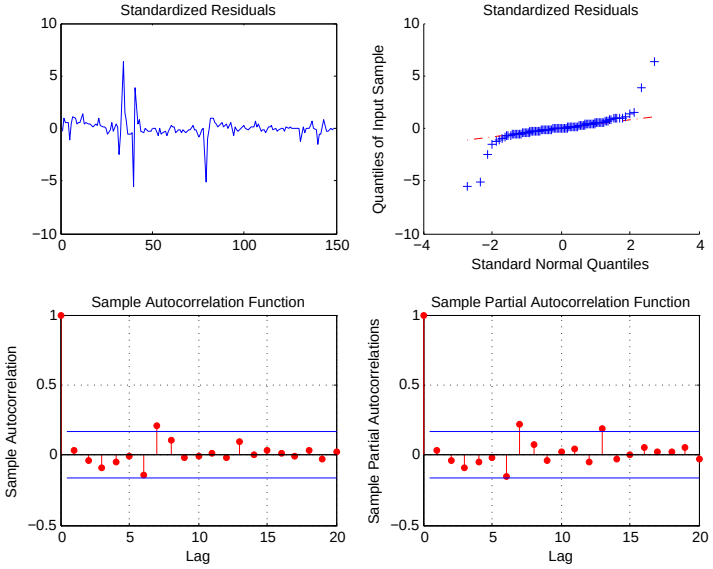


FIGURE 2.21: Residual diagnostics of ARIMA(2,1,1) fit for wine ID 346



Again we see the residual diagnostics suggest non-normality and non-constant variance as in the case of the wines in the first category, as seen in the fat tails of the QQplot of the residuals as well as the clear pattern in the plot of the standardised residuals vs. fitted values. We repeat the forecasting procedure as per the wines in category 1 and hold out February, March, and April returns data for forecasting and we get an overall slightly higher average MSE value of 8.21 $\times 10^{-5}$ for the wines in this category.

2.5.2 GARCH

To deal with the apparent non-normality of the residuals and the slight evidence of non-constant variance in the returns series of the wines, we first test for evidence of non-normality using both QQplots and the Jarque Bera test for non-normality at significance level $\alpha = 0.05$ and heteroscedasticity using the Engle test at $\alpha = 0.05$ in the returns of all the wines. We note that almost all of the wines display non-normality in the returns and most of the wines display significant heteroscedasticity at certain time lags which suggest either non-normality or non-constant variance in the random process. We thus looked at fitting a more general class of generalised autoregressive conditional heteroscedasticity (GARCH) models which do not assume stationarity, in particular, the variance of the distribution of variables across the series need not be constant. We begin by defining a GARCH(p,q) process as follows:

$$\sigma^2_{t|t-1} = \omega + \beta_1\sigma^2_{t-1|t-2} + ... + \beta_p\sigma^2_{t-p|t-p-1} + \alpha_1 Y^2_{t-1} + ... + \alpha_q Y^2_{t-q} \tag{2.2}$$

Essentially, this just says that the conditional variance of the series at any given time given the previous variance value ($\sigma^2_{t|t-1}$) is a linear combination of p past conditional variances and q past variances of the random process. Note that we assume the process $\{Y_t\}$ has zero mean, which we can get by subtracting the means of the wine returns from each of the wine returns series to centre it around 0.

We follow the model identification technique in [10] which suggests looking at the acf and pacf plots of the squared returns to see if they fit an ARMA(max(p,q),q) model. For the wines in category 1, we found that the acf of the squared returns for almost all the wines has a significant value at lag 1 and following that seems to be very close to 0. As for the pacf for the wines, we note that most of them display a prominent spike at lag 1 and a slightly less prominent dip at lag 2, suggesting an overall GARCH(2,1) or GARCH (1,1) model for the returns process. A plot of the acf and pacf of the squared returns of wine ID 1885 is shown in figure 2.22 below as well as the residual diagnostics of the GARCH(1,1) and GARCH(2,1) model when fitted to the returns of the same wine in figures 2.23 and 2.24 respectively.

FIGURE 2.22: Acf and pacf plots for squared returns of wine ID 1885

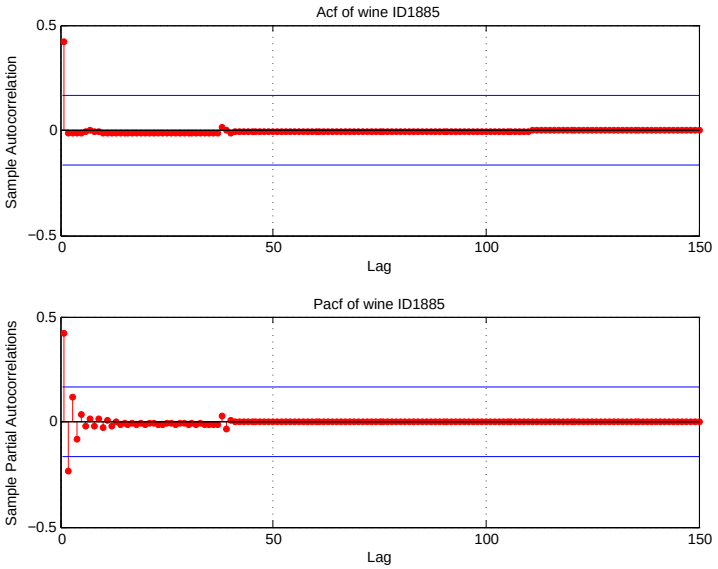


FIGURE 2.23: Residual diagnostics of GARCH(1,1) model for wine ID 1885

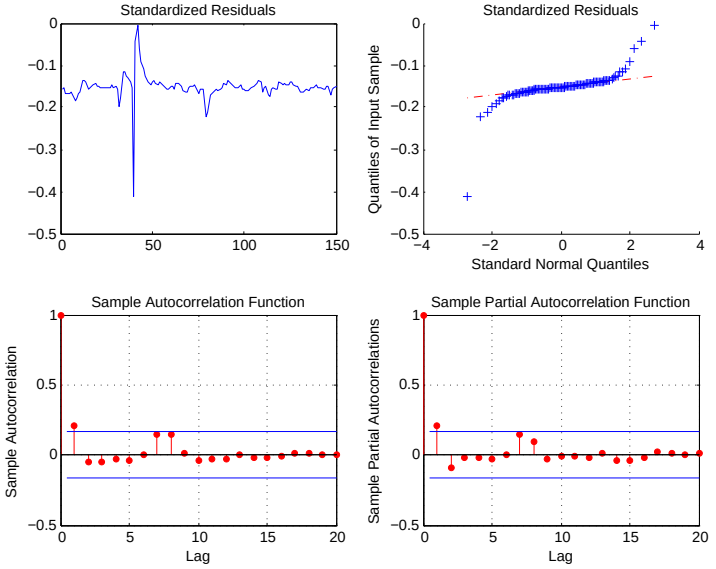
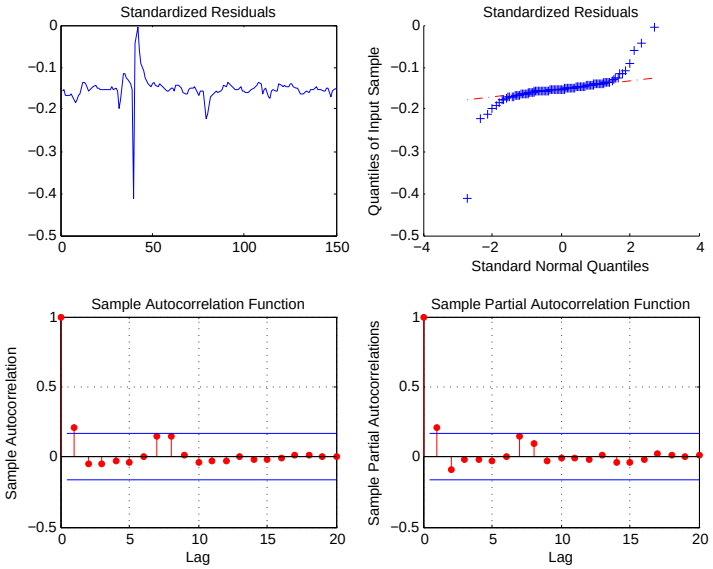


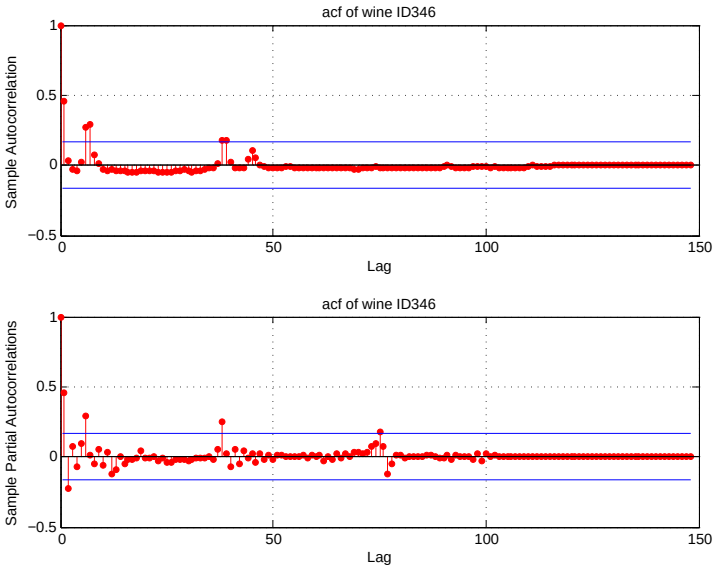
FIGURE 2.24: Residual diagnostics of GARCH(2,1) model for wine ID 1885



The residual analysis of both GARCH models seem similar and as before, we choose to focus on the GARCH(1,1) model for simplicity purposes. As compared to the residual diagnostics for the ARMA(1,1) model for this wine (see figure 2.19), the plot of standardised residuals vs. fitted values contains less extreme values and the QQplot appears more normal. We do note though that there is still evidence of heaviness in the tails. Splitting the data into training and testing like before and forecasting returns using the GARCH(1,1) model, we obtain a an average MSE of 7.95×10^{-5} for the wines in category 1, which is higher than what we got with the ARMA(1,1) model forecasting in the previous section.

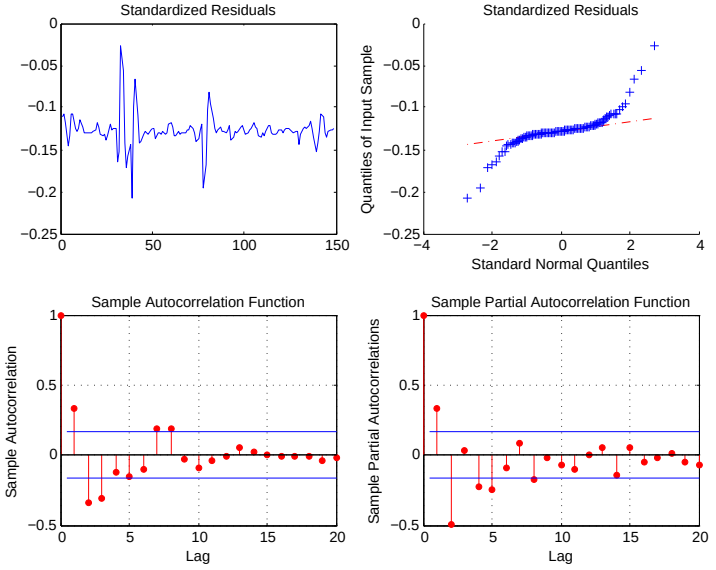
For the wines in category 2, we follow a similar method of model identification but this time we looked at acf and pacf of the squared first difference of the returns series instead of the returns series since the returns appeared to follow an ARIMA(2,1,1) process. For a fair portion of the wines in category 2, we note significant acf and pacf values at lag 1, but nevertheless also note a lot of noise surrounding the values of the acf and pacf values at other lags. A plot highlighting this phenomenon is seen in figure 2.25 below.

FIGURE 2.25: Acf and pacf plots for squared first difference returns of wine ID 346



We thus attempt to fit a GARCH(1,1) model on the first difference of the returns for the wines in this category. Residual diagnostics similarly show better standardised residual values but also display heavy tails in the QQplot of the residuals as well as the existence of a slight pattern in the plot of standardised residuals vs. fitted values as seen in figure 2.26 below.

FIGURE 2.26: Residual diagnostics of GARCH(1,1) model for the first difference of the returns of wine ID 346



Splitting the data into training and testing like before and forecasting returns using the GARCH(1,1) model on the first difference, we obtain an average MSE of 4.5×10^{-3} which is significantly higher than that of the ARIMA(2,1,1) model. To perhaps understand the reason why this is so, we plot both the forecasted returns and actual returns against time for the forecasted ARIMA(2,1,1) model and the GARCH(1,1) model on the first difference of the returns of wine ID 346 in figures 2.27 and 2.28 respectively below. We see that the forecasted returns for the GARCH(1,1) model on the first difference follows closely to the observed returns for the first 10 new test observations but deviates very significantly after that which results in the larger MSE between forecasted returns and actual returns. The forecasted returns for the ARIMA(2,1,1) model, on the other hand, just predicts values closer to the mean of the returns and thus returns values which on average do not deviate that far from the observed values. Also, it is important to note that the GARCH(1,1) model on the first difference does not fit all of the wines in category as well as the GARCH(1,1) model does on category 1 which could also result in larger deviance between forecasted values and actual values.

FIGURE 2.27: ARIMA(2,1,1) forecasted vs actual returns of wine ID 346

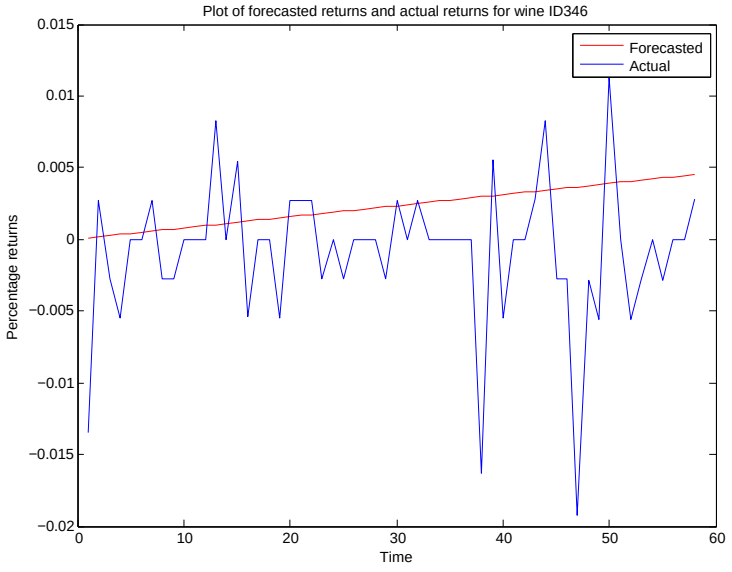
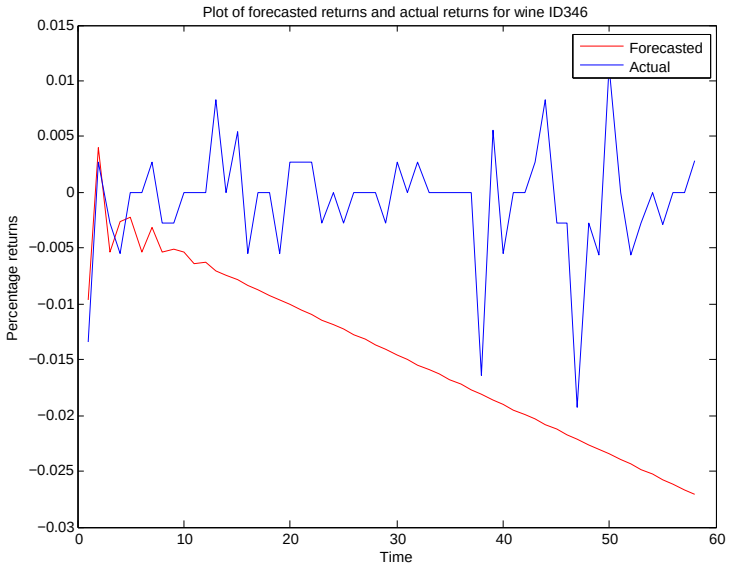


FIGURE 2.28: GARCH(1,1) forecasted vs actual returns of wine ID 346



Finally, we note that a common shortfall of both models as seen from the plots of forecasted vs. actual returns is that the predicted values do not capture the spikes and dips in the actual returns very well. For the ARIMA(2,1,1) model in figure 2.27, we simply get a straight line as a prediction and for the GARCH(1,1) model we get a prediction which follows the spikes and dips pretty closely at first, but following that, not only fails to capture the spikes and dips, but also deviates sharply from the trend. In chapter 3 we hope to better capture the spikes and dips in the actual returns process with Gaussian process regression.

Chapter 3

Gaussian processes

3.1 Gaussian process theory and regression

The formal definition of a Gaussian process as taken from Rasmussen and Williams [11] is as follows:

Definition 3.1. A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

Here we note three features of Gaussian processes which are particularly pertinent in the context of this project. The first being that Gaussian processes are completely specified by their mean (m($\mathbf{x}$)) and covariance ($k(\mathbf{x}, \mathbf{x}')$) functions (i.e. the "kernel"), which are defined as m($\mathbf{x}$) = $\mathbb{E}(f(\mathbf{x}))$, and $k(\mathbf{x}, \mathbf{x}^T) = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$ respectively. As such, the Gaussian process can be written as f($\mathbf{x}$) $\sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$. It is also common practice to assume the mean function is 0 everywhere since it is very easy to account for uncertainty over the mean by adding an extra term to the kernel and thus the kernel entirely determines the structure which can be captured by the Gaussian process model. More information on the choice and influence of such kernel functions will be detailed in the next section.

The second feature of Gaussian processes is its relationship to linear regression. In particular, we note that Gaussian process regression is essentially equivalent to Bayesian linear regression with an appropriate prior on the weight vector $\mathbf{w}$. Thus given a general learning problem of the form

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \tag{3.1}$$

where $\mathbf{w}$ is an N-dimensional weight vector and $\boldsymbol{\phi}(\mathbf{x})$ is a feature map $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^N$, with N usually > n, and $\boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), ..., \phi_N(\mathbf{x})]$ where $\phi_i(\mathbf{x})$ are basis functions, that is, non-linear mappings of the input instance $\mathbf{x}$ into a higher dimensional feature space, we can place a 0 mean isotropic Gaussian prior with variance $\sigma^2\boldsymbol{I}$, $\sigma \in \mathbb{R}$, over the weight vector $\mathbf{w}$. With this prior over $\mathbf{w}$ and integrating, we can rewrite (3.1) as $y \sim GP(0, \sigma^2\boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{x}'))$ [12].

The final feature of Gaussian processes is that they are non-parametric. As seen above, given a learning problem in the form of (3.1) we can place a prior on $\mathbf{w}$ and integrate to get $y \sim GP(0, \sigma^2\boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{x}'))$. Placing a prior on $\mathbf{w}$, however, is equivalent to placing a prior on the space of functions y($\mathbf{x}$) without parametrising y and thus Gaussian process regression is non-parametric [11].

Gaussian processes are used for both classification and regression, although in the context of this project we will only focus on the latter since our goal is to predict wine returns. For regression, we have training data of the form

$$t_n = y(\mathbf{x}_n) + \epsilon_n \tag{3.2}$$

where $\epsilon_n \sim N(0, \beta^{-1})$ is again assumed to be Gaussian white noise and t_n refers to the true observed target value, which, in the context of the wine data, would be the true observed returns of the wines at time index n. Using the properties of the Gaussian distribution as well as other methods outlined in [12], we derive the predictive distribution of a new target, that is, the conditional distribution of a new target value given all the target values observed, to be as follows

$$\begin{aligned} P(t_{N+1}|\mathbf{t}) &= N(\mathbf{k}^T\mathbf{C}_n^{-1}\mathbf{t}, c - \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{k}) && (3.3)\\ &= N(m(\mathbf{x}_{N+1}), \sigma^2(\mathbf{x}_{N+1})) && (3.4) \end{aligned}$$

where $\mathbf{k} = k(\mathbf{x}_n, \mathbf{x}_{N+1})$ is the kernel function as defined above applied from n = 1 to N, $c = k(\mathbf{X}_{N+1}, \mathbf{X}_{N+1}) + \beta^{-1} \in \mathbb{R}$, and $\mathbf{C} =$

$$\begin{bmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{bmatrix}$$

where $\mathbf{C}_N$ is the covariance matrix of the joint normal distribution of the observed targets defined as $C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}$, where δ_{nm} is a Kronecker delta which is one iff n = m and zero otherwise [12].

3.2 Kernel functions

A kernel function (:= $k(\mathbf{x}, \mathbf{x}')$) is essentially just a function of any inputs $\mathbf{x}$ and $\mathbf{x}$' such that the resulting function is positive semi-definite [13]. For Gaussian process regression, we use the covariance kernel cov(f($\mathbf{x}$), f($\mathbf{x}$')) = k($\mathbf{x}$,$\mathbf{x}$') to define the prior covariance between any two function values, that is, we use the kernel to specify how similar the values of the function evaluated at each data point $\mathbf{x}$ and $\mathbf{x}$' are [14]. With this similarity information, we then get a better idea of which functions are more likely than others under the covariance prior. For instance, if we believe that function values evaluated at two very close data points $\mathbf{x}$ and $\mathbf{x}$' should be very similar, we can then choose a covariance kernel which incorporates such information.

Some examples of commonly-used covariance kernels are shown in table 3.1 below:

TABLE 3.1: Common covariance kernels

Kernel	
Squared exponential covariance kernel (SE)	$k(\mathbf{x}, \mathbf{x}') = \sigma^2 exp(\frac{-\lvert\mathbf{x}-\mathbf{x}'\rvert^2}{2l^2})$
Degree p polynomial kernel	$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + \sigma^2)^p$
Periodic kernel (Per)	$k(\mathbf{x}, \mathbf{x}') = \sigma^2 exp(-\frac{2}{l^2} sin^2(\pi \frac{\mathbf{x}-\mathbf{x}'}{p}))$
Linear kernel (Lin)	$k(\mathbf{x}, \mathbf{x}') = \sum_{d=1}^{D} \sigma_d^2 \mathbf{x}_d \mathbf{x}'_d$
Rational Quadratic kernel (RQ)	$(1 + \frac{\lvert\mathbf{x}-\mathbf{x}'\rvert^2}{2\alpha l^2})^{-\alpha}$
Matern	$\frac{2^{1-\nu}}{\Gamma(\nu)} (\frac{\sqrt{2\nu}\lvert\mathbf{x}-\mathbf{x}'\rvert}{l})^\nu K_\nu (\frac{sqrt2\nu\lvert\mathbf{x}-\mathbf{x}'\rvert}{l})$

In addition, we note that white noise can be modelled by the SE kernel with a short lengthscale (l). As l tends to 0, we get a white noise (WN) kernel and draws from a Gaussian process with a WN kernel are simply iid draws from a Gaussian random variable [14].

Here, we note two important properties of kernel functions, that is, additive and multiplicative closure. Thus the addition and multiplication of any number of kernels will always result in a valid kernel. The proofs of which are quite trivial and has been detailed fully by Shawe-Taylor and Cristianini and thus will be omitted here [13]. These properties, however, will be will be very useful in constructing new kernels from old ones so as to capture more prior information regarding the structure of the data. A table of the more interesting structures and regression models we can capture just by combining kernels in such a fashion has been compiled by Duvenaud [14] and we reproduce some of the more pertinent ones as follows:

TABLE 3.2: Structures and regression models captured by combination of kernels

Regression Model	Kernel
Linear regression	WN + Lin
Polynomial regression	Π Lin + WN
Trend, Cyclical, Irregular	Σ SE + Σ Per +WN
Time-changing variance	SE + Lin × WN
Semi-parametric	Lin + SE + WN

Lastly, before concluding this section, we would like to acknowledge that most of the information concerning the properties of covariance kernels and how they combine together to capture common regression models has been kindly provided by Duvenaud in his thesis. We direct the curious reader there for more information and details [14].

3.3 Gaussian process regression on data

Based on what we have done so far, there are two main ways of proceeding with Gaussian process regression on the wine dataset. The first models and forecasts the returns by using a Gaussian process based on ARMA/ARIMA models highlighted in the previous chapter. As seen in chapter 2, we could reasonably fit an ARMA model to the wine returns of the form

$$Y_t = \phi_1 Y_{t-1} + ... + \phi_p Y_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - ... - \theta_q \epsilon_{t-q} \tag{3.5}$$

$$= \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{j=i}^{q} \theta_j \epsilon_{t-j} + \epsilon_t \tag{3.6}$$

where the returns Y_t is a linear combination of p past return values and q past values of the random term ϵ_t. For the wines in the first category, the returns will be a function of the previous return value since the ARMA(1,1) fitting was found to be ideal for such wines. Thus we have the following model for the wine returns:

$$Y_t = \mathbf{w}\boldsymbol{\phi}(Y_{t-1}) + \epsilon_t \tag{3.7}$$

where $\boldsymbol{\phi}$ is some feature map as explained above. For the wines in the second category, the first differenced returns sequence will be a function of the previous two first differenced sequence values based on the optimal ARIMA(2,1,1) fitting in chapter 2. We thus

get a model for the first differenced returns sequence of the following form:

$$D_t = \mathbf{w}\boldsymbol{\phi}(\mathbf{D}) + \epsilon_t \tag{3.8}$$

where $\mathbf{D} :=$

$$\begin{bmatrix} D_{t-1} \\ D_{t-2} \end{bmatrix}$$

As shown in the first section of this chapter, this is equivalent to Gaussian process regression by incorporating a suitable prior on the weight vectors $\mathbf{w}$ for both cases.

The second way to proceed with Gaussian process regression is to model and forecasts the variance of the process by combining Gaussian processes and GARCH models which were also highlighted in the previous chapter and also recently seen in the financial literature where Gaussian process and GARCH models are combined to predict stock market volatility with superior performance over the standard GARCH process [8], [9].

We will only employ the first method in this section and compare the results of this method to the results we obtained in the previous chapter as well as a couple of benchmarks. We ignore the second method mainly because it is mainly focused on volatility forecasting which is not the aim of this project. In addition, we found the ARMA/ARIMA model fitting and forecasting to be much more favourable as compared to GARCH model fitting in the previous chapter, suggesting that perhaps modelling the conditional variance as a linear function of previous conditional variance terms and the variance of the random process might not be the most suitable for our data. Most of the code used to implement the regression comes from Rasmussen and William's Gaussian process for Machine Learning Matlab toolbox [15].

3.4 GP-ARMA/GP-ARIMA regression

3.4.1 Category 1 wines

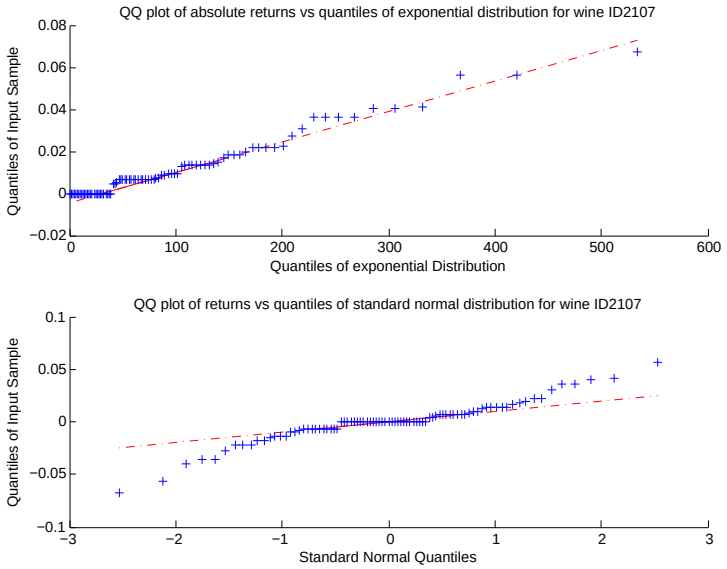
Before fitting a Gaussian process model to the returns of the wines in the first category, we first need to ascertain the data likelihood of the returns data, as that would influence the method in which we conduct inference to train our Gaussian process model as well as the resulting goodness of fit of the model. As seen from chapter 2, the returns data of the wines in the first category seem to display heavier tails as evident in the residual diagnostics of the ARMA and GARCH model fitting to the category 1 wines in the previous chapter (refer to 2.19 for ARMA fitting and 2.23 for GARCH fitting). Given

that this is so, the Gaussian likelihood might not be that good a fit to the data as compared to some other distribution with fatter tails like the Laplace distribution for instance.

We thus fit a mean 0 Gaussian distribution for all the wines in this category and got an average maximal log likelihood value of 141.18 as well as an optimal average variance parameter of $\sigma^2_{Gaussian} = 0.05$. Using the Laplace distribution to fit the data, we get an average variance parameter of $\sigma^2_{Laplace} = 0.01$ with a higher average maximal log likelihood value of 236.68, suggesting that the Laplace distribution might be a better fit for for the data.

To better confirm the expediency of the Laplace fit to the wine returns compared to the Gaussian one, we look at QQ plots of the wine returns vs. the quantiles of the standard normal distribution in the case of fitting the Gaussian likelihood to the data, and the quantiles of the exponential distribution in the case of fitting the Laplace likelihood to the data, using the fact that if $X \sim Laplace(0, b)$, then $|X| \sim exp(\frac{1}{b})$. In general, the QQ plots of the absolute returns vs. the quantiles of the exponential distribution look much better than the returns vs. the standard normal quantiles and we show the plots for wine ID 2107 below in figure 3.1 as an example. From first plot of the absolute returns vs. the quantiles of the exponential distribution, we see that the absolute returns of wine ID 2107 seem to lie mainly on a straight line which suggests that the exponential fit, and thus the Laplace fit, is probably a good one. From the second plot of the returns vs. the quantiles of the standard normal distribution, we see that the returns of wine ID 2107 appear to curve away from the line towards both ends which not only suggests a poor fit, but also heavy tails.

FIGURE 3.1: QQ plots comparing Gaussian and Laplace likelihoods for wine ID 2107



Finally, we look at fitting a Gaussian process model with both likelihoods to see if there is any difference between the MSE results, which could further support the choice of one likelihood over the other. We first chose the SE kernel as the covariance function for the Gaussian process regression on the dataset due to its stationary and non-degenerate properties. That is to say, the covariance function when applied to any two input points $\mathbf{x}$ and $\mathbf{x}$' only depends on the relative position of the $\mathbf{x}$ and $\mathbf{x}$' and not on their absolute location (stationarity), and the covariance kernel is the inner product of an infinite number of basis functions (non-degeneracy) [11]. The stationary characteristic is important as it corresponds to the stationarity of the wine returns as observed in the previous chapter. As for the non-degeneracy, we value this property due to the fact that it increases the flexibility of the function learned which could result in a better fit to the data.

For the Gaussian data likelihood, we use exact inference to compute the posterior as well as to approximate the negative log-marginal likelihood. Choosing the optimal hyperparameters σ^2 and l of the SE kernel from a range of 0.1 to 2 and 0.1 to 1 with increments of 0.1 respectively using cross-validation, as well as using the optimal $\sigma^2_{Gaussian}$ parameter for the Gaussian likelihood as obtained above, we found the optimal hyperparameters to be $\sigma^2 = 0.1$ and $l = 0.5$ respectively. With these optimal hyperparameter values, we obtained an optimal MSE of 3.56 $\times 10^{-5}$, which is smaller than the one obtained by the ARMA(1,1) forecasting in the previous chapter. A surface plot of the MSE as well as a plot of the predicted returns vs. actual returns of wine ID 8112 is shown in figures 3.2 and 3.3 below.

FIGURE 3.2: Surface plot of average MSE for category 1 wines

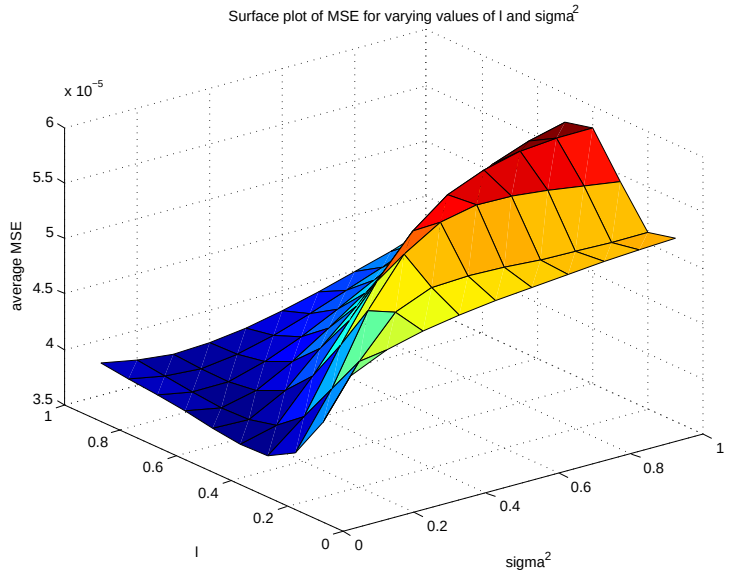
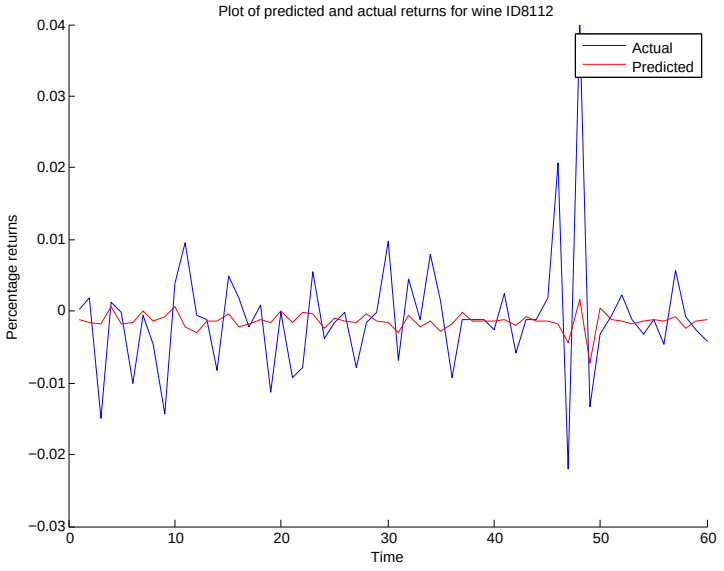


FIGURE 3.3: Predicted vs. Actual wine returns of wine ID 8112



We repeat the same procedure using the Laplace likelihood function with the optimal $\sigma^2_{Laplace}$ parameter as shown above instead of the Gaussian likelihood function. Using the Variational Bayesian approximate inference algorithm since exact inference is only compatible with Gaussian likelihoods, we get optimal values of 0.5 and 1 for σ^2 and l respectively and an optimal MSE of 3.50 $\times 10^{-5}$, which is lower than the MSE in the previous case with Gaussian likelihood and exact inference. This again supports our initial hypothesis that the Laplace likelihood would be more appropriate compared

to the Gaussian one for the data and as such, for the rest of this subsection, we will choose to assume the returns data for the wines in this category comes from a Laplace distribution. The surface plot of the average MSE for the different combinations of hyperparameter values as well as a plot of the predicted returns vs. actual returns of wine ID 8112 is shown in figures 3.4 and 3.5 below.

FIGURE 3.4: Surface plot of average MSE for category 1 wines

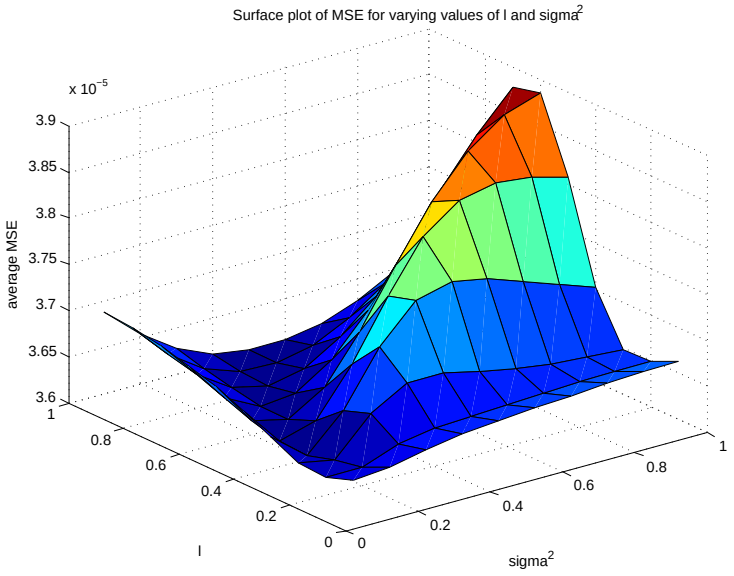
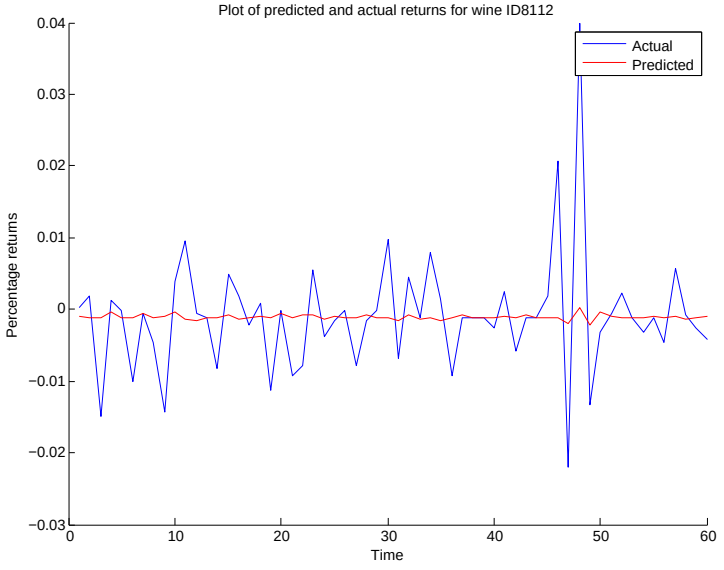


FIGURE 3.5: Predicted vs. Actual wine returns of wine ID 8112



In addition, from figures 3.3 and 3.5, we note that we seem to be predicting near 0 all

the time. Using predicting 0 all the time as a benchmark, we find that the we get an MSE of 3.65 $\times 10^{-5}$. which is higher than using GP regression with the SE kernel. We also attempted to use other covariance functions with the Laplace data likelihood like the RQ kernel which is a squared mixture, that is, an infinite sum, of SE kernels. With the RQ kernel and the Laplace likelihood, we get an optimal MSE of again 3.50 $\times 10^{-5}$ with optimal parameters of $\sigma = 0.1$, $\mathrm{l} = 0.2$, and $\alpha = 3$. We omit plots of predicted vs actual returns as the plots look very similar to figure 3.5.

We also tried combining kernels to achieve a time-changing variance model as detailed in table 3.2. Using the SE + Lin $\times$ WN combination and tuning the parameters for each individual kernel, we get optimal parameters of $\sigma_{WN} = 0.3$, $l_{Lin} = 2$, $\sigma_{SE} = 0.1$, $l_{SE} = 0.6$, and an optimal MSE value of 3.54 $\times 10^{-5}$ which is lower than predicting 0 all the time and using the Matern kernel but higher than using the SE and RQ kernels. We also tried combining two SE kernels, one with a longer length-scale and the other with a shorter one so as to try to capture both slow and fast variation in the data. With this kernel combination, we get a slightly lower optimal MSE value of 3.52 $\times 10^{-5}$, with optimal length-scales of 0.1 and 0.6, but still not as low as the MSE obtained from using the RQ and the SE kernels. Again, since the plot of predicted vs. actual returns resulting from using these kernel combinations look very similar to that when we used the RQ and SE kernels, we will omit showing the plots.

Finally, we note that equation (3.7) above implies that we are using only the previous return value for our Gaussian process regression. This effectively means that we are using only 1 training data point (the previous return value) as input to the covariance kernel, which might explain why we do not seem to capture that much variation in the returns data as evident from figure 3.5 above. To cope with this issue, we repeated the same procedure as before, but using 3 previous return values as inputs such that the regression model is modified to be of the following form:

$$Y_t \quad = \quad \mathbf{w}\boldsymbol{\phi}(\mathbf{Y}) + \epsilon_t \tag{3.9}$$

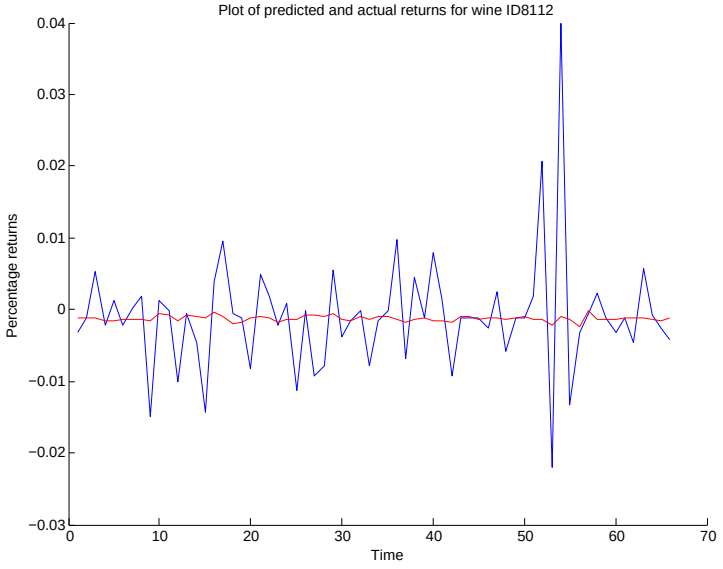
where $\mathbf{Y} :=$

$$\begin{bmatrix} Y_{t-1} \\ Y_{t-2} \\ Y_{t-3} \end{bmatrix}$$

With this modified model and SE kernel, we obtained an optimal MSE value of 3.42 $\times$ 10^{-5}, with optimal hyperparameters of $\sigma^2 = 0.2$ and $l = 1$ trained by cross-validation, which is only slightly smaller than the MSE value obtained with the same kernel but trained with just the previous return value. For clarity, this modified Gaussian process

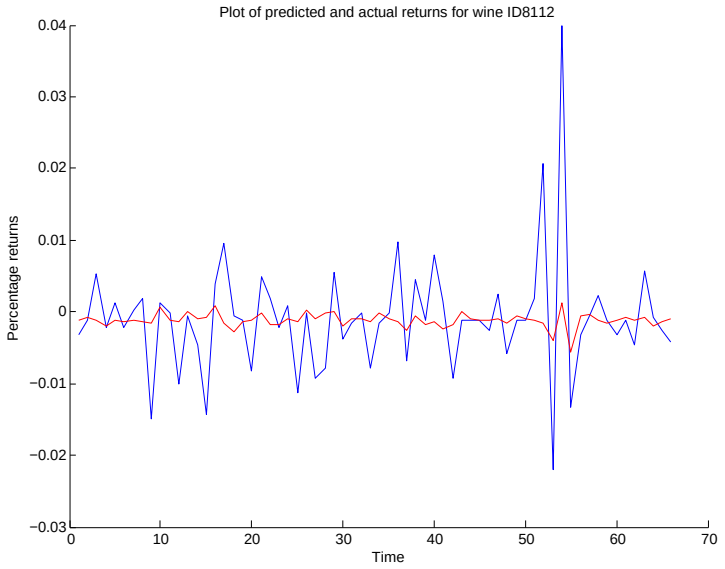
regression model will be known as the newer or modified model and the previous model will be known as the earlier model for the rest of this chapter. The plot of predicted vs actual returns of wine ID 8112 with this optimal combination of hyperparameters is shown below in figure 3.6. As we can see from the plot, our learned predictive function seems to be a little bit wigglier than using the same kernel but with the previous model as seen above in figure 3.5, which suggests we are capturing more information in the wine returns series with this newer model.

FIGURE 3.6: Predicted vs. Actual wine returns of wine ID 8112



Again we repeat the above procedure of testing other different kernels on this modified regression model. With the RQ kernel, we get a slightly lower MSE of 3.41×10^{-5} with optimal hyperparameters of $\sigma^2 = 0.1$ and $l = 1$ and $\alpha = 3$. With the Matern kernel, we get a similar optimal MSE of 3.41×10^{-5} with optimal parameters of $\sigma^2 = 0.1$ and $l = 0.5$. Again we tried various kernel combinations as before and got even lower (albeit only slightly) optimal MSEs of 3.40×10^{-5} for both the SE + Lin $\times$ WN combination and the sum of 2 SE kernels, one with a longer length-scale and the other with a shorter one. As can be seen in the plot of predicted vs. actual returns of wine ID 8112 in figure 3.7 below and comparing it to the similar plots using the earlier version of Gaussian process regression with only 1 data point as input as well as the newer version of Gaussian process regression using the previous 3 data points as input but using the SE kernel in figures 3.5 and 3.6 respectively, using the SE + Lin $\times$ WN kernel combination gives us a wigglier prediction which conforms more closely to the peaks and dips in the returns plot. We are thus able to capture more information and variation in the data.

FIGURE 3.7: Predicted vs. Actual wine returns of wine ID 8112



Summaries of the results are provided in tables 3.3, 3.4, and 3.5 and a more detailed discussion of the results will also be provided in the next section.

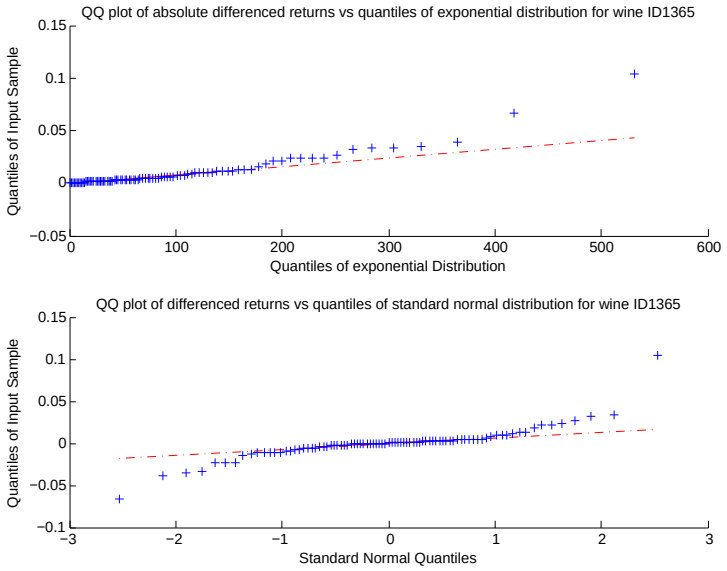
3.4.2 Category 2 wines

Moving on to the wines in the second category, we employed the same procedure of first checking to see if the Gaussian likelihood or the Laplace likelihood fits the data better. In the case of the wines in category 2, however, we need to check the goodness of the Gaussian or the Laplacian fit to the first difference of the returns due to the ARIMA(2,1,1) structure as shown in chapter 2 and equation (3.8). Again just comparing maximal log likelihoods, we get an average log likelihood of 222.85 with an optimal variance parameter of $\sigma^2_{Gaussian} = 0.05$ when fitting a 0 mean Gaussian distribution to the first difference of the returns. Using the Laplace distribution to fit the data, we get an average variance parameter of $\sigma^2_{Laplace} = 0.01$ with a higher log likelihood value of 250.24, again suggesting that the Laplace distribution might be a better fit for for the data.

Again, we look at QQ plots of the first differenced wine returns vs. the quantiles of the standard normal distribution in the case of fitting the Gaussian likelihood to the data, and the absolute values of the first differenced wine returns vs. quantiles of the exponential distribution in the case of fitting the Laplace likelihood to the data. We note that like in the case of the wines in category 1, the QQ plots of the absolute first differenced returns vs. the quantiles of the exponential distribution look much better

than the returns vs. the standard normal quantiles. We show the plots for wine ID 1365 below in figure 3.1 as an example. From the first plot of the absolute differenced returns vs. the quantiles of the exponential distribution, we see that the absolute differenced returns of wine ID 1365 seem to lie mainly on a straight line with only a slight curvature towards the ends on one side which suggests that the exponential fit, and thus the Laplace fit, is not too bad. From the second plot of the differenced returns vs. the quantiles of the standard normal distribution, we see that the returns of wine ID 1365 clearly curves away from the line at both ends which again suggests both a poor fit as well as heavy tails.

FIGURE 3.8: QQ plots comparing Gaussian and Laplace likelihoods of wine ID 1365



Finally as in the case of before, we proceed to Gaussian process regression using both Gaussian and Laplace likelihoods just to compare the difference in MSE. Using the squared exponential covariance kernel first together with the Gaussian likelihood and the exact inference method to train the returns data of the wines in this category, we get optimal σ^2 and l hyperparameters of 1 and 0.7 respectively using cross validation, leading to a minimum average MSE of 7.63×10^{-5} which is smaller than that obtained by both the ARIMA(2,1,1) and the GARCH(1,1) forecasting in chapter 2. A surface plot of the average MSE for different combination of the hyperparemeters as well as a plot of the predicted vs. average returns of wine ID 346 is shown in figures 3.9 and 3.10 below.

FIGURE 3.9: Surface plot of average MSE for category 2 wines

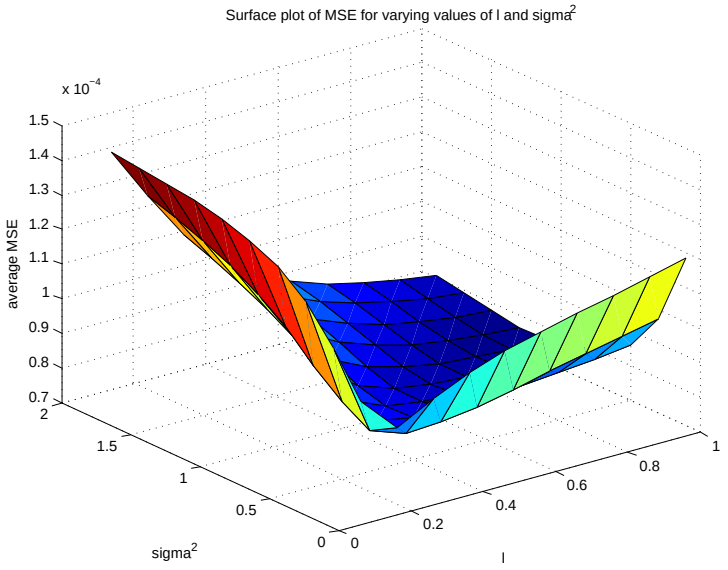


FIGURE 3.10: Predicted vs. Actual wine returns of wine ID 346

Using the Laplace likelihood and Variational approximate inference algorithm instead of the Gaussian likelihood with exact inference, we get an optimal MSE value of 6.94×10^{-5}, with optimal hyperparameters of $\sigma^2 = 2$ and $l = 0.7$ using cross-validation. This time, the optimal MSE value using the Laplace likelihood and the Variational Bayesian approximate inference algorithm is a marked improvement over that when we use the Gaussian likelihood and exact inference, and thus for the rest of this subsection, we will choose to use the Laplace data likelihood instead of the Gaussian one. The average MSE

for different combination of the hyperparemeters and the predicted vs. average returns of wine ID 346 is shown in figures 3.11 and 3.10 below.

FIGURE 3.11: Surface plot of average MSE for category 2 wines

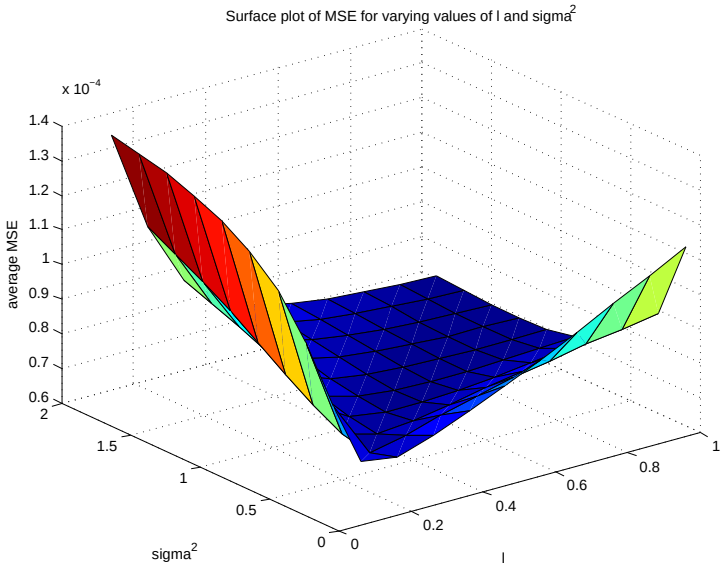
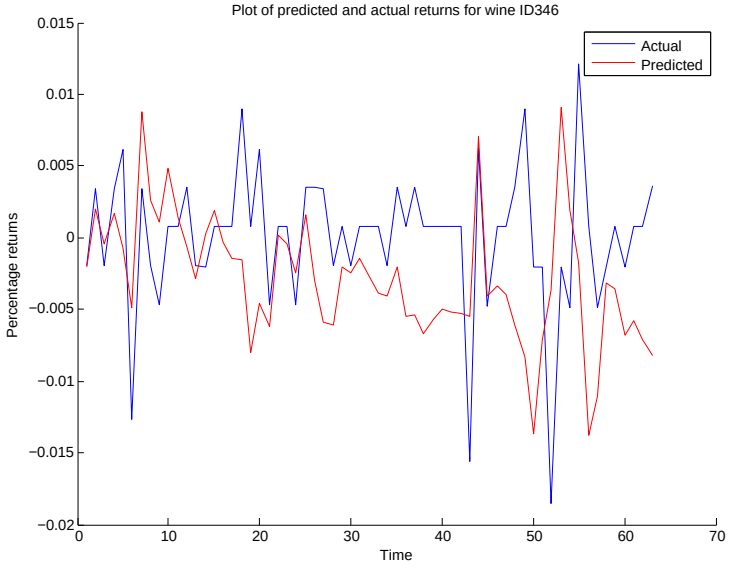


FIGURE 3.12: Predicted vs. Actual wine returns of wine ID 346



Again as before, we tried a variety of other kernels to see if we can capture more of the variation in the data and possibly lower the MSE values we get. With the RQ and Matern kernels, we get a slightly higher MSE of 6.96×10^{-5} and 6.99×10^{-5} respectively. Like in the case for the category 1 wines, using fancier combinations of kernels to achieve

both a time-changing variance model as well as capturing slow and fast variation does not work that well with this dataset and we get optimal MSE values of 7.20×10^{-5} and 7.55×10^{-5} for these models respectively. These MSE values are clearly higher than using basic SE, RQ and Matern kernels, though they are also clearly lower than the ones we obtained when using the ARIMA and GARCH models.

Finally, we also tried incorporating more data points into the regression formula, that is, we use the previous k values of the first difference with $k > 2$ just to see if increasing the number of input points in our covariance kernel would result in learning a function which could perhaps capture more information about the data and thus give us a lower MSE. The result, however, was not very different from just using the previous two values and the resulting plots look pretty much similar. In addition, unlike the previous case of the wines from the first category, using GP regression on the wines from the second category already give us MSE results and predicted vs. actual returns plots which are clearly superior to that obtained by the benchmark of predicting 0 as well as the previous ARIMA and GARCH models and so we omit going down this route to avoid creating unnecessarily complicated models. Summaries of the results are provided in tables 3.6 and 3.7 and a more lengthy discussion of the results will be provided in the next section.

3.5 Summary and discussion of results

We summarise the results of both Gaussian process regression using various covariance kernels as well as a comparison between the optimal Gaussian process regression result and a few other benchmarks for the wines in both categories in the tables as follows:

TABLE 3.3: Summary of Gaussian process regression results for wines in category 1

Kernel	Hyperparameters	MSE
SE	$\sigma = 0.1$, l $= 1$	3.50×10^{-5}
RQ	$\sigma = 0.1$, l $= 0.2$, $\alpha = 3$	3.50×10^{-5}
Matern	$\sigma = 0.1$, l $= 1$	3.55×10^{-5}
SE + Lin $\times$ WN	$\sigma_{WN} = 0.3$, $l_{Lin} = 2$, $\sigma_{SE} = 0.1$, $l_{SE} = 0.6$	3.54×10^{-5}
SE (long) + SE (short)	$\sigma_{long} = 0.1$, $l_{long} = 0.6$, $\sigma_{short} = 0.1$ $l_{short} = 0.1$	3.52×10^{-5}

TABLE 3.4: Summary of modified Gaussian process regression results for wines in category 1

Kernel	Hyperparameters	MSE
SE	$\sigma = 0.2$, l $= 1$	3.42×10^{-5}
RQ	$\sigma = 0.1$, l $= 0.2$, $\alpha = 3$	3.41×10^{-5}
Matern	$\sigma = 0.1$, l $= 0.5$	3.41×10^{-5}
SE + Lin $\times$ WN	$\sigma_{WN} = 0.3$, $l_{Lin} = 1$, $\sigma_{SE} = 0.1$, $l_{SE} = 0.4$	3.40×10^{-5}
SE (long) + SE (short)	$\sigma_{long} = 0.1$, $l_{long} = 0.6$, $\sigma_{short} = 0.1$ $l_{short} = 0.4$	3.40×10^{-5}

TABLE 3.5: Overall summary of results for wines in category 1

Method	MSE	AIC
Predicting 0	3.65×10^{-5}	-473.44
ARMA(1,1)	3.92×10^{-5}	-315.3062
GARCH(1,1)	7.95×10^{-5}	-374.1902
Gaussian process regression (optimal)	3.40×10^{-5}	-457.44

FIGURE 3.13: Box plots of MSE across all wines in category 1 for all techniques

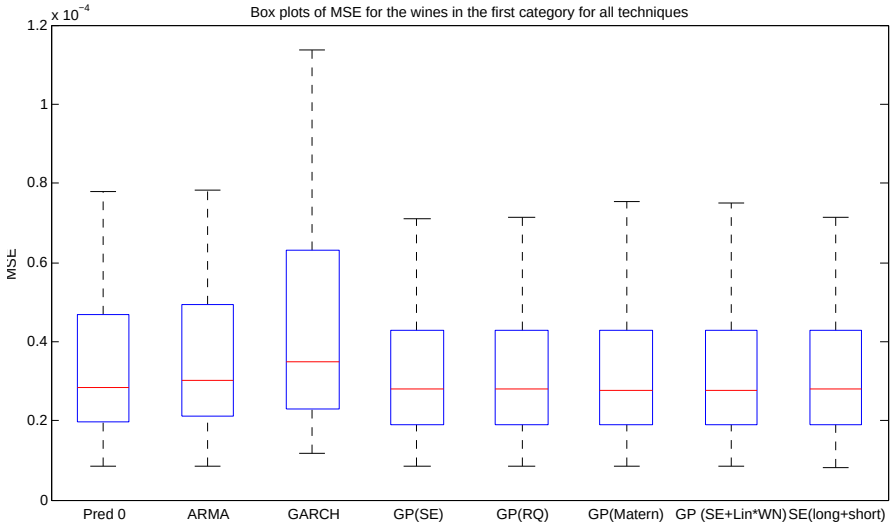


FIGURE 3.14: Comparing MSE across all wines in category 1 for the modified Gaussian process regression techniques

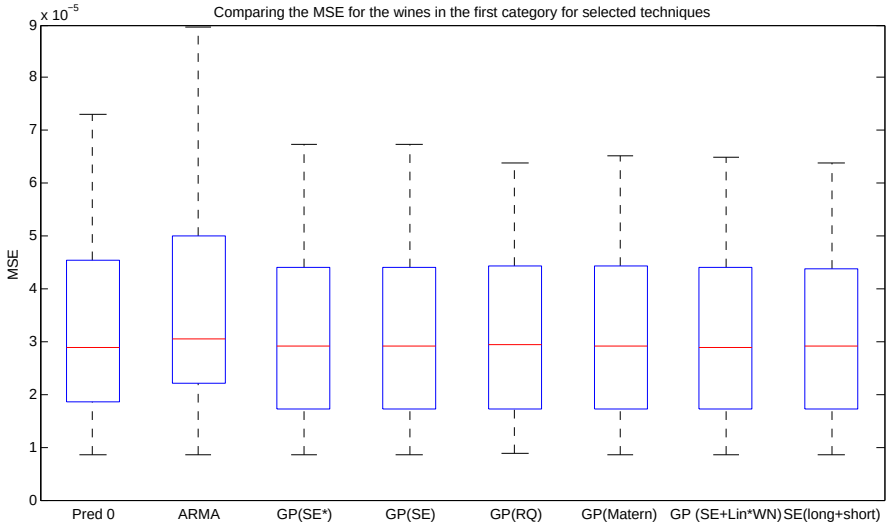


FIGURE 3.15: Bar chart of table 3.5

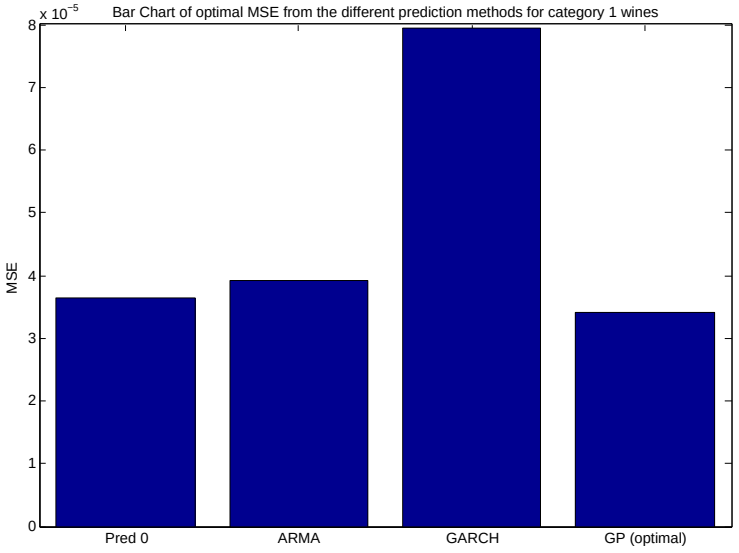


TABLE 3.6: Summary of Gaussian process regression results for wines in category 2

Kernel	hyperparameters	MSE
SE	$\sigma = 2, \mathrm{l} = 0.7$	6.94×10^{-5}
RQ	$\sigma = 2, \mathrm{l} = 0.9, \alpha = 4$	6.96×10^{-5}
Matern	$\sigma = 2, \mathrm{l} = 1$	6.99×10^{-5}
SE + Lin $\times$ WN	$\sigma_{WN} = 0.4, l_{Lin} = 2, \sigma_{SE} = 0.2, l_{SE} = 0.1$	7.20×10^{-5}
SE (long) + SE (short)	$\sigma_{long} = 0.1, l_{long} = 0.5, \sigma_{short} = 0.1\ l_{short} = 0.1$	7.55×10^{-5}

TABLE 3.7: Overall summary of results for wines in category 2

Method	MSE	AIC
Predicting 0	1.54×10^{-4}	-500.48
ARIMA(2,1,1)	8.21×10^{-5}	-412.34
GARCH(1,1)	4.5×10^{-3}	-411.10
Gaussian process regression (optimal)	6.94×10^{-5}	-492.48

FIGURE 3.16: Box plots of MSE across all wines in category 2 for all techniques (except GARCH)

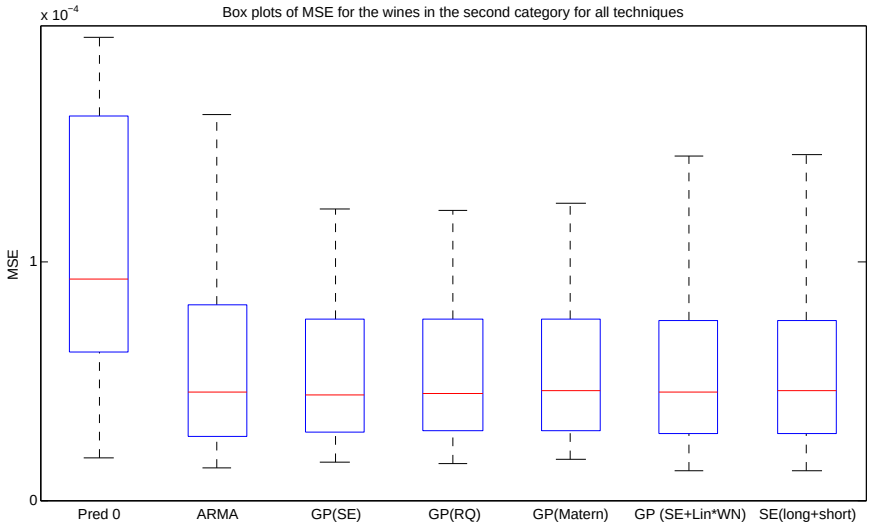
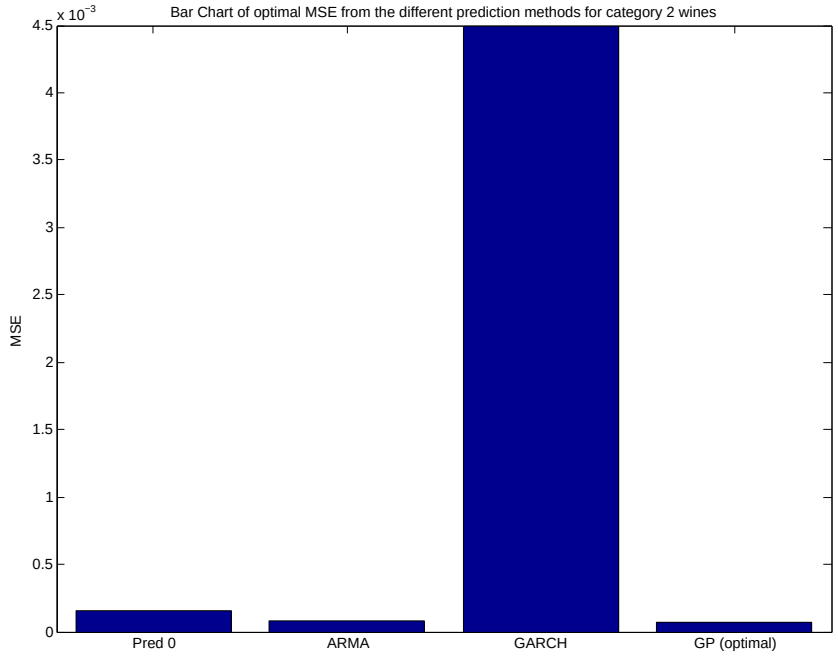


FIGURE 3.17: Bar chart of table 3.7



For the wines in the first category and using the earlier Gaussian process regression model, as evident from the table 3.3, the MSE values we get from using the various covariance kernels are very close and it is a toss up between the SE and RQ kernels for the lowest MSE. The difference between using the SE or RQ kernels versus using the other ones shown in table 3.3 though is very slight and the plots of predicted vs. actual returns look very similar. When we use the modified Gaussian process regression model, we also get pretty close MSE values for all the various kernel combinations we tried, but we get optimal MSE values from the more complicated kernel combinations, particularly the SE + Lin $\times$ WN and SE(long) + SE(short) combination as seen in table 3.4. The reason for this is perhaps due to the fact that for the modified Gaussian process regression model, we use the previous three returns values as input to the model instead of just the previous value as in the earlier model. This increase in the dimension of the input data favours the more complicated kernels as they are able to better relationships between the input data and use them to learn a better regression function.

Looking at the boxplots comparing the spread of MSE values across the different methods both both the earlier Gaussian process regression model and the modified Gaussian process in figures 3.13 and 3.14 respectively, we note that although it is hard to compare the average MSE for each method since the mean values all fall quite closely to one another, the Gaussian process models do seem to be more advantageous to the ARMA, GARCH, and trivial models in terms of the spread of MSE values across the wines in this category. For both earlier Gaussian process regression and modified Gaussian process regression models, we observe a persistently smaller whisker range for the Gaussian

process regression methods with all the various kernel combinations as compared to the ARMA, GARCH, and trivial models, with the GARCH model having the largest whisker range and the trivial model having the smallest out of these three. This suggests that we get more consistent results with Gaussian process regression, although the difference is not really that much.

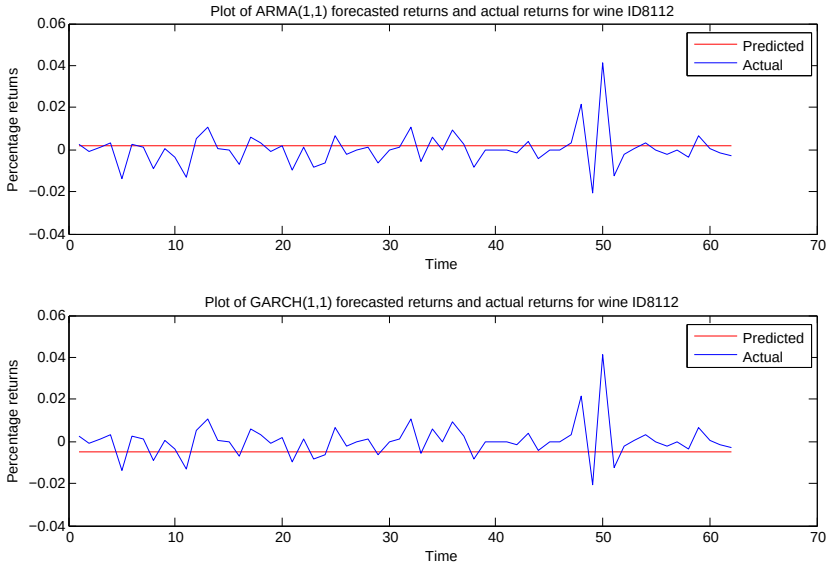
We chose the result we got from the modified regression model with the SE + Lin $\times$ WN kernel for our optimal Gaussian process regression model due to the fact that we get the smallest MSE value using this method when comparing to the other Gaussian process regression methods. Comparing the MSE scores of the different methods in table 3.5, we note that using this optimal Gaussian process regression model, we get a lower MSE value compared to trivially predicting 0 as well as significantly lower values compared to predictions made with the ARMA and GARCH models in chapter 2. This lowered MSE advantage is also evident in the bar chart representation of the optimal MSEs from the different techniques in figure 3.15 above. From the bar chart, we see that again the optimal MSE obtained from Gaussian process regression is clearly lower than that of ARMA and GARCH forecasting and is slightly but observably lower than that of predicting 0. We also like to compare the complexity of the models, just to see if the small gain in MSE we made with Gaussian process regression is not completely overshadowed by the complexity of the model. To do so, we look at the Akaike information criterion (AIC) of the different models, which measures the goodness of fit of a model whilst penalising for excessively complex models (models with too many parameters). The AIC is defined as follows:

$$AIC \quad = \quad 2k - 2l(\hat{\theta})$$

where k refers to the number of model parameters and $l(\hat{\theta})$ is the maximised value of the data log likelihood with respect to the vector of parameters θ. Lower AIC values imply a more parsimonious model which better models the data. From table 3.5, we note that the AIC value for the optimal Gaussian process regression model (using the SE kernel) is lower than that of the ARMA(1,1) and GARCH(1,1) models which, together with the lower MSE values, again suggests that Gaussian process regression is superior to ARMA and GARCH forecasting for the wines in this category. We note also for Gaussian process regression, when we compare the plots of predicted vs. actual returns to those from ARMA and GARCH forecasting, we find that we seem to learn a more realistic function which better captures how the data varies. Comparing the plot of predicted vs actual returns of wine ID 8112 using Gaussian process regression with the SE kernel in figure 3.5 above to the plots of predicted vs actual returns of the same wine using ARMA and GARCH regression in figure 3.18 below, we see that using Gaussian

process regression, we manage to learn a function which incorporates the variation of the data, and can even capture a small peak and dip at around test time points 48-50 where we see a huge spike and dip in the returns data, and is not just predicting a straight line as in the case of ARMA and GARCH forecasting.

FIGURE 3.18: Predicted vs actual returns for ARMA and GARCH forecasting of wine ID 8112



As for comparing Gaussian process regression to the trivially predicting 0, we note that not only is the MSE result for our optimal Gaussian process regression not that much lower than predicting 0 all the time, but also the AIC of predicting 0 all the time is lower than that Gaussian process regression, simply because of the larger number of model parameters we have for our optimal Gaussian process regression model with the SE kernel. The fact that the MSE is not much lower for the optimal Gaussian process regression is not surprising, given that we constantly predict near 0 all the time and we still have trouble capturing the peaks and dips in the returns data as seen in figure 3.5. Nevertheless, as mentioned above, the predicted vs actual returns plots we get from Gaussian process regression are much better compared to the plots of predicted vs. actual returns for the ARMA and GARCH forecasting.

From table 3.6, the optimal kernel for Gaussian process regression on the wines from the second category seems to be the SE kernel, although the MSE values of the SE, RQ, and Matern kernels are all very close. Again we choose the SE kernel as the optimal Gaussian process regression kernel to compare with the other forecasting methods in table 3.7 due to the fact that choosing the SE kernel for our Gaussian process regression model implies choosing a simpler model. From table 3.7, we note that the optimal Gaussian process

model clearly outperforms the trivial model which predicts 0 all the time in terms of MSE, as the optimal MSE value we get from predicting 0 all the time is almost an order of magnitude larger than that we get from our optimal Gaussian process model. As the AIC of the optimal Gaussian process model is not that much larger than that of the trivial model, as evident in table 3.7, it is quite obvious that our Gaussian process model is clearly superior to the trivial model.

As for comparing our optimal Gaussian process model with results of the ARIMA and GARCH models obtained in chapter 2, again we note that our Gaussian process model clearly surpasses the GARCH model due to the fact that the GARCH model gives an MSE value of almost 2 orders of magnitude larger than that obtained from our optimal Gaussian process regression and also corresponds to a larger AIC (we even had to omit the boxplot of the MSE values across all the wines from the second category in figure 3.16 due to the fact that it would drastically alter the scale and make it impossible to compare the performance of the other techniques). In the case of the ARIMA model, we note that though the difference between the MSEs of the optimal ARIMA(2,1,1) model and the optimal Gaussian process regression model is not as large as that between the GARCH and the Gaussian process model, it is still fairly significant (8.21×10^{-5} vs. 6.94×10^{-5}). In addition, the Gaussian process model seems to fit the data better, as seen in the lower AIC of the model. We also go back to the plots of predicted vs. actual values of wine ID 346 we displayed in figure 2.27 in chapter 2 and we note that unlike the ARIMA(2,1,1) forecasting which simply predicts a linear function, our optimal Gaussian process prediction of the returns of wine ID 346 as seen in figure 3.12 actually appears to follow pretty closely to the spikes and dips in the actual returns data. In this sense, we see that the function learned from our optimal Gaussian process model for the wines in this category seems to capture more information and variation in the data and thus we can reasonably argue that this model surpasses the optimal ARIMA(2,1,1) model.

Finally, we compare the results we get from implementing Gaussian process regression on the wines from both categories. From tables 3.4 and 3.6, we note that in general, our algorithms perform better on the wines in category 1 and it certainly seems like the returns of wines from category 1 are easier to predict in general. This is hardly surprising, as we have noted several times in chapter 2 that the wines in category 2 have a more complicated structure and are less similar to one another in general than those in category 1 (we direct the reader to figure 2.7 once again for a visual refresher on the similarities of the wines). Our results from the ARMA and GARCH fits and forecasting in chapter 2 also point towards the same phenomenon. Nevertheless, we note that interestingly, though Gaussian process regression does give us a better MSE result compared to other techniques like ARMA, GARCH, and even the trivial model of just predicting 0 all the time for the wines in category 1, the improvement in MSE

is quite minimal. For the wines in the second category, however, the improvement in MSE of Gaussian process regression over the ARIMA, GARCH, and the trivial model is quite stark. Altogether, this suggests that Gaussian process regression models might perhaps be more suited for the wines in category 2 as it gives us a marked improvement in the overall average MSE for the wines in this category. This could be due to the more complicated autocorrelation structure the wines in this category possess which makes it difficult to capture in the simpler ARIMA and GARCH models, since they often make stronger assumptions on the nature of the data (eg. Gaussian innovations etc.). Gaussian process regression, however, being essentially a non-parametric, data-driven approach, makes fewer and weaker assumptions on the nature of the data and thus is able to draw more information from the data and thus learn a function which better fits the data.

Before concluding this chapter, it is noteworthy that all we have done so far is train Gaussian process models for each wine in each category independently of the other wines in the category. Our results so far have been decent, but we also note that the sequence of consistently frequent wine prices only go back to September 2013, which does not give us a lot of returns data for each wine with work with. This could perhaps result in problems like overfitting, especially when we use more complicated kernel options like the SE + Lin $\times$ WN kernel. In the next chapter, we use multi-task feature learning to see if we can gain a further performance advantage by "pooling" together data across the wines in each category and learning both features common to the wines in each category as well as using these common features to improve the learning of wine-specific functions for each wine.

Chapter 4

Multi-task learning

4.1 Task relatedness and feature learning

As mentioned at the close of chapter 3, a problem we could potentially face with using a larger number of previous return values as inputs for Gaussian process regression is that we could overfit our training data due to the reduced number of training samples we have to work with as well as the increase in the dimension of the features. In general, we note that the lack of sufficient training data is a persistent issue with the wine data as the total length of the wine price time series we are working with is only about 156 lags long. This chapter explores the use of multi-task learning which aims to provide a solution to the problem of learning a sequence of related tasks (i.e. a learning problem) simultaneously but in a data-poor environment, i.e. with only few data points for each task.

The basic idea behind multi-task learning is to make use of some measure of task-relatedness and use this relationship between tasks as additional information so as to improve learning for each task in the data-poor environment. We will focus on a feature-centric notion of task-relatedness as developed by Argyriou, Evgeniou, and Pontil in *Multi-task Feature Learning* [16] and *Convex Multi-task feature learning* [17], that is, a sequence of tasks can be related in that they all share a common, low-dimensional feature representation. Assuming we know this low-dimensional feature representation of the data, we can re-weight the weight vector in our regression for each task to place more weight on the features in this low-dimensional representation (i.e. the salient features) of the data. In addition, we note that because multi-task feature learning only capitalises on the related features among tasks to improve learning the individual task functions, we have to use multi-task feature learning in conjunction with other learning algorithms

(eg, ridge regression, SVM regression etc.) to learn the individual task functions. In our case, we will continue using Gaussian process regression to learn the wine returns.

Multi-task learning had been used most frequently in the area of collaborative filtering [16], [17], [18], where the goal is to learn user preferences for an item based on a subset of attributes or features. In this context, multi-task feature learning becomes useful in that some times the feature dimension is very large and certain features, like price of the item for instance, are arguably more predictive than others. Using a smaller subset of these more predictive features to learn the preferences for each user has been shown to be more accurate [17]. Nevertheless, there has been existing research which uses multi-task learning in slightly more uncommon situations like time series forecasting [19]. In the simplest example for this case, assuming that the value of the time series at a certain point in time is entirely dependent on previous values, like what is most commonly assumed with the ARMA and GARCH models used in most financial time series modelling, and assuming that we do not use a non-linear feature map on the input points, we get the feature space to be the set of previous values for each value of the series. Using multi-task feature learning, we can reduce the feature space to a subset of pertinent previous values which might give us better predictive results.

4.2 Learning algorithm

In the case of the wine data, we need to simultaneously learn both the low-dimensional feature representation of the wine returns as well as the weights for the Gaussian process regression for each wine. To do so, we adapt the multi-task feature learning algorithm developed by Argyriou, Evgeniou, and Pontil in [16] to suit our Gaussian process regression framework, although the main idea behind the algorithm remains very similar. The key idea behind the algorithm is to alternately perform a supervised step, which learns the regression functions for each wine independently, and an unsupervised step, which learns the shared, salient features in the form of a feature matrix. This process continues until a specified number of iterations is met, or until the feature matrix converges to a certain rank. The rank of the final feature matrix denotes the number of shared features among the tasks.

We will first introduce the basic framework of the alternating algorithm developed by Argyriou, Evgeniou, and Pontil in [16] and then go on to give a very brief explanation on why the algorithm works before going on to explain our adaptations to the algorithm. Before doing so though, we introduce some notation which will be heavily used throughout this section. Let T be the total number of tasks we wish to simultaneously learn, with each task t consisting of a sequence of m input/output pairs

$(x_{t1}, y_{t1}), ..., (x_{tm}, y_{tm}) \in \mathbb{R}^d \times \mathbb{R}$. Our goal is to learn T predictive functions, one for each task t.

We define $< ., . >$ as the standard inner product of two d-dimensional vectors for d $\in \mathbb{R}$. Symbolically, $< u, v >:= \sum_{i=1}^{d} u_i v_i$, for u, v $\in \mathbb{R}^d$. For a matrix D, we denote D^+ as its pseudoinverse. If D is a $d \times d$ matrix, we define $tr(D)$ as the trace of D, that is, $tr(D) := \sum_{i=1}^{d} D_{ii}$. Finally, L(.,.) denotes an arbitrary loss function (eg. ridge loss, hinge loss). The basic framework of the algorithm is thus as follows:

Data: wine features X and actual wine returns Y
Result: feature matrix D and weight matrix W = $[w_1, ..., w_T]$
initialise W;
while *convergence criterion not fulfilled* **do**
  $D = \frac{WW^T\frac{1}{2}}{tr(WW^T)^{\frac{1}{2}}}$;
  $\tilde{W} = argmin\ L(Y, < D^{+\frac{1}{2}}\tilde{W}, \phi(X) >) + \gamma\|\tilde{W}\|^2$;
  $W = \tilde{W}$;
end

Algorithm 1: Multi-task feature learning

Algorithm 1 starts off with a weight matrix W which can be easily initialised to the weight matrix learned by simple regression or any other method. Following that, for each iteration consists of first setting the feature matrix D to $\frac{WW^T\frac{1}{2}}{tr(WW^T)^{\frac{1}{2}}}$ (unsupervised step), re-learning the weights by incorporating D into the regression equation (supervised step), and finally updating the weight matrix W.

The proof that the optimal solution for learning the feature matrix D is to set $D = \frac{WW^T\frac{1}{2}}{tr(WW^T)^{\frac{1}{2}}}$ is quite lengthy and is the subject of [16] and [17] and so we omit reproducing it here. Briefly, an intuitive reason for why this is so is that after doing an eigendecomposition of D into $UDiag(\lambda)U^T$, U orthogonal, and minimising over first λ and then U, we end up with a solution which sets u_i as an eigenvector of WW^T. Given that this is so, we see that learning D involves learning the direction of greatest variances of WW^T and thus, if we take $\tilde{W} = D^{+\frac{1}{2}}W$, the update step for W, after learning W from minimising some loss function with an appropriate penalty, is simply equivalent to setting $W = D^{\frac{1}{2}}W$. Thus each update step for W involves multiplying it by the Cholesky factorisation of D, that is $D^{\frac{1}{2}}$, which then effectively performs feature selection on the wine features since it re-weights W such that more weight is given to the rows of W which, when multiplied with the wine features X, would capture more of the variation in the data.

Finally, we are aware that what we have presented so far is only a very brief intuition of the theory behind the algorithm. Further details about the algorithm can be found in *Multi-task Feature Learning* by Argyriou, Evgeniou, and Pontil [16].

The main adaptations we made to the algorithm are twofold. Firstly, we note that although Argyriou, Evgeniou, and Pontil did outline another algorithm in their other paper which shows how to adapt their existing algorithm to incorporate kernels in [17], they did not provide a software implementation of this algorithm. We thus further adapted our algorithm to incorporate the possibility of using kernels so we can not only use it on top of our existing Gaussian process framework, since we can now use the optimal Gaussian process covariance kernel we got from our Gaussian process regression in the previous chapter on the input points, but also allow our algorithm to serve as a useful reference for future work in this area.

The effect of using a kernel on the training and test data inputs is that the training and test data inputs are mapped to a higher dimensional feature space, which can be potentially infinite dimensional if we use a radial basis function kernel for instance. However, since the kernel function is an inner product of two feature vectors, that is, the feature map evaluated on an input vector, we can use the kernel function to create a kernel (Gram) matrix K on all the training and test input points with $K_{i,j}$ consisting of inner products between the ith and jth training or test input points respectively, that is, $K_{i,j} =< \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) >$ with $\mathbf{x}_i$ and $\mathbf{x}_j$ being some training or test input point and ϕ being a feature map. Using the submatrix of K which only consists of inner products between training data $=: \tilde{K}$, we can use the columns of $\tilde{K}$ as explicit representations of our training data. The result of this on our feature space is that unlike in the simple example detailed above concerning a time series without kernels, the dimension of our feature space will not be the number of previous values used to predict any given value, but rather the length of a column of $\tilde{K}$. Assuming T tasks and n training points for each task, we get a feature space of size $n \times T$.

Because $n \times T$ might be very large, especially if we have a large number of tasks, we present a way to reduce the dimensionality of the feature space by selecting only the left-singular vectors of $\tilde{K}$ which correspond to the largest singular values of $\tilde{K}$. We do this by first performing a singular value decomposition of $\tilde{K}$ and finding the total number of singular values greater than some ϵ threshold $=: dd$. To get the dd by $n \times T$ matrix whose columns are the explicit representations for each training input point, we simply take the square root of the singular values which are greater than the ϵ threshold and multiply them by the left singular vectors of $\tilde{K}$ which correspond to these singular values. We thus work with a feature space of dd instead of $n \times T$. To perhaps facilitate

an understanding of the method we describe above, we present a snippet of MATLAB code which basically implements our dimensionality reduction method below:

```
[U,S,V] = svd(K);
SD = diag(S);
dd = sum(SD > epsilon);
R = diag(sqrt(SD(1:dd)))*U(:,1:dd)';
```

As for our second adaptation to the algorithm, we note that the existing software implementation of Algorithm 1 written by Argyriou, Evgeniou, and Pontil calculates the root of the feature matrix D by first inverting it and then taking the square root. Our software implementation avoids this matrix inversion altogether, making our algorithm faster and more stable. Instead, to get the root of the feature matrix D we simply perform a singular value decomposition of W^T, multiply the left singular vectors with the square root of the singular values, and divide that by the square root of the Frobenius norm. Again we provide a snippet of our MATLAB code which highlights this adaptation below:

```
[U, S, ~] = svd(W');
embedS = [sqrt(diag(S)); zeros(size(U,1) - size(diag(S,1),1) -1,1)];
temp = U*diag(embedS)*U';
Dsqrt = temp/sqrt((norm(W','fro')));
```

4.3 Demonstration

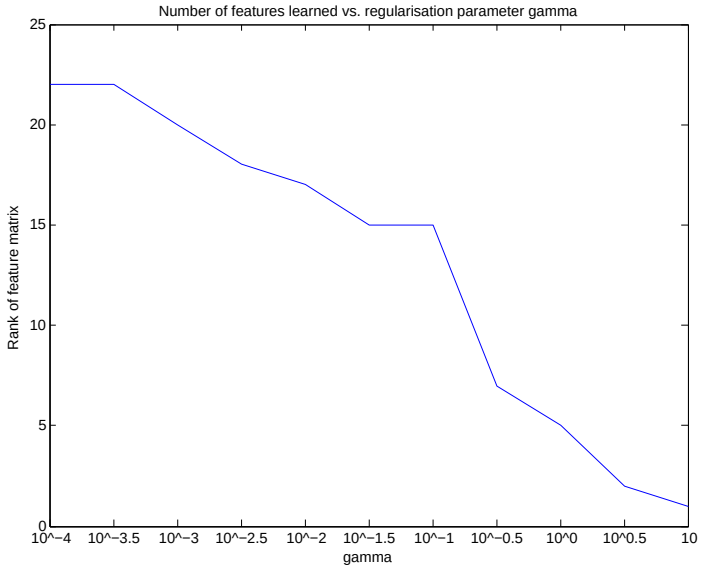
4.3.1 Toy data

We simulated 2 different toy datasets to test if firstly, our algorithm is working correctly, and secondly, if our algorithm matches up to that of Argyriou, Evgeniou, and Pontil. The first is a simple regression dataset taken with a d = 15 dimensional weight vector for each of the T = 200 tasks. We first chose 5 relevant dimensions from a zero-mean Gaussian distribution with covariance = $diag(1, 0.25, 0.1, 0.05, 0.01)$ and then added 10 irrelevant dimensions of exactly 0 to each weight vector. Input data $x_{t,m}$ for m = 10 was generated from arbitrarily and uniformly from $(0, 1)^d$ and the outputs $y_{t,m}$ was generated by multiplying the weight vector by the input data and adding a zero mean unit variance white noise term. This synthetic dataset is very similar to the one generated by Argyriou, Evgeniou, and Pontil in [16] and the purpose of running our algorithm on this dataset

is to test if we get similar results to that achieved by Argyriou, Evgeniou, and Pontil's algorithm.

We ran the algorithm on this dataset and found that, like Argyriou, Evgeniou, and Pontil, the number of features learned by our algorithm as measured by $(rankD, tol)$ for some tolerance level tol is sensitive to the value of the regularisation term γ as can be seen from figure 4.1 below. Since γ penalises large values in the weight vector learned for each individual task regression, we expect larger γ values to shrink more of the coefficients in the weight vector to 0 and thus result in less features learned. As expected, our algorithm picks out less features as γ increases for a fixed tolerance level tol, which is similar to that achieved by Argyriou, Evgeniou, and Pontil's algorithm.

FIGURE 4.1: Number of features learned vs. regularisation parameter γ



The second toy dataset tests the performance of our algorithm on synthetic time series data to see how the algorithm performs on data which is similar to the wine data. We simulated synthetic time series data using a 10 arbitrary, latent directions in a 40 dimensional feature space. We first fixed the number of tasks T to be 50 and then randomly chose 10 directions within the d = 40 dimensional feature space. Multiplying this projection with a T by 10 random matrix, we get a T by d weight matrix, with each row of the matrix representing a weight vector for each time series. To simulate the time series for each task, we first chose a random d-dimensional vector as the first input vector $x_{t,1}$ and multiplied it by the weight vector from the aforementioned weight matrix, adding on a zero mean unit variance white noise term to get the output term y_{t1}. We extended this process to create $m = 200$ input output observations $\{\mathbf{x}_{t,m}, y_{t,m}\} \in \mathbb{R}^{40} \times \mathbb{R}$ for each of the 50 tasks, with each input created by taking the last d-1 values in the previous input

vector and concatenating it with the previous output, that is (in MATLAB notation), $x_{t,m+1} = [x_{t,m}(2 : end), y_{t,m}]$.

We then ran our algorithm above to check the rank of our feature matrix after a number of iterations. Ideally we would like to be able to retrieve a rank 10 feature matrix since the toy data was created using this latent, 10 dimensional vector. Nevertheless, as highlighted above, the effective rank of the feature matrix we get is very sensitive to the regularisation parameter γ and thus just by varying γ, we get different ranks for our learned feature matrix. Again we realised that as γ increases, we we learn a feature matrix of smaller rank and vice versa.

We also ran the implementation written by Argyriou, Evgeniou, and Pontil but found that unlike in the case of our algorithm, the learned feature matrix when using their algorithm is only either a full rank feature matrix or a rank 1 feature matrix, even when tweaking with the values of the regularisation parameter and the tolerance *tol*. This perhaps suggests that our algorithm might perform better for time series data, which we seek to ascertain in the next section with our actual wine data.

4.3.2 Actual wine returns

As in the earlier chapters, we look at learning the wine returns separately for the wines from each category. For each category we run three algorithms to see which performs best: Argyriou, Evgeniou, and Pontil's software implementation of their multi-task feature learning algorithm, our implementation without using the kernel, and our implementation with the kernel. Unlike before where we used only either one or three previous return values to predict a given return value, we will be using the previous 20 return values to predict a given return value in the wine time series for the wines in the first category.

For the wines in the first category, we first ran the algorithm from Argyriou, Evgeniou, and Pontil on the wines from category 1. Using 3 fold cross validation for $\gamma = 0.0001$ to 10, we found the optimal average MSE for the wines in this category to be 3.59×10^{-5} with $\gamma = 0.1$. With our algorithm without the kernel and again using 3 fold cross validation for $\gamma = 0.0001$ to 10, we unfortunately achieved a higher optimal average MSE of 1.34×10^{-4} with an optimal γ of 0.01.

Finally, we ran our algorithm with the optimal kernel we got from our Gaussian process regression in chapter 3. Since we got similar MSE results for both the SE + Lin $\times$ WN kernel as well as the SE(long) + SE(short) kernel as seen in table 3.4, we chose to just use the SE(long) + SE(short) kernel since its simpler to implement. Using this kernel

and adopting the notation we used in the previous section, we applied the dimensionality reduction technique mentioned in the previous section to get a reduced feature space dimension of $dd = 201$. Again using 3-fold cross validation with the same γ range, we get a higher average MSE of 5.66×10^{-4} this time with an optimal γ parameter of 0.1. This MSE result we get is much higher than what we got with our optimal Gaussian process regression in the previous chapter as well as what we got for our ARMA prediction in chapter 2.

We plot the predicted vs. actual returns of wine ID 8112 after running the algorithm written by Argyriou, Evgeniou, and Pontil, our algorithm without the kernel, and our algorithm with the kernel adaptation below in figures 4.2, 4.3, and 4.4 just to see why the predictive performance of our algorithm is not as good as expected.

FIGURE 4.2: Predicted vs. Actual wine returns of wine ID 8112

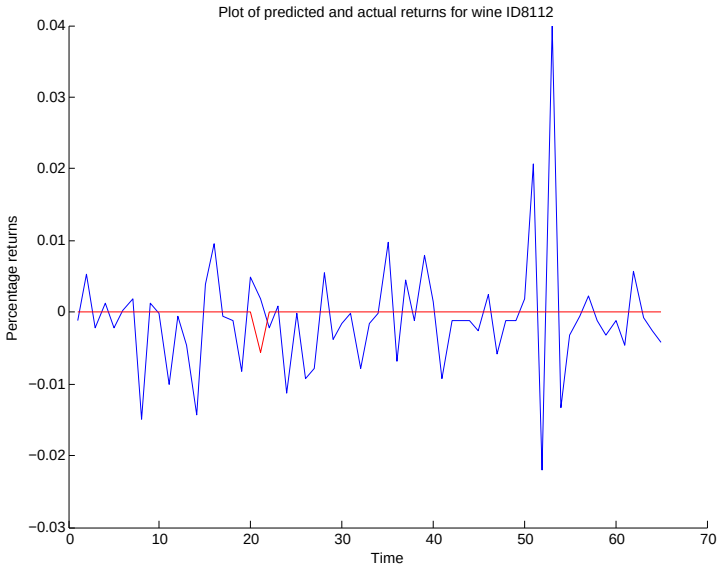


FIGURE 4.3: Predicted vs. Actual wine returns of wine ID 8112

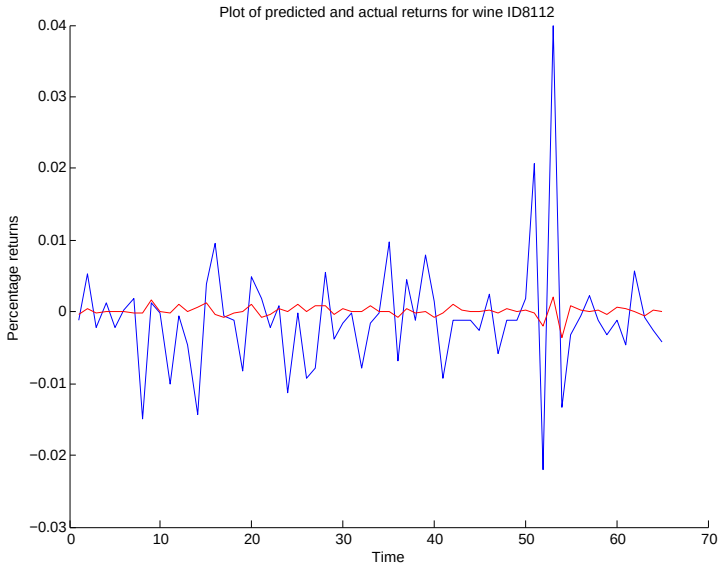
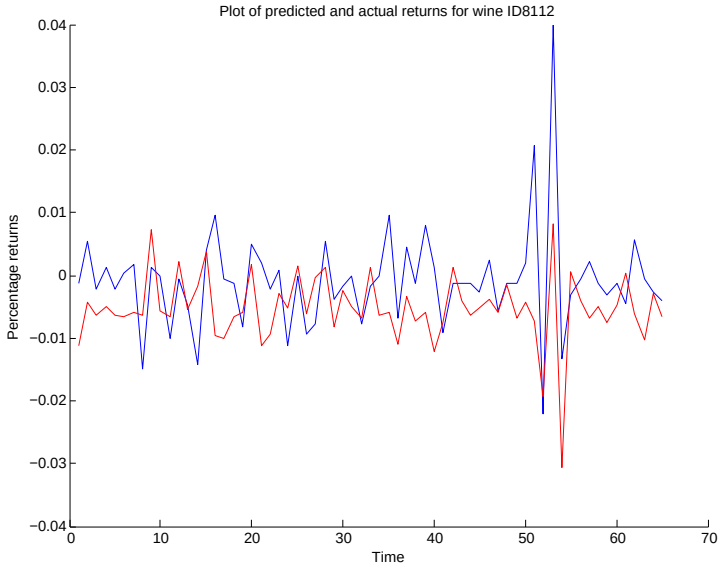


FIGURE 4.4: Predicted vs. Actual wine returns of wine ID 8112



From figure 4.4 above, we note that though we got the worst MSE result for the prediction of our algorithm with the kernel, the plot of predicted returns we get by running our algorithm with the kernel certainly appears to the best out of the three as can be seen in figure 4.4. Our predicted returns for wineID 8112 follow the actual returns most closely, with all the spikes and dips in the right places. In fact, we manage to capture the huge spike and dip at around the 50^{th} to 55^{th} test point in our prediction, something which our previous algorithms all had difficulty doing. We note though that a plausible

reason for the higher MSE value we get from running our algorithm with the kernel could be due to the fact that though we seem to be correctly predicting the direction of change most of the time, we have a difficulty in predicting the magnitude of the change, resulting in our predicted spikes and dips sometimes being much higher or lower than the spikes and dips in the actual series.

Comparing figures 4.2 and 4.3, we note again that the plot of predicted vs. actual returns for our algorithm without the kernel seems to follow the actual data more closely compared to that of Argyriou, Evgeniou, and Pontil, as it does seem wigglier in the right places and does capture a bit of that huge spike at around the 50^{th} to 55^{th} test point. Nevertheless, we note that though the plot of predicted vs. actual returns of wine ID 8112 we get from our algorithm appears better than that of Argyriou, Evgeniou, and Pontil's, we believe that the plot of predicted vs. actual returns of wine ID 8112 we get from our algorithm with the kernel still looks the best out of the three as our predictions seem to best follow the actual returns series.

We devised two different tests of predictive ability since we feel that though our algorithm underperforms on the wine data in terms of MSE when compared to the other techniques we have tried, our results seem to conform best to the data just from the plots alone. The first test checks if we can perhaps better predict positive or negative wine returns with our algorithm, as measured by the accuracy of our algorithm in predicting the sign of the actual returns series with our predicted series with a 0-1 loss function, instead of focusing on predicting the actual return values. For Argyriou, Evgeniou, and Pontil's algorithm, we get an accuracy rate of 0.49. With our algorithm without the kernel, we get a slightly higher accuracy rate of 0.50. With our algorithm with the kernel, we got a much higher accuracy rate of 0.59. Finally, we test this against the trivial prediction benchmark of predicting 0 all the time and we get an accuracy rate of 0.47 for this method.

We next test if we can guess the direction of the change in the actual returns series as measured by predicting the sign of the first difference of the actual returns series with a 0-1 loss function. This corresponds to predicting the sign of the gradient of the actual series. We will take predicting a change of 0 as predicting a negative change. For Argyriou, Evgeniou, and Pontil's algorithm, we get an accuracy rate of 0.49. With our algorithm without the kernel, we get a much higher accuracy rate of 0.73. With our algorithm with the kernel, we got a slightly higher accuracy rate of 0.75. Finally, we test this against the trivial prediction benchmark of predicting 0 all the time and we get an accuracy rate of 0.48 for this method.

Finally, we also take a look at the rank of the final feature matrix we get after running all three algorithms, just to see if the results make sense. For the first two algorithms,

fixing the tolerance level tol at 0.01, we get a feature matrix of rank 1 after 50 iterations as measured by $rank(D, tol)$. This is unsurprising, as without using a kernel on the training and test inputs, the feature space for these wines are simply their previous return values. Based on our analysis in chapter 2, we note that the wines in category 1 are characterised by a strong lag 1 autocorrelation structure, which explains the fact that the feature matrix converges to a rank 1 matrix, since most of the variance in the wine returns series appears to be captured by the previous wine return value. For our algorithm with the kernel, we get a feature matrix of rank 21 after 50 iterations with the same tolerance level which is quite a large reduction from the $dd = 201$ space we were working in.

Repeating the same procedure for the wines in the second category using the first difference of their returns instead of the actual returns like before, we first ran Argyriou, Evgeniou, and Pontil's software implementation of Algorithm 1. With 3-fold cross validation and $\gamma = 0.0001$ to 10, we get the optimal γ parameter to be 10 and a corresponding average MSE of 6.99×10^{-5}. With our algorithm without the kernel and again using 3 fold cross validation for $\gamma = 0.0001$ to 10, we achieved a higher average MSE of 2.10×10^{-4} with an optimal γ of again 10.

Finally, we ran our algorithm with the optimal kernel we got from our Gaussian process regression in chapter 3. Since we got the best results in terms of MSE when we used the SE kernel as seen in table 3.6, we will use the SE kernel with $\sigma = 2$ and $l = 0.7$ on the training and test input data for our algorithm. Using this kernel and adopting the notation we used in the previous section, we applied the dimensionality reduction technique mentioned in the previous section to get a reduced feature space dimension of $dd = 154$. Again using 3-fold cross validation with the same γ range, we unfortunately get a larger average MSE of 6.33×10^{-4} as compared to our algorithm without using the kernel and that of Argyriou, Evgeniou, and Pontil with an optimal γ parameter of 10.

To get a better sense of how the predictions of each algorithm relate to the actual data, as well as why our algorithm does worse than that of Argyriou, Evgeniou, and Pontil's for the wines in this category, we plot the predicted vs. actual returns of wine ID 346 from running the algorithm written by Argyriou, Evgeniou, and Pontil, our algorithm without the kernel, and our algorithm with the kernel adaptation in figures 4.5, 4.6, and 4.7 below:

FIGURE 4.5: Predicted vs. Actual wine returns of wine ID 346

FIGURE 4.6: Predicted vs. Actual wine returns of wine ID 346

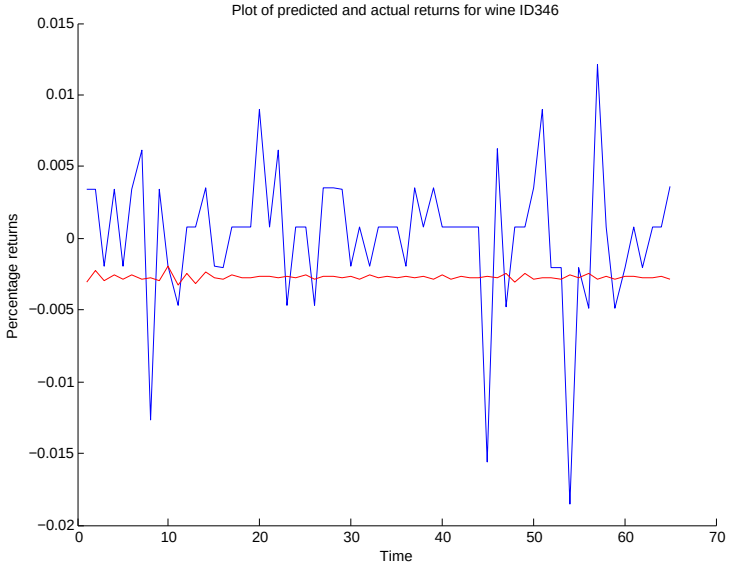
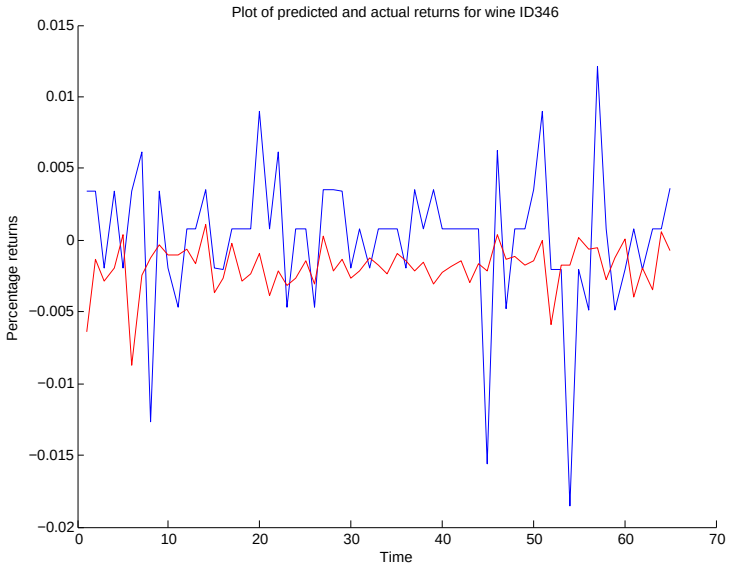


FIGURE 4.7: Predicted vs. Actual wine returns of wine ID 346



Comparing the plots of predicted vs. actual returns of wine ID 346 we get from running the three algorithms, we see that the prediction we get from Argyriou, Evgeniou, and Pontil's algorithm is very close to 0 and is also a straight line prediction as evident in figure 4.5. In addition, it is also quite obvious from the plots why the MSE values from the predictions we get from both our algorithms (with the kernel and without) are larger than that we get from Argyriou, Evgeniou, and Pontil's algorithm. In the case of our algorithm without the kernel, we note that our predicted returns appear to be translated and centred at a point below the mean of the returns series as can be seen from figure 4.6, even though we do get a wigglier line with spikes and dips which correspond quite well to the data, especially around the 10th to 12th test points.

For our algorithm with the kernel, we note that although we got the worst average MSE for our predictions, our predicted returns series seem to correspond the best to the actual returns data as seen in figure 4.7 above. Although we note that, like in the case of our algorithm without the kernel, our predictions seem to be translated and centred at a point somewhat below the mean of the actual returns series, our predictions, however, seem to capture most of the significant spikes and dips in the actual data. Of course there are also a couple of glaring misses and we note that this is not as close as what we got in figure 4.4 with our algorithm with the kernel but on the wines from the first category, but still our predictions with the kernel and our algorithm seems to fit the data better than that of Argyriou, Evgeniou, and Pontil's algorithm even though we get a better MSE result from running their algorithm.

Again because we believe that though we do not get the best MSE results with our algorithm with the kernel, we get plot which conform quite closely with the actual data, we run a similar test of predictive ability by testing to see how well our algorithm with the kernel performs when we focus solely on predicting the sign of the wine returns with a 0-1 loss function. For Argyriou, Evgeniou, and Pontil's algorithm, we get an accuracy rate of 0.64. With our algorithm without the kernel, we get a slightly higher accuracy rate of 0.70. With our algorithm with the kernel, we got a slightly lower accuracy rate of 0.69 as compared to our algorithm without the kernel, but still higher than that of Argyriou, Evgeniou, and Pontil's. Finally, we test this against the trivial prediction benchmark of predicting 0 all the time and we get an accuracy rate of 0.66 for this method.

For the second test of the accuracy of predicting the sign of the change in the actual returns, we get an accuracy rate of 0.6 for Argyriou, Evgeniou, and Pontil's algorithm. Without the kernel, we get a significantly higher accuracy rate of 0.86 for our algorithm. Finally, we get a lower accuracy rate of 0.69 for our algorithm with the kernel as compared to without the kernel, but still higher than Argyriou, Evgeniou, and Pontil's. With the trivial prediction, we get an accuracy of 0.69.

Like before, we also take a look at the rank of the final feature matrix we get after running all three algorithms. Again like before and using the same tolerance level of $tol = 0.01$, we get a feature matrix of rank 1 after 50 iterations of alternating the learning of the feature matrix and the weight matrix. This result is also very much expected since when alternately learned the weight and feature matrices we used the first difference of the wine returns series as input and we know from chapter 2 that the first difference of the returns of the wines in the second category have a strong negative lag 1 autocorrelation structure. For our algorithm with the kernel, we get a feature matrix of rank 25 after 50 iterations with the same tolerance level which, like in the case of the wines in the first category, is quite a large reduction from the $dd = 154$ space we were working in.

4.4 Summary and discussion of results

We summarise the results we obtained from using multi-task feature learning as well as an overall table of the best results we get from all the techniques we have tried so far as follows:

TABLE 4.1: Summary of multi-task learning prediction result for category 1 wines

Algorithm	Kernel	MSE
Argyriou, Evgeniou, and Pontil	none	3.50×10^{-5}
Ours	none	1.34×10^{-4}
Ours	SE (long) + SE (short)	5.66×10^{-4}

TABLE 4.2: Overall summary of optimal results for wines in category 1 for all methods in terms of predicting the actual wine returns

Method	MSE
Predicting 0	3.65×10^{-5}
ARMA(1,1)	3.92×10^{-5}
GARCH(1,1)	7.95×10^{-5}
Gaussian process regression (optimal)	3.40×10^{-5}
Multi-task feature learning (optimal)	3.50×10^{-4}

TABLE 4.3: Overall summary of optimal results for wines in category 1 for all methods in terms of predicting the sign of the actual wine returns

Method	Accuracy
Predicting 0	0.47
ARMA(1,1)	0.47
GARCH(1,1)	0.50
Gaussian process regression (optimal)	0.53
Multi-task feature learning (Argyriou, Evgeniou, and Pontil)	0.49
Multi-task feature learning (ours without kernel)	0.50
Multi-task feature learning (ours with kernel)	0.59

TABLE 4.4: Overall summary of optimal results for wines in category 1 for all methods in terms of predicting the sign of the change the actual wine returns

Method	Accuracy
Predicting 0	0.48
ARMA(1,1)	0.48
GARCH(1,1)	0.47
Gaussian process regression (optimal)	0.21
Multi-task feature learning (Argyriou, Evgeniou, and Pontil)	0.49
Multi-task feature learning (ours without kernel)	0.73
Multi-task feature learning (ours with kernel)	0.75

FIGURE 4.8: Bar chart of table 4.2

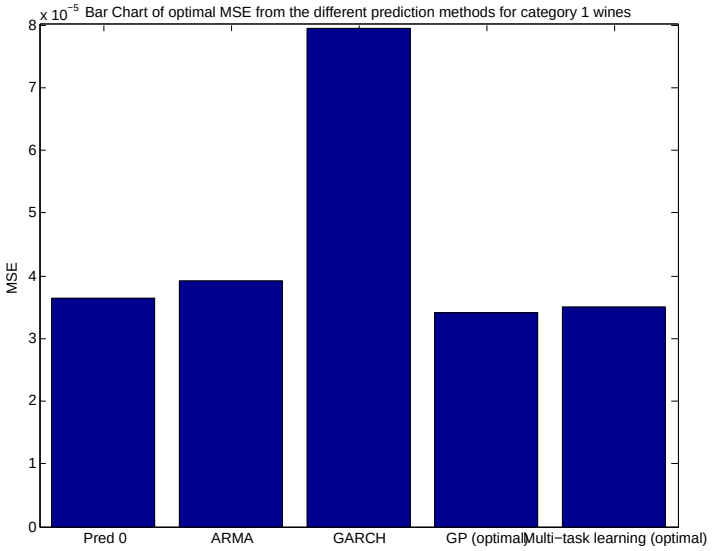


FIGURE 4.9: Bar chart of table 4.3

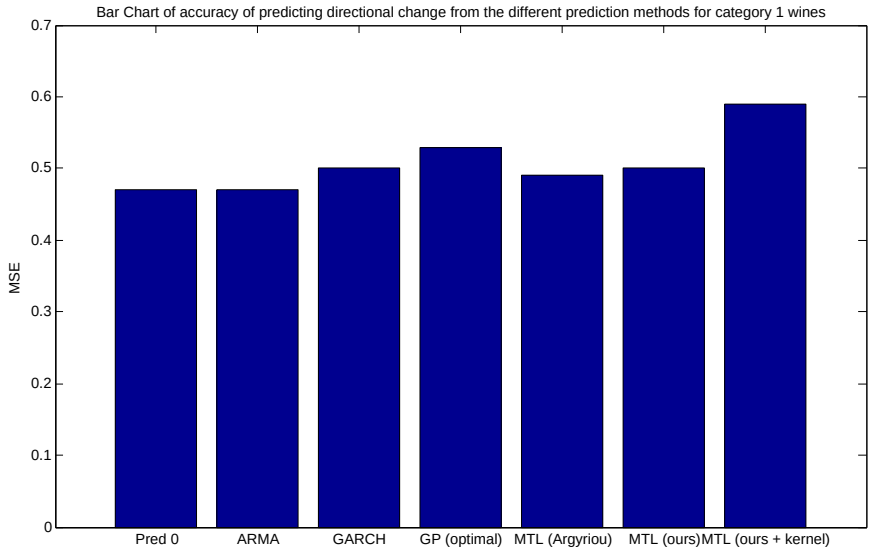


FIGURE 4.10: Bar chart of table 4.4

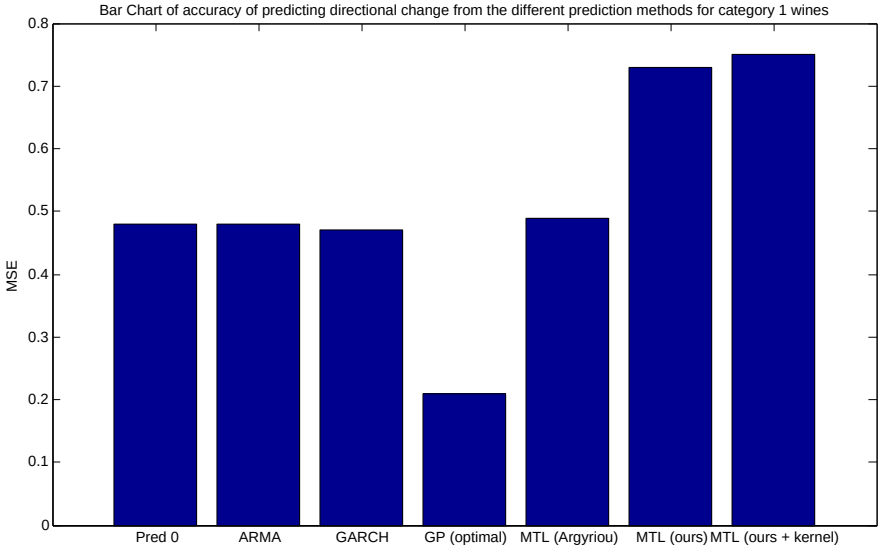


TABLE 4.5: Summary of multi-task learning prediction result for category 2 wines

Algorithm	Kernel	MSE
Argyriou, Evgeniou, and Pontil	none	6.99×10^{-5}
Ours	none	2.10×10^{-4}
Ours	SE	6.33×10^{-4}

TABLE 4.6: Overall summary of optimal results for wines in category 2 for all methods in terms of predicting the actual wine returns

Method	MSE
Predicting 0	1.54×10^{-4}
ARIMA(2,1,1)	8.21×10^{-5}
GARCH(1,1)	4.5×10^{-3}
Gaussian process regression (optimal)	6.94×10^{-5}
Multi-task feature learning (optimal)	6.99×10^{-5}

TABLE 4.7: Overall summary of optimal results for wines in category 2 for all methods in terms of predicting the sign of the actual wine returns

Method	Accuracy
Predicting 0	0.66
ARMA(1,1)	0.66
GARCH(1,1)	0.66
Gaussian process regression (optimal)	0.73
Multi-task feature learning (Argyriou, Evgeniou, and Pontil)	0.64
Multi-task feature learning (ours without kernel)	0.70
Multi-task feature learning (ours with kernel)	0.69

TABLE 4.8: Overall summary of optimal results for wines in category 2 for all methods in terms of predicting the sign of the change of the actual wine returns

Method	Accuracy
Predicting 0	0.64
ARMA(1,1)	0.43
GARCH(1,1)	0.67
Gaussian process regression (optimal)	0.62
Multi-task feature learning (Argyriou, Evgeniou, and Pontil)	0.60
Multi-task feature learning (ours without kernel)	0.86
Multi-task feature learning (ours with kernel)	0.69

FIGURE 4.11: Bar chart of table 4.6

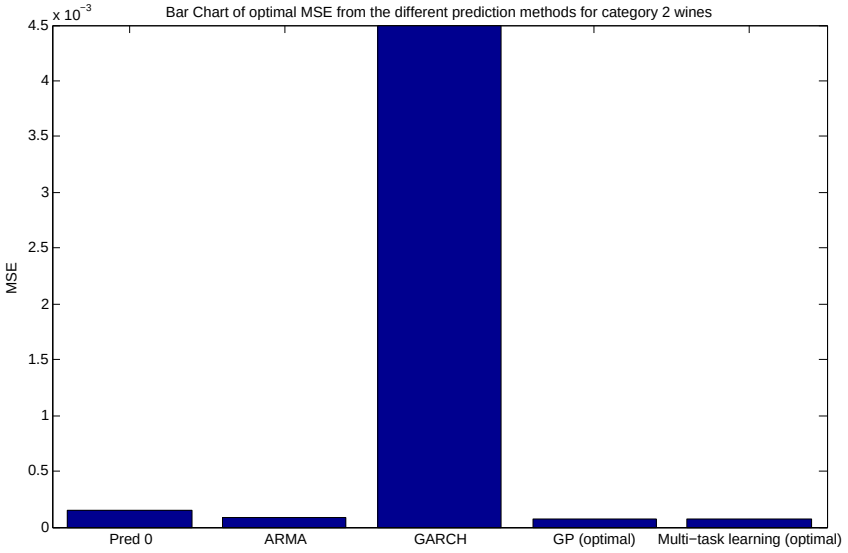


FIGURE 4.12: Bar chart of table 4.6 without GARCH MSE

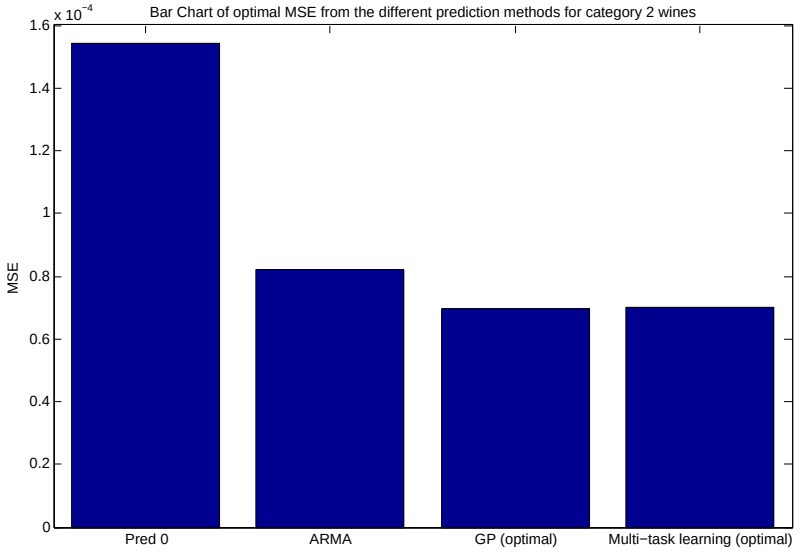


FIGURE 4.13: Bar chart of table 4.7

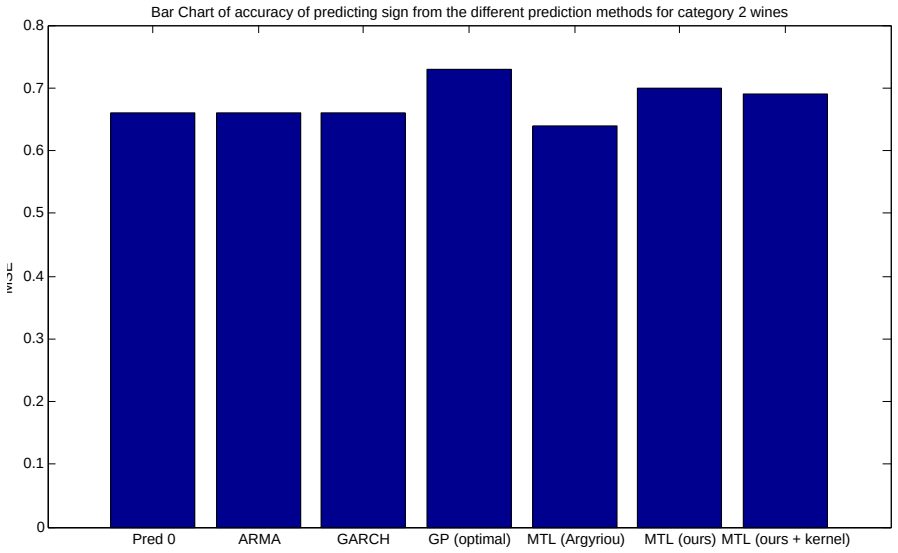
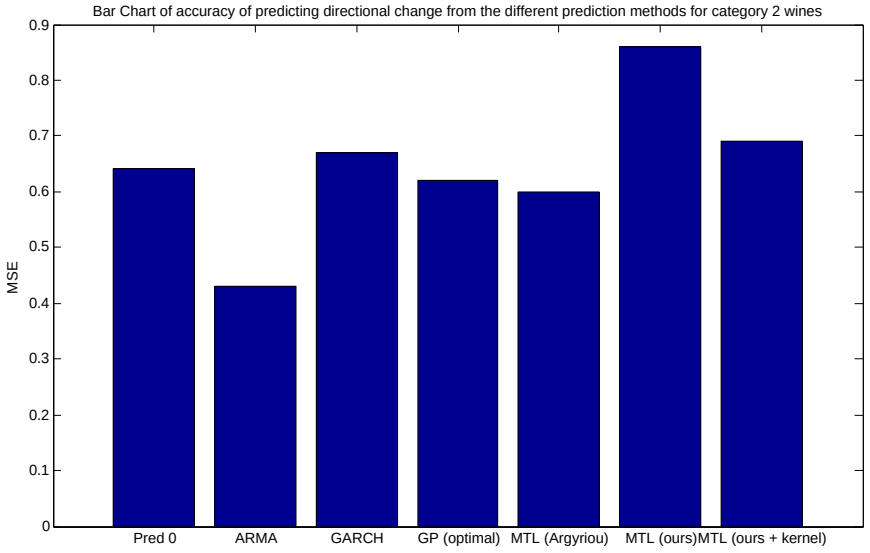


FIGURE 4.14: Bar chart of table 4.8



For the wines in category 1, if we just compare the performance of all three multi-task learning algorithms in terms of accuracy of predicting the value of the actual wine returns with MSE as the loss function, we note that Argyriou, Evgeniou, and Pontil's algorithm is does the best with the lowest MSE of 3.50×10^{-5} out of the three as seen in table 4.5. Nevertheless, this is still higher than the MSE value we get from our optimal Gaussian process regression as seen in table 4.2 as well as the bar chart illustrating this comparison in figure 4.8.

When we look at predicting either the sign or the directional change of the actual returns series, however, we note from tables 4.3 and 4.4 and the corresponding bar charts in figures 4.9 and 4.10 that we seem to get the best results from both our multi-task learning algorithms, with the multi-task learning algorithm with the kernel slightly outperforming that without the kernel for both tests. This is also illustrated when we compare the plots of predicted vs. actual returns for our multi-task learning algorithms with that we get from the other methods as the predicted returns series we get from our multi-task learning algorithms really do appear to accurately capture a lot of the spikes and dips in the actual data. This suggests that though our MSE results for our multi-task algorithms might not be as good as those we get from the other methods, including Gaussian process regression, we do seem to learn the signs of the actual returns as well as directional change better than the other methods. Also, this reinforces our earlier suspicion that perhaps the reason why we get higher MSE results when running our multi-task learning algorithms on the wines in this category is due to the fact that the magnitude of our prediction might be wrong, even though we seem to be predicting better in terms of the sign of the returns as well as the directional change.

A possible reason for this could be due to the fact that the wines in this category have a more straightforward negative lag 1 autocorrelation structure as elaborated in chapter 2. What this implies is that when we run the step in the algorithm which alternately learns the feature matrix and the weight matrix, the unsupervised step will tend to learn the straightforward negative lag 1 autocorrelation structure as it captures most of the variance in the training data, and re-weight the weight matrix so as to place more emphasis on this structure. This makes it easier for the algorithm to learn the changes in direction of the actual returns series as well as the sign of the actual returns since the final weight matrix has been re-weighted to place most emphasis on the input features which contribute to the prominent negative lag 1 autocorrelation structure characterising the wines in this category. This constant re-weighting of the weight matrix in the alternating part of the algorithm, however, is also a sort of double-edged sword as by reducing the dimensionality of the feature space to space of features which is most representative of the data in the unsupervised step, we run the risk of over-simplification and getting a representation which is too sparse and thus potentially sacrificing some predictive accuracy.

Altogether, our results for the wines in this category points to the fact that we can draw up two different research problems for the wines in category 1, each favouring a particular predictive algorithm. The first is more of a regression type of problem and seeks to learn a function which minimises the difference between the predicted and actual results measured in terms of MSE. In this case, we get the best results when we run our Gaussian process regression algorithm with the optimal kernel we found in the

previous chapter. The second research problem is classificatory in nature and seeks to learn a function which can most accurately classify the sign of the actual returns as well as accurately classify the direction of the change of the returns series from one point to another. Here, we get best results from using our multi-task learning algorithm. It is noteworthy that both research questions are equally important in their own light and that to compare both techniques to see which works best for the wines in this category seems a little unfair and contrived, as their advantage over the other techniques appears to be suited for two different research questions altogether.

As for the question of the extent to which our advanced machine learning techniques outperform the trivial and simple ARMA/GARCH models for the wines in this category, we have already shown in chapter 3 that our Gaussian process model outperforms the trivial model and the simple ARMA and GARCH models in terms of both MSE when we take into account AIC, as well as how well the predictions appear to fit the actual data, as measured by how good the plots of predicted vs. actual returns look. Nevertheless we do note that, as mentioned in chapter 3, the improvement of the results over the trivial prediction as well as the simpler ARMA and GARCH models is not that much, due to the simpler autocorrelation structure of these wines.

In the case of multi-task learning, we will omit taking AIC into consideration when comparing models as the usage of the kernel on the training inputs for our algorithm with the kernel significantly increases the number of parameters we will have in our model (nTrain $\times$ T), and that this number of parameters increases in proportion to the amount of training data we have. We will thus focus more on the fit to the data based on the plots as well as accuracy rate. Thus unlike the marginal improvement of Gaussian process regression for the wines in this category, we note that the improvement over the trivial model as well as the simpler ARMA and GARCH models when we look at the accuracy rate as shown in tables 4.3 and 4.4 as well as their corresponding bar charts in figures 4.9 and 4.10 is quite significant. Also as mentioned above, the improvement in terms of how our predicted series fit the actual series when we compare the plots is also quite stark.

For the wines in category 2, we also observe that Argyriou, Evgeniou, and Pontil's algorithm does the best in terms of predicting the value of the actual wine returns with MSE as the loss function. With their algorithm on the wine data, we get an optimal average MSE of 6.99×10^{-5}, which is lower than that we get from our algorithms as seen in table 4.5. Like in the first category, this optimal MSE value we get is still higher than the MSE value we get from our optimal Gaussian process regression as can be seen in table 4.6 as well as the bar chart illustrating this comparison in figure 4.11.

Looking at predicting either the sign or the directional change of the actual returns series, we note from tables 4.7 and 4.8 as well as their corresponding bar charts in figures 4.13 and 4.14 that although our multi-task learning algorithms do better than Argyriou, Evgeniou, and Pontil's as well as trivially predicting 0 and the simple ARMA/GARCH models, the difference in accuracy rate is not as stark as when we compare the accuracy rate of our algorithms to that we get from the other methods when we look at the wines in category 1. In addition, we note that our results from our optimal Gaussian process regression for the wines in this category actually surpasses that we got from our multi-task algorithm when we look at just predicting the sign of the actual returns. In terms of plots, while we certainly seem to get better plots for our multi-task algorithms, especially compared to that of trivially predicting 0, the simple ARMA/GARCH models as well as Argyriou, Evgeniou, and Pontil's algorithm, we note that the plots we get from our optimal Gaussian process regression as seen in figure 3.12 in chapter 3, are at least as comparable to that we get from our algorithms as seen in figures 4.6 and 4.7 above. Altogether, this suggests that, for the wines in this category at least, multi-task learning might not be that much more advantageous as compared to Gaussian process regression in terms of predicting either the sign or the directional change of the actual returns series.

A plausible reason for this could be due to the fact that the wines in the second category have a more complicated autocorrelation structure as compared to that of the wines in the first category. As elaborated in chapter 2, we note that though we managed to cluster the wines into two categories based on autocorrelation features, there is a lot more variance in the features of the wines belonging to the second category. This implies that learning a common feature representation for them in the unsupervised step of the alternating part of the multi-task feature learning algorithm would not only be tricky due to the complicated autocorrelation structure, but also due to the fact that the wines in this category are not as similar as those in the first, making it difficult to even learn the directional change of the actual returns series or the sign of the actual returns.

In addition, when we consider the extent to which our advanced machine learning techniques outperform the trivial and simple ARMA/GARCH models for the wines in this category, we note that, as elaborated in chapter 2, unlike the marginal improvement of the optimal Gaussian process regression on the prediction results for the wines in category 1, we achieve a significant improvement on the prediction results when we ran our optimal Gaussian process algorithm on the wines in this category. This, coupled with the fact that multi-task learning and Gaussian process regression seem to be comparable in terms of the accuracy of predicting the sign of the actual returns as well as the directional change in the actual returns series for the wines of this category, suggest that Gaussian process regression certain appears to work better and outperform the other

simpler prediction methods for the wines in this category. In contrast, we note that multi-task feature learning does not seem to add much value in terms compared to the simpler prediction models for the wines in the second category, especially in the light of our superior Gaussian process regression results.

Chapter 5

Conclusion

To summarise, our main results in this paper are that, firstly, the wines in the Liv-Ex 100 index can be clustered into two distinct categories based on autocorrelation structure, the first category consisting of wines with a simpler autocorrelation structure and the second consisting of wines with a more complex one. Secondly, we identified two main, distinct research questions: the first seeks to learn a function which minimises the difference between the predicted and actual results measured in terms of MSE, and the second seeks to learn a function which can most accurately classify the sign of the actual returns as well as accurately classify the direction of the change of the returns series from one point to another. Lastly, we showed that for the wines in the first category, the first research question favours Gaussian process regression, although the results we get are not much better than that of the simpler ARMA/GARCH prediction techniques, while the second research question favours our multi-task learning algorithm and we get results which are much better than those we get from the simpler ARMA/GARCH techniques. For the wines in the second category, both research questions favour Gaussian process regression, which gives us results that significantly surpasses those of the simpler models.

A natural consequence of our results is the question: where do we go from here? A well-known theory in finance is the random walk hypothesis, which argues that the prices of stocks evolve according to a random walk and as such cannot be predicted. Consistent with this theory, the best "prediction" we can get measured in terms of MSE is to just predict the empirical mean of the random walk all the time [20].

Nevertheless our research has shown that, for the 100 wines in the Liv-Ex 100 index at least, this random walk hypothesis does not quite hold. Our superior results with both Gaussian process regression and multi-task learning with regard to both research questions compared to the trivial model of just predicting 0 all the time highlights that there is some pattern we can learn in the wine returns series. In addition, we have

also shown that our machine learning techniques give us results with respect to both research questions which are superior to the simpler time series models. Perhaps no where is this more evident than in the plots of predicted returns vs. actual returns. Comparing the ones we get from running our optimal Gaussian process regression and multi-task learning algorithms on the dataset to the ones we get by predicting 0 or just using ARMA/GARCH, we note that our predicted processes manage to capture more of the spikes and dips of the corresponding actual processes compared to that of the simpler models, which suggests that our techniques seem to be capturing more information from the previous return values so as to generate predictions which follow the actual values more closely.

Of course we are aware that we have only been looking at a small subset of wines and that perhaps these 100 Liv-Ex 100 wines just happen to be more predictable than others. Nonetheless, as mentioned in the beginning of this paper, using machine learning techniques on the field of fine wine price prediction is completely unprecedented and our results point to a huge potential of conducting more of such machine learning research to the field.

In particular, we would like to point out two directions for future research which seem especially promising. Firstly, it would be interesting to pull in more wines from other regions and vintages to see if we can discern other interesting autocorrelation structures or if we can find a relationship between wines from certain regions or vintages and autocorrelation structure. Supposing there is some correlation, it would be interesting to see if we can perform a clustering of wines which lack enough previous price information into categories based on their region and vintage year. Secondly, it would be good to both test these machine learning techniques on more wines just to better ascertain the extent of their predictive advantage, as well as try out more machine learning techniques like deep Gaussian process belief networks for instance to see if we can achieve better predictive results. Altogether, we believe there is a lot of promise for more machine learning research in this very nascent field.

Bibliography

[1] Orley Ashenfelter. Predicting the quality and prices of Bordeaux wines. Technical report, Economics Department, Princeton University, New Jersey, 2007.

[2] Gregory V. Jones, Karl-Heinz Storchmann. Wine market prices and investment under uncertainty: an econometric model for bordeaux crus classes. *Oenometrie*, 9, 2001.

[3] Alessandro Corsi, Orley Ashenfelter. Predicting italian wine quality from weather data and experts ratings. *Agricultural Economics*, 26, 2001.

[4] Tony Lima. Price and quality in the california wine industry: An empirical investigation. Technical report, California State University, 1999.

[5] Orley Ashenfelter, David Ashmore, Robert LaLonde. Bordeaux wine vintage quality and the weather. *Chance*, 8, 1995.

[6] Bernardete Ribeiro and Noel Lopes. Deep belief networks for financial prediction. In *Neural Information Processing*, pages 766–773. 2011.

[7] Graham W. Taylor and Geoffrey E. Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *ICML*, page 129, 2009.

[8] Miguel Lazaro-Gredilla, Michalis K. Titsias. Variational heteroscedastic gaussian process regression. Technical report, London, UK.

[9] PhichHang Ou, Hengshan Wang. Modeling and forecasting stock market volatility by gaussian processes based on GARCH, EGARCH and GJR models. *World Congress on Engineering*, 1, 2011.

[10] Jonathan D. Cryer, Kung-Sik Chan. *Time Series Analysis with Applications in R.* Springer, 2008. ISBN 9780387759586.

[11] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning.* Massachusetts Institute of Technology Press, 2006. ISBN 026218253X.

[12] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006. ISBN 0387310738.

[13] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521813972.

[14] David Duvenaud. *Automatic Model Construction with Gaussian Processes*. PhD thesis, Computational and Biological Learning Laboratory, University of Cambridge, 2014.

[15] Documentation for gpml matlab code version 3.4. `http://www.gaussianprocess.org/gpml/code/matlab/doc/index.html`. Accessed: 2014-05-20.

[16] Andreas Argyriou, Theodoros Evgeniou, Massimiliano Pontil. Multi-Task Feature Learning. *Advances in Neural Information Processing Systems*, 19, 2006.

[17] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Mach. Learn.*, 73(3):243–272, December 2008. ISSN 0885-6125. doi: 10.1007/s10994-007-5040-8. URL `http://dx.doi.org/10.1007/s10994-007-5040-8`.

[18] Nguyen Duy Phuong and Tu Minh Phuong. Collaborative filtering by multi-task learning. In *Research, Innovation and Vision for the Future, 2008. RIVF 2008. IEEE International Conference on*, pages 227–232, July 2008. doi: 10.1109/RIVF.2008.4586360.

[19] Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *J. Mach. Learn. Res.*, 4:83–99, December 2003. ISSN 1532-4435. doi: 10.1162/153244304322765658. URL `http://dx.doi.org/10.1162/153244304322765658`.

[20] Gregory F. Lawler. Random Walk and the Heat Equation. `http://www.math.uchicago.edu/~lawler/reu.pdf`. Accessed: 2014-07-30.