4   NAÏVE BAYES, k-NN, and SUPPORT VECTOR MACHINES

In this chapter we shall study several algorithms that perform prediction of numerical targets as well as classifications. Naïve Bayes, k-NN (k-nearest neighbors), and Support Vector Machine (SVM) are supervised learning algorithms that are commonly used for classification of targets. They can also be used for regression in predicting continuous target values. For the case of SVM, regression is often called Support Vector Regression (SVR).

4.1 Naïve Bayes

The Naïve Bayes classifier (or classification algorithm) is based on Bayes' Rule or Bayes' Theorem. The theorem states that the conditional probability of event $\{Z=1\}$ given event Y has occurred is

$$P(\{Z=1\}|Y) = P(\{Z=1\} \cap Y) / P(Y)$$

where probability of event Y in the context of a universe set Ω in which Y resides is $P(Y)$, and $P(\{Z=1\} \cap Y)$ denotes the probability of event that $\{Z=1\}$ and Y occurred jointly. Examples of $P(Y)$ are the probability of random draw of a blue ball from a bag containing 30 blue balls, 30 red balls, and 40 green balls is 30/100 or 0.3, the random draw of a point falling in interval [0.3,0.5] in $\Omega = [0,1]$ is 0.2, and so on.

Now Y represents the event that a blue ball is drawn. Suppose in the bag with the same 100 balls, 10 of the 30 blue balls are painted with a black dot, 20 of the 30 red balls are painted with a black dot, and 30 of the 40 green balls are painted with a black dot. Let $\{Z=1\}$ represent the event that the ball drawn contains a black dot. $P(\{Z=1\}) = 0.6$ since altogether 60 balls have black dots. Suppose in a drawn ball, we are given the information that it is blue, then the probability that it also carries a black dot is $P(\{Z=1\}|Y) = 1/3$ or 10 out of 30 blue balls. $P(\{Z=1\}|Y)$ can also be computed as $P(\{Z=1\} \cap Y) / P(Y) = 0.1/0.3 = 1/3$ since $P(\{Z=1\} \cap Y) = 10/100$.

Now think of event $\{Z=1\}$ as a class/type, e.g., those who are marksmen (sharp shooters in national service who can score at least 21 out of 25 shots at training targets), and event $\{Z=0\}$ as the rest of the universe, i.e., those who are not marksmen. In the context of sequential events, $P(\{Z=1\})$ is sometimes called the prior probability or probability with no other information connected to it except for its sample count in the population Ω. Similarly, $P(\{Z=0\})$ is also a prior probability of event $\{Z=0\}$. The Bayes' Theorem is very useful for updating probabilities with sequential information inputs.

Now think of Z as a random variable that can take the value 1 or else 0, i.e., there is a binary classification. Bayes' Theorem can be expressed as:

$$P(Z|Y) = P(Y|Z) \times P(Z) / P(Y) = P(Y|Z) \times P(Z) / [P(Y|Z) \times P(Z) + P(Y|Z^c) \times P(Z^c)]$$

where Z^c is complement of event Z, i.e., $Z^c = \Omega \setminus Z$ (Ω less Z), and $P(Y) = P(Y \cap Z) + P(Y \cap Z^c)$. Clearly, in our notations, $Z^c \equiv \{Z=0\}$. This latter version of Bayes' Theorem is useful in the context of updating prior information or probability of Z, $P(Z)$. Given new information in the form of event Y, the updated or posterior probability of Z is $P(Z|Y)$. Sometimes it may not be convenient to obtain $P(Z \cap Y)$ directly. Hence the computation of posterior $P(Z|Y)$ may be better done via $P(Y|Z) \times P(Z) / [P(Y|Z) \times P(Z) + P(Y|Z^c) \times P(Z^c)]$ where $P(Y|Z)$ and $P(Y|Z^c)$ are more readily available.

A real example is in medical testing, where a patient has apriori (without any testing for further information) a probability $P(Z) = x$ of having the medical condition Z. x (%) could be obtained from some statistical averages over reports of such medical conditions in various countries. A medical test on the

patient could produce information Y, e.g., a positive test case. It may not be easy to obtain total information across all countries of the % of all tests producing Z and Y, i.e., $P(Z \cap Y)$. However, it may be easier to obtain summary reports on $P(Y|Z)$ and $P(Y|Z^c)$, e.g., percentage of cases that have Z and tested for Y, and percentage of non-Z and tested (false positive) for Y.

The Naïve Bayes classification algorithm works using the Bayes Theorem:

$$P(Z|Y) = P(Y|Z) \times P(Z) / P(Y)$$

where $Z = 1$ or else 0 are the two class types we seek for a case under testing and Y is some information/features or evidence we can use to estimate/predict Z.

Suppose it is a prediction of next stock price index change employing sentiment analysis (a kind of natural language processing technique) based on density of mood words prevalent in stock market reports, assuming words like "optimistic", "growth", "boom", etc. would more likely lead to stock index rises, while "pessimistic", "poor", "cautious", etc. would likely lead to stock index drops. Let $\{Z = 1\}$ be the event of next stock price index increase. Therefore $\{Z = 0\}$ is the event of next stock price index decrease or no change. Let features $F_1, F_2, \ldots, F_n$ be mood words in a particular day. These are used for predicting if the stock price index will rise or fall the next day.

Since the mood words are single elements, it is more convenient to use sets to represent collections of these elements. Now, all the features found in a day constitute event set $Y = \{F_1, F_2, \ldots, F_n\}$, the evidence or signal for the day. A feature or mood word can occur many times on a day due to many reports or appearing many times in a single report, but we could just count it as one occurrence for that day.

$$P(Z| F_1, F_2, \ldots, F_n) = P(F_1, F_2, \ldots, F_n |Z) \times P(Z) / P(Y)$$

Naïve Bayes typically naively makes a simplifying assumption. The conditional probability of $P(F_j |Z)$ is assumed to be independent of any other feature F_k, $k \neq j$. In other words, for any j,

$$P(F_j |Z, F_1, F_2, \ldots, F_{j-1}, F_{j+1}, \ldots, F_n) = P(F_j |Z).$$

This assumption makes the computations more tractable. It is however, simplifying, because it is possible that a mood word "downward" (as in downward trend) is likely associated with "poor" (as in poor performance), and both words are not independent given Z=0. The simplification is not ideal and may create ineffectiveness in the prediction.

By Bayes' Theorem,

$$
\begin{aligned}
P(F_1, F_2, \ldots, F_n |Z) &= P(F_1| F_2, \ldots, F_n, Z) \times P(F_2, F_3, \ldots, F_n |Z) \\
&= P(F_1| F_2, \ldots, F_n, Z) \times P(F_2| F_3, \ldots, F_n, Z) \times P(F_3, F_4, \ldots, F_n |Z) \\
&= P(F_1| F_2, \ldots, F_n, Z) \times P(F_2| F_3, \ldots, F_n, Z) \times P(F_3| F_4, \ldots, F_n, Z) \times P(F_4, \ldots, F_n |Z) \\
&= \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&= P(F_1| F_2, \ldots, F_n, Z) \times P(F_2| F_3, \ldots, F_n, Z) \times P(F_3| F_4, \ldots, F_n, Z) \times P(F_4| F_5, \ldots, F_n, Z) \\
&\quad \times P(F_5| F_6, \ldots, F_n, Z) \times \ldots\ldots \times P(F_{n-1}| F_n, Z) \times P(F_n |Z)
\end{aligned}
$$

Using the simplifying assumption on the last line,

$$P(F_1, F_2, \ldots, F_n |Z) = P(F_1|Z) \times P(F_2|Z) \times P(F_3|Z) \times P(F_4|Z) \times P(F_5|Z) \times \ldots.. \times P(F_{n-1}| Z) \times P(F_n |Z)$$

$$= \prod_{j=1}^{n} P(F_j|Z)$$

Then,

$$P(Z| F_1, F_2, \ldots, F_n) = \prod_{j=1}^{n} P(F_j|Z) \times P(Z) / P(Y) \qquad (4.1)$$

Since there are only two classes/types {Z=1} and {Z=0}, we can use the training data set to estimate $\hat{P}(Z=1)$ (percentage of days when next day stock price index increases) and $\hat{P}(Z=0)$ (percentage of days when next day stock price index decreases or remains constant) respectively, where $\hat{P}(Z=1) + \hat{P}(Z=0) = 1$. The method can of course be extendible to consider not just two classes or types (sometimes also called 'categories'), such as using features to predict whether a test student obtains grade in class/type "A+", "A", "A-", "B+", or "B" based on the student's features such as background strength, number of hours put into study, and level of motivation, etc.

Similarly, for all features {F_1, F_2, ..., F_n} in the training data set, we can estimate $\hat{P}(F_j|Z=1)$ (percentage of occurrences of F_j when next day stock price index increases) and $\hat{P}(F_j|Z=0)$ (percentage of occurrences of F_j when next day stock price index decreases or remains constant). Now, suppose a sample point $Y^* = \{F_1, F_3, \ldots, F_m\}$ (m < n) from the test sample is chosen. We suppose we do not know the test point's Z value (the type) and want to use Naïve Bayes algorithm to predict its Z value. Then

$$P(\{Z=1\}| F_1, F_3, \ldots, F_m) = \hat{P}(F_1|Z=1) \times \hat{P}(F_3|Z=1) \ldots \times \hat{P}(F_m|Z=1) \times \hat{P}(Z=1) / P(Y^*)$$

and $\quad P(\{Z=0\}| F_1, F_3, \ldots, F_m) = \hat{P}(F_1|Z=0) \times \hat{P}(F_3|Z=0) \ldots \times \hat{P}(F_m|Z=0) \times \hat{P}(Z=0) / P(Y^*).$

P(Y*) does not need to be computed above as it is the same for both the quantities above. We find

$$\text{Max } \{P(\{Z=1\}| F_1, F_3, \ldots, F_m) , P(\{Z=0\}| F_1, F_3, \ldots, F_m)\}$$

given Y*. The predictor (of class Z, whether 1 or 0) for this test sample point with Y* is

$$\arg \max_{Z=1,0} P(\{Z\}|F_1, F_3, \ldots, F_m) = \arg \max_{Z=1,0} \prod_j \hat{P}(F_j|Z) \times \hat{P}(Z) \qquad (4.2)$$

The accuracy of the Naïve Bayes algorithm is then evaluated based on the performance of each of the predictors of each sample point/case in the test dataset. For small data sets, the Naïve Bayes method is shown to perform reasonably well.

In Eq. (4.1), for estimating $\hat{P}(Z=1)$ and $\hat{P}(Z=0)$, we can use their frequency counts/total number of sample points/cases in the training sample. These estimates add to one. There are several methods for estimating $\hat{P}(F_j|Z=1)$ and $\hat{P}(F_j|Z=0)$ for every j in the training data set, depending on what we assume is the probability distribution of the features F_j. Some common distribution assumptions are the categorical, the Gaussian/normal distributions, and the multinomial. The Naïve Bayes predictor derived from each of these distributions of the features are in turn called the Categorical NB, the Gaussian NB, and the Multinomial NB.

In the sentiment analyses using mood words as features, it is natural to assume that the features F_j, j=1,2,...,n follow the categorical distribution. This means that for any sample point (day), given Z, the probability of a feature F_j occurring is $P(F_j|Z)$ and the probability of the feature not occurring is $1 - P(F_j|Z)$. Thus, any feature F_1 through F_n can either occur or not occur. The categorical variable of F_j or no F_j can be hot-encoded as dummy variable 1 or 0. If there is only one feature in the dataset, then this is also a Bernoulli Naïve Bayes. In any one day, different categorical variables F_1, F_2, etc. can occur

together. Here we have made the simplification that a dictionary of mood words is created for the study such that all the mood words in this dictionary appear at least once in the training data set in the cases Z=1 and in the cases Z=0. In other words, we do not have a situation of 'zero frequency'.

For example, suppose we have 700 days in our study for the training set and 300 other days for the test set. Suppose the dictionary contains 100 mood words. Any one of these 100 words would have occurred in at least one day of the 700 where the next day has Z=1 and would also have occurred in at least one day of the 700 where the next day has Z=0. If in the training set of 700 days, 500 have Z=1 or index rise the day after, and 200 have Z=0 which represents index decrease or staying constant the day after. Of the 500, 200 days have reports containing "optimistic", 100 days have reports containing "growth", and so on. For estimating $\hat{P}(F_j|Z = 1)$, we can use their frequency counts of occurrence of F_j /total number of cases given Z in the training sample. Here, $\hat{P}(F_j = \text{"optimistic"} |Z = 1) = 200/500 = 0.4$. $\hat{P}(F_j = \text{"growth"} |Z = 1) = 100/500 = 0.2$. Of the other 200, 4 days have reports containing "optimistic", and 10 days have reports containing "growth", and so on. Then, $\hat{P}(F_j = \text{"optimistic"} |Z = 0) = 4/200 = 0.02$. $\hat{P}(F_j = \text{"growth"} |Z = 0) = 10/200 = 0.05$. Note that both "optimistic" and "growth" can occur together in any one day.

Sometimes, the probability estimates may be too small or close to zero and can make the prediction in Eq. (4.2) difficult because of the nearly zero probability on the right-side. To make the numerical computation more robust, a smoothing parameter $\alpha > 0$ (usually not larger than 1) is added in the estimation of $\hat{P}(F_j |Z = 1)$ and $\hat{P}(F_j |Z = 0)$. For example, $\hat{P}(F_j = \text{"boom"} |Z = 0) = (2+\alpha)/(200+\alpha n)$ where n is the number of features.

Having found $\hat{P}(F_j|Z)$ and also $\hat{P}(Z)$ for all j and for Z=1 and Z=0, we can take a test sample point from the 300 test cases (the other 300 days), find the number of features that occur at least once, e.g., F_1, F_3, F_4, F_7,…,F_m out of the 100 words in the dictionary (based on the training set), then evaluate (4.2) accordingly.

Sometimes the features may appear to be organized into sub-categories, e.g., daily reports could come from 3 sources, viz., news, analysts' reports, and social media. Each source would have features such as the mood words; the same mood words could occur in more than one source. Instead of using an extension of Eq. (4.1):

P(Z| news{$F_1, F_2, …, F_n$}, analysts' {$F_1, F_2, …, F_n$}, social media{$F_1, F_2, …, F_n$})
$= \prod_{j=1}^{n} P(F_j|Z, \text{news, analysts}', \text{social media}) \times$ P(news, analysts', social media | Z) $\times$ P(Z) / P(Y)

where Y contains not {$F_1, F_2, …, F_n$}, but news {$F_1, F_2, …, F_n$}, analysts'{$F_1, F_2, …, F_n$}, social media{F_1, F_2, …, F_n}, i.e., including sources and their mood words. We often could avoid the unnecessary complication, and use instead news-F_j, analysts'-F_j, social media-F_j, for all j, as separate complex features. For estimating $\hat{P}(\text{news}: F_j|Z = 1)$, we can use their frequency counts of occurrence of news:F_j /total number of cases given Z in the training sample. There is some expansion in the number of features, but the computational method remains the same as in the categorical NB.

Now suppose there are other non-natural language measures of features G_j that are continuous, such as volatility (j=1), volume traded on index futures (j=2), lagged index change (j=3), etc., so that it is not feasible to count category frequency to estimate $\hat{P}(G_j|Z)$. In such a situation, we can use the Gaussian Naïve Bayes (GNB) method which employs the Gaussian or normal probability distribution suited for continuous variables/features. GNB assumes that $P(G_j|Z)$ (for each continuous variable/feature G_j) follows a Gaussian/normal probability distribution. Suppose Z has two classes/types/categories, Z=1 or Z=0. The conditional probability density function of each feature G_j is given by

$$P(G_j|Z) = \frac{1}{\sqrt{2\pi\sigma_{zj}^2}} \exp\left(-\frac{1}{2}\left[\frac{G_j - \mu_{zj}}{\sigma_{zj}}\right]^2\right)$$

where $\mu_{zj} = E(G_j|Z)$ and $\sigma_{zj} = \sqrt{var(G_j|Z)}$. These can be estimated using the sample mean and sample standard deviation of the feature j values given Z, $\hat{\mu}_{zj}$ and $\hat{\sigma}_{zj}$. If it is further assumed that these parameters are independent of Z, then $\mu_{zj} = \mu_j = E(G_j)$ and $\sigma_{zj} = \sigma_j = \sqrt{var(G_j)}$, and these can be estimated using the unconditional sample mean and sample standard deviation of the feature j values. And if the assumption is that these parameters are independent of both Z and j, then $\mu_{zj} = \mu = E(G)$ and $\sigma_{zj} = \sigma = \sqrt{var(G)}$ where no subscripts for G indicate that the sampling mean and variance are taken over all features, across all j.

Then, the predictor (of class Z, whether 1 or 0) for each test sample case with Y (with its associated features $\{G_j\}$) is

$$\arg\max_{Z=1,0} P(\{Z\}|G_1, G_2, \ldots, G_n) = \arg\max_{Z=1,0} \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi\hat{\sigma}_{zj}^2}} \exp\left(-\frac{1}{2}\left[\frac{G_j - \hat{\mu}_{zj}}{\hat{\sigma}_{zj}}\right]^2\right) \times \hat{P}(Z) \qquad (4.3)$$

where $\hat{P}(Z)$ is the frequency count given Z obtained from the training data set, and $\hat{\mu}_{zj}, \hat{\sigma}_{zj}$ are estimates from the training data where Z is known or given. Y is the information containing all the non-natural language features. Note that with GNB, there is no need to first standardize/normalize the features first before training if the test data features also have the same mean and variance as those in the training data since $\hat{\mu}_{zj}, \hat{\sigma}_{zj}$ estimates from the training data would suitably represent parameters from the test data set. However, if the mean and variance from the test data set are different from those in the training data set (even if features from both sets are Gaussian), it may be more robust to first standardize the training and the test data features before applying the GNB method. After standardizing, the $\hat{\mu}_{zj}, \hat{\sigma}_{zj}$ estimates of the standardized features are 0 and 1 respectively, for both the training and test set data.

For mixed type of data in features, e.g., some features F_j are categorical while others G_j are continuous variables, we can apply the NB method as follows. Following Eq. (4.1):

$$P(Z| F_1, F_2, \ldots, F_u, G_1, G_2, \ldots, G_v) = \prod_{j=1}^{u} P(F_j|Z) \prod_{j=1}^{v} P(G_j|Z) \times P(Z) / P(Y)$$

where $Y \equiv \{F_1, F_2, \ldots, F_u, G_1, G_2, \ldots, G_v\}$. The independence of all features conditional on Z is assumed. Then, the predictor (of class Z, whether 1 or 0) for each test sample case with Y is

$$\arg\max_{Z=1,0} P(\{Z\}|F_1, F_2, \ldots, F_u, G_1, G_2, \ldots, G_V)$$

$$= \arg\max_{Z=1,0} \prod_{j=1}^{u} \hat{P}(F_j|Z) \prod_{j=1}^{v} \frac{1}{\sqrt{2\pi\hat{\sigma}_{zj}^2}} \exp\left(-\frac{1}{2}\left[\frac{G_j - \hat{\mu}_{zj}}{\hat{\sigma}_{zj}}\right]^2\right) \times \hat{P}(Z)$$

The maximization can also be taken over the logarithm.

Now suppose the continuous features do not actually follow Gaussian distributions. Then the Gaussian NB may yield biased predictions due to the incorrect distributional specification. Suppose the probability distribution of the features is unknown. In this case we may use the Multinomial NB. Unlike categorical variables where features $\{F_j\}_j$ can occur together in a test sample point where each feature F_j de facto takes dummy value 1 if the feature occurs or dummy value 0 if the feature does not occur, a feature with multinomial distribution basically can take on only one of several possible occurrences, e.g., the

outcome of drawing a blue or red or green ball from a bag, and each draw or trial is repeated a number of times so that the feature can be characterized as 5 blue, 2 red, 3 green in 10 draws. The next feature could be 4 blue, 3 red, 3 green. Yet the next could be 6 blue, 1 red, 3 green, and so on.

If the training data set under condition of Z=1 yields most cases with features having more blue balls, while under condition of Z=0 yields most cases with features having more red balls, then the observed features in a test case with more blue could likely predict that the case is Z=1. For each case, there could be multiple features each with a different multinomial distribution, but here we use only one such multinomial feature.

More formally, suppose for a sample point (day) in the training data set, let V_u denote the event that volatility is up and V_d denote the event that volatility is down. Let L_u denote the event that trading volume is up and L_d denote the event that trading volume is down. . Let I_u denote the event that the index is up and I_d denote the event that index is down. Then the trading feature have one of 2^3 or 8 possibilities, viz., (V_u,L_u,I_u), (V_u,L_u,I_d), (V_u,L_d,I_u), (V_u,L_d,I_d), (V_d,L_u,I_u), (V_d,L_u,I_d), (V_d,L_d,I_u), (V_d,L_d,I_d), each with probability θ_1, θ_2, …, θ_8 where $\sum_{i=1}^{8} \theta_i = 1$. Suppose the day is broken into 3 sessions: early morning, late morning, and early afternoon in the trading sessions of the exchange. Then the above possibilities can be measured 3 times during the day. These possibilities combined into a feature become a multinomial distribution with 3 trials.

Under condition of Z=1 (next day index rise), the probability of one training sample point could be (V_u,L_u,I_u), (V_u,L_u,I_u), (V_d,L_u,I_u). This has a multinomial probability of $3! \times (\theta_1^2 \times \theta_5^{3-2}) / (2! \times [3-2]!)$. In general, $\hat{\theta}_i$ is estimated from the training data set using frequency count of the number of occurrences of the possibility divided by all trading features (number of training set days times the 3 sessions). The same process is done for the condition of Z= 0. Then for a test sample point (day), the possibility in each of the 3 sessions is noted and these form the multinomial probability of the feature H_j for the day. In the above, $H_j = (2,0,0,0,1,0,0,0)$ where 2 denotes two draws on the first possibility and 1 denotes one draw on the fifth possibility. The predictor of Z becomes

$$\arg \max_{Z=1,0} P(\{Z\}|H_j) = \arg \max_{Z=1,0} n! \times \left[\frac{\left(\hat{\theta}_1^{x_1} \hat{\theta}_2^{x_2} \hat{\theta}_3^{x_3} \hat{\theta}_4^{x_4} \hat{\theta}_5^{x_5} \hat{\theta}_6^{x_6} \hat{\theta}_7^{x_7} \hat{\theta}_8^{x_8} \right)}{x_1! x_2! x_3! x_4! x_5! x_6! x_7! x_8!} \right] \times \hat{P}(Z) \qquad (4.4)$$

where $x_1 + x_2 + …. + x_8 = n$. For Z=1 and H_j, $x_1= 2$, $x_5=1$, and the other x_j's are all zeros. The accuracy of the Naïve Bayes algorithm is then evaluated based on the performance of each of the predictors of each sample point/case in the test dataset.

When sample data is split into training and test data sets, the Naïve Bayes algorithm is first applied to the training data set to compute the classification accuracy. More features can be added or selected so that the training set can be adjusted to optimize the accuracy of a balanced data set. In GNB, the training data set also provides estimation of the feature parameters of means and variances in order to compute the conditional probabilities for predicting the labels or targets. If there are no competing models or no hyperparameters to optimize, the optimized and estimated parameters are then applied to predict cases from the test features. The predictions are compared with the test labels to find the various performance metrics as well as the ROC AUC.

Naïve Bayes is typically used for classification or prediction of classes/types. However, the method may be adjusted for regression prediction of quantitative or numerical values. The Gaussian Naïve Bayes (GNB), making use of the Gaussian or normal probability distribution, is a type of Naïve Bayes method that can be applied to predict quantitative/numerical targets. Here the targets Z^* are not classes/types. Z^* is a continuous variable, e.g., the next day stock index value. Suppose we use features such as volatility (j=1),

volume traded on index futures (j=2), and lagged index change (j=3) to help predict the next day index level Z^*. There are possibly several methods and research on new methods is still ongoing.

One method is to partition the continuous values of Z^* into a finite number of q intervals where the mid-point of an interval serves as discrete value approximations to the continuous values in the interval. Denote these discrete points as $\{Z_k^{**}\}$ for k=1,2,…,q. They are de facto classes or types. Then we use the features to predict which class/type will occur. For a sample point in the test data, from the right-side of Eq. (4.3), find $P(Z_k^{**})$, given the features, for the different possible values of Z_k^{**}. Then predict Z^* using $\Sigma_{k=1}^q$ $Z_k^{**} \times P(Z_k^{**})$. More sophisticated methods use non-parametric kernel density estimation.

4.2 k-Nearest Neighbors Algorithm or k-NN (kNN)

Another popular method for predicting classification is the k Nearest-Neighbor algorithm.
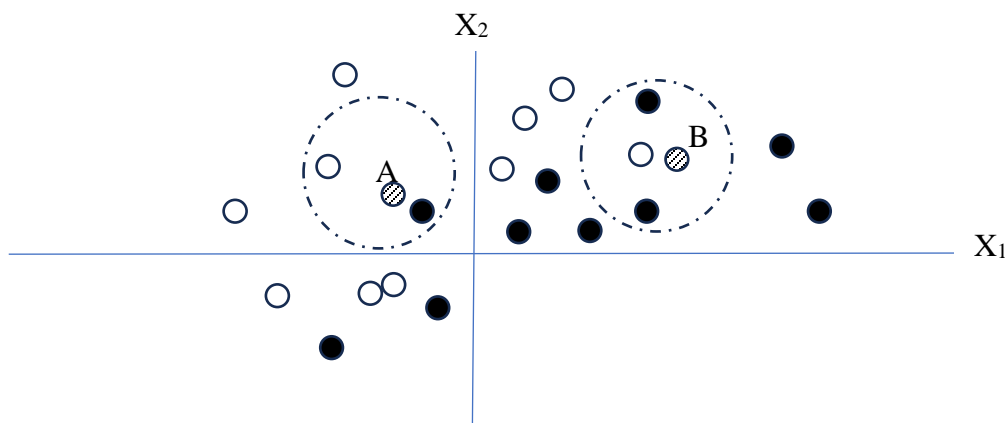

Figure 4.1

In the diagram, the training data are 20 sample points represented by the black circles (target value/label 1) and the white circles (target value/label 0). Each sample point has two features represented by values in (X_1, X_2). The feature values X_k are obtained by transforming the raw features using the standard scalar (where all transformed values X_k have mean 0 and standard deviation 1) or else by MinMaxScaler (where all transformed values X_k lie within [0,1]). The transformation is useful in k-NN algorithm so that outliers or some features with much larger values on a larger scale would not create bias when the distances are measured.

Sometimes categorical variables are found in available features, e.g., X_3 = male or else X_3 = female. While categorical variables can be a problem in k-NN since they do not operate on a quantitative basis with ratio or else relative comparisons, they may be usable if we provide a suitable dummy. In this case, we can expand gender to two dummy features, where $X_{31} = \alpha$ if male and 0 if female, and $X_{32} = 0$ if male and α if female. α should be chosen to be of the same order of magnitude as the rest of the standardized/scaled features. Too small an α close to zero may render the categorical/qualitative feature ineffective. Too large an α may unintentionally outsize the effect of categorical/qualitative feature.

Another example is X_4 = unemployed or blue-collar or else white-collar worker. In k-NN this may be fixed as $X_{41} = 1$ if unemployed and 0 otherwise, $X_{42} = 1$ if blue-collar and 0 otherwise, and finally $X_{43} =$

1 if white-collar and 0 otherwise. Using dummies will of course increase the number of features and expand the dimensionality of the dataset, which makes computations more tedious.

In the diagram, the test set data are entered one at a time. Sample point A from the test set is shown with the given features in X_{A1} and X_{A2}. The distance between any pair of points u and v is measured typically using the Euclidean distance which is

$$\sqrt{\sum_{k=1}^{2}(X_{u1} - X_{v1})^2}$$

where (X_{u1}, X_{u2}) are the feature values of point u, and (X_{v1}, X_{v2}) are the feature values of point v. Sometimes the square of this distance is used as the distance metric. Other distance metrics include the Manhattan distance $\sum_{k=1}^{2}|X_{u1} - X_{v1}|$, the Minkowski distance $\left(\sum_{k=1}^{2}|X_{u1} - X_{v1}|^p\right)^{1/p}$ for some p ≥ 1, and so on.

If k=1 or using 1 Nearest-Neighbor algorithm, then only the nearest neighbor (of the training sample points) to A, i.e., shortest distance from A, will be picked. In this case, it is a black circle. The prediction for test case A is therefore black or target value/label 1. If k=3 or using 3 Nearest-Neighbor algorithm, then only the nearest 3 neighbors to A (of the training sample points) will be picked. In this case, the 3 neighbors are within the dotted circle showing 2 white and 1 black circle. The probability of white circle for the test point is 2/3 and probability of black circle for the test point is 1/3. Since the majority is white (or the higher probability > 0.5), the 3-NN algorithm for test case A is white or target value/label of 0. Note that in k-NN algorithm, there is de facto no active computations done on the training data, so the training is sometimes called lazy learning or fitting. The use of the training data is delayed until a test query is made.

For test sample point B, using 1 Nearest-Neighbor, the prediction of B is white or label 0 since the white circle (of the training sample points) is nearest. But for 3-Nearest-Neighbor, the prediction of B is black or label 1 since within the dotted circle of the 3 closest neighbors (of the training sample points), there is a majority of black circles (or probability 2/3 > 0.5 of black circles).

In using the k-NN algorithm, one important and about the only hyperparameter (there may be other hyperparameters if some other more complicated distance metric is used) to choose is k. In general, for k-NN algorithm, k is chosen to be odd so as to avoid a tie. We may have to re-run the algorithm several times for different hyperparameter values of k (tuning k) before we reach a satisfactory prediction or classification accuracy and good AUC. When k is decreased toward 1, the prediction for each case becomes unstable as prediction is based only on what happens to be the nearest neighbor of the test sample point. The prediction is myopic and may miss the correct affiliation as it may happen that there are more of the incorrect categories that are closer. But as k is increased, the prediction becomes more stable due to majority voting/averaging phenomenon, but this prediction can worsen as k becomes too large when the expanding neighborhood gets in more of both categories or types of the sample points. The k-NN algorithm also becomes slower as the number of features used for computing distances increases.

The hyperparameter of k could be selected using a validation data set. Another way of tuning k is to implement cross-validation, e.g., splitting the data set into 5 equal-sized groups and using 4 for training k while keeping 1 as the validation set. After optimal k is selected, the k-NN algorithm can then be applied to the test set to check the test sample prediction performance.

In the case of regression where k-NN is used to predict a target value based on the test sample point's features, the average of the values of the training cases in the nearest-neighbor dotted circle will be

used as the prediction.[1] Note that k-NN is a non-parametric algorithm. There is no need for any assumption of probability distributions on the features, and therefore avoids distributional misspecification errors.

If a test point sits in the k-neighborhood of m number of white circles and k-m of black circles, then the predicted probability of a white circle of the test point may be construed as m/k while the probability of a black circle as predicted value is $1 - m/k$. Typically $m/k > 0.5$ (a hyperparameter threshold) implies prediction of white, otherwise black. But for constructing the Receiver Operating Characteristics (ROC) Curve in a binary classification, the hyperparameter threshold of white circles can be varied between 0 and 1 so that only when the predicted probability of white is higher than the threshold, the prediction is white, otherwise black. Thus, ROC can be constructed for the k-NN algorithm.

4.3 Support Vector Machine

Another algorithm for making class/type prediction is the Support Vector Machine (SVM). We first provide the ideas behind the SVM method before we dive more into the theory.

Suppose we try to predict the targets/labels/classes of the cases represented as black or white circles. The diagram shows a simple example of cases with two features with values X_1 and X_2. In the training of the SVM, a boundary is drawn to separate the two classes/groups as far as is possible. In this example, the colored circles are easily separated by a linear straight line.
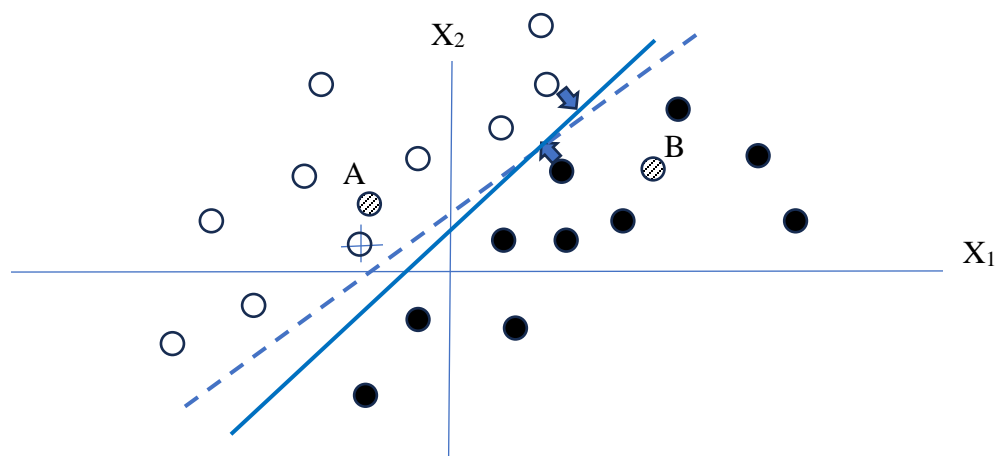

Figure 4.2

As shown in Figure 4.2, the maximum separation is attained via the solid boundary line with positive slope. Both the colored circles are clearly separated. There, the nearest white circle to the boundary and the nearest black circle to the boundary have equal distances represented by the arrows. Any shift in the boundary (see dotted line) would end up having at least one colored circle being closer to the boundary, e.g., the white circle shown with a cross. Hence the linear SVM provides for a solid line boundary with the

---

[1] Note that there is a related algorithm that is unsupervised and used to group a set of data points into clusters – called k-means clustering. Each point within the cluster is closest to the cluster centroid/mean and there are k such clusters. This is not the same as k-NN for supervised prediction.

maximum margin (sum of perpendicular distances from the nearest feature points on either class to the line). The two closest points from both classes are called the support vectors as they help determine the boundary. In other words, if these support vectors are shifted, then the boundary will also change.

Once the SVM has fitted the training data set, i.e., determined the boundary to separate the classes in the training data set, then the test data sample points, e.g., A and B, can be easily predicted as being in the white circle and the black circle group/class respectively. Of course, the prediction need not be always accurate as we suppose that the test sample target values are not observable during the prediction. The latter is called a generalization or out-of-sample error when trained/fitted model is used to predict the class of test sample points.

Now suppose we use the same example of the small-sized 20 training data set seen in the k-NN algorithm. Figure 4.3 below shows that it is not possible to find a linear straight line to separate the two classes/groups of colored circles. When we try to draw a straight line to keep the black circles on one side (see dotted lines in the diagram), it will always rope in some white circles as well. Vice-versa for trying to keep the white circles on one side.
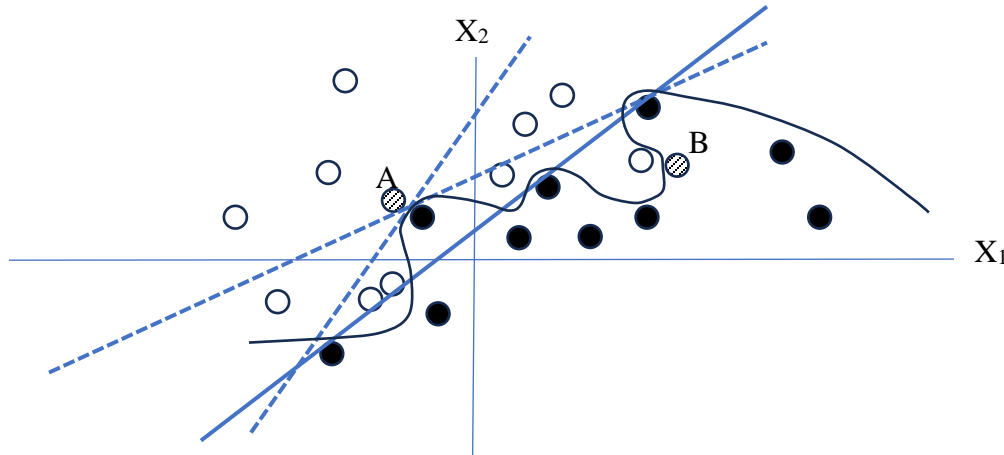

Figure 4.3

In SVM there are at least two ways to proceed. Firstly, we could still apply linear boundary as in the solid line but allow some classification errors in training. In this case, there is one mis-classified black circle (black circle in the white circled upper-half plane) and one mis-classified white circle (white circle in the black circled lower-half plane). Secondly, we may be able to find a polynomial function (shown in the curved line) to separate the colored circles. This involves transforming the original n training data points $(X_1^{(1)}, X_2^{(1)})$, $(X_1^{(2)}, X_2^{(2)})$, $(X_1^{(3)}, X_2^{(3)})$, …. , $(X_1^{(N)}, X_2^{(N)})$ to new points $\phi(X_1^{(1)}, X_2^{(1)})$, $\phi(X_1^{(2)}, X_2^{(2)})$, $\phi(X_1^{(3)}, X_2^{(3)})$, …. $\phi(X_1^{(N)}, X_2^{(N)})$ that may be in the same 2-dimensional space or in a larger dimensional space. What is significant here is that under ϕ transformation, the new points $\phi(X_1^{(1)}, X_2^{(1)})$, $\phi(X_1^{(2)}, X_2^{(2)})$, $\phi(X_1^{(3)}, X_2^{(3)})$, …., $\phi(X_1^{(N)}, X_2^{(N)})$ becomes linearly separable. After the training or fitting, a test data sample point t's features are subject to the transformation so that $\phi(X_1^{(t)}, X_2^{(t)})$, $\phi(X_1^{(t)}, X_2^{(t)})$, $\phi(X_1^{(t)}, X_2^{(t)})$, …. , $\phi(X_1^{(t)}, X_2^{(t)})$ can be checked if it falls in which separable groups to predict its affiliation.

Often there are complicated grouping patterns such as the following whereby a transformation must be applied to the features to separate the classes. This transformation involves a higher-dimensional feature

space. Such a transformed feature space with a higher dimension than the original one for the purpose of separating the classes/types is often called a "kernel trick".
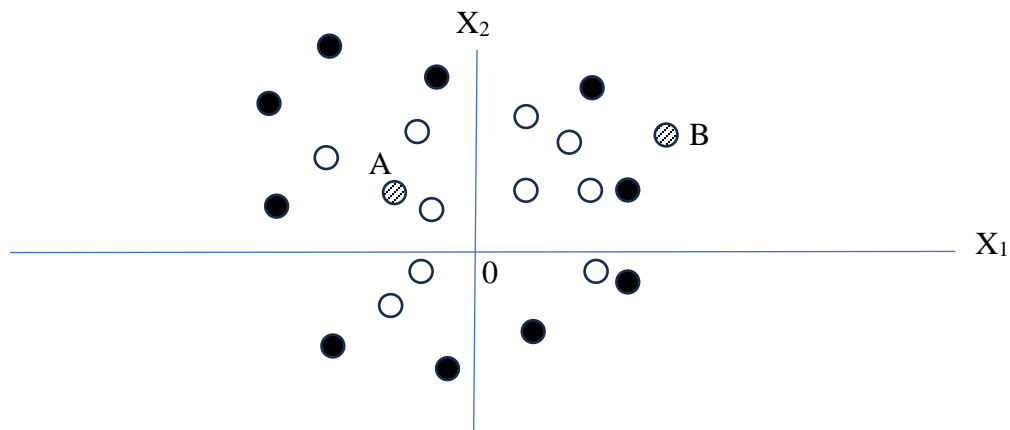

Figure 4.4

As a specific (need not be optimal) solution, we could use the transformation $\phi(X_1^{(j)}, X_2^{(j)}) = (X_1^{(j)}, X_2^{(j)}, X_3^{(j)})$ for the jth training data set sample point, where the third coordinate is $X_3 = X_1^2 + X_2^2$. By extending the feature space, we can obtain the following 3-dimensional representation of the circles in Figure 4.5. Now the black circles will have higher X_3 values since they lie on an outer circle in the X_1-X_2 axes. (Drawing may not be exactly to scale.)
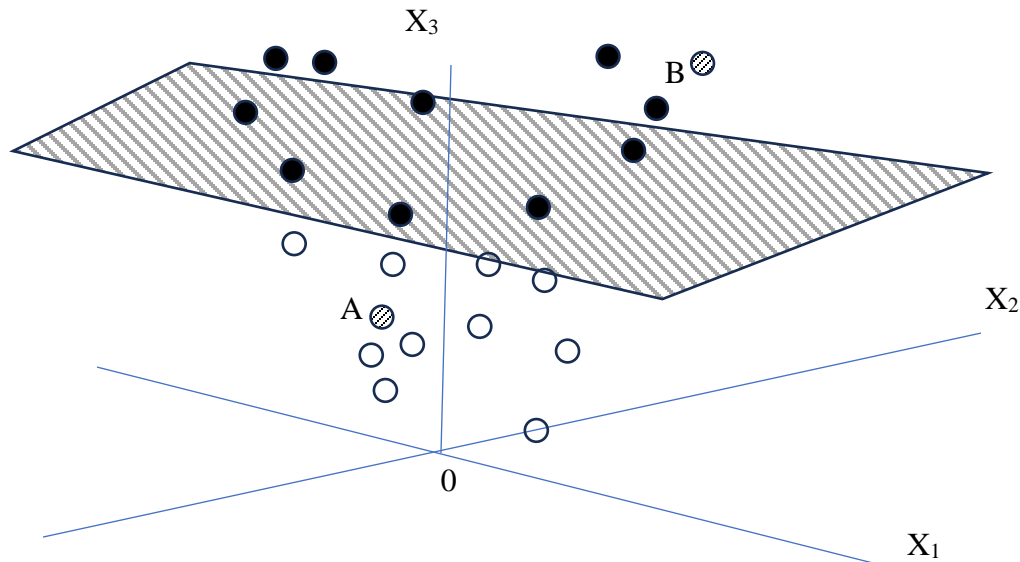

Figure 4.5

In the 3-dimensional representation, a plane (shaded) can be seen to separate the black circles from the white circles below. Clearly now A and B from the test sample can be easily predicted as belonging to the white and black circle groups respectively.

More formally, the theory of SVM can be explained as follows.[2] Suppose a training sample consists of N number of data or sample points. Each sample point has p features and the jth sample point can be characterized in Euclidean space as a p-dimensional vector $(X_1^{(j)}, X_2^{(j)}, X_3^{(j)}, ..., X_p^{(j)})$. In addition, each point belongs to one of two classes. Depending on the problem, the binary or dichotomous classes could be diseased patient versus non-diseased patient using medical and physical attributes to predict, spam or no-spam emails using natural language processing of the words to predict, rating R or non-R classification using film visual features, detection of presence or absence of specific object using the object features, weather prediction of rain or no-rain using weather conditions as features, stock decrease or no decrease using market conditions and the firm conditions as features, and so on. In the categorical nature of the class/type, there is no loss of generality in assigning numerical value +1 or -1 to the jth sample point category/type, i.e., let Y_j = +1 or −1 depending on which on the binary class/type.

Consider again the two-dimensional features in an earlier example. We want to draw a hyperplane (a straight line in 2-dimension) to separate the black circles (target/label Y_j = −1) from the white circles (target/label Y_j = +1). Moreover, we want a straight line with the widest margin (wide "street") with the two margin lines (shown as bold lines) touching the support vectors or the points closest to (or on) the lines. The double-arrowed line in Figure 4.6 shows the extent of the margin.

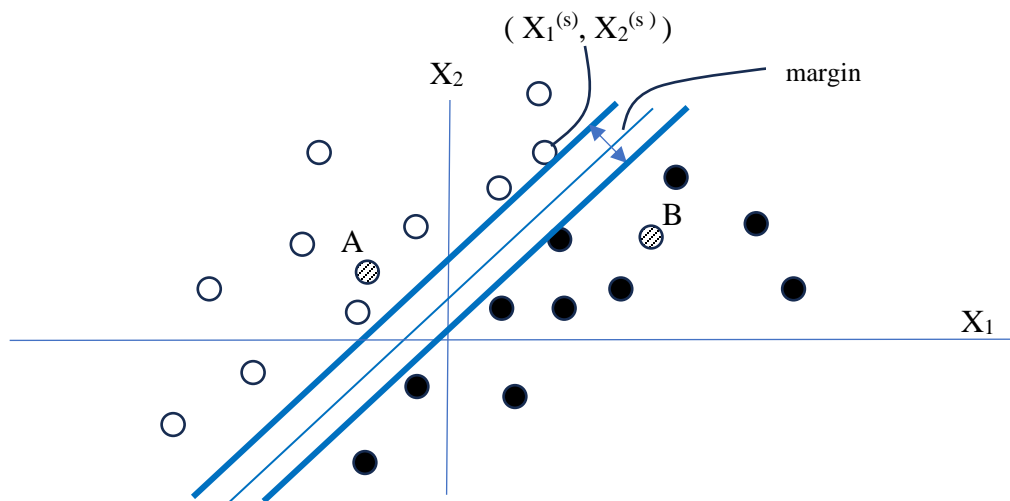

Figure 4.6

In the 2-D case here, the straight lines can be represented algebraically as equations $aX_1+bX_2 = c$ for different values of c. This is a more general representation (allowing b to be possibly 0) of the familiar $X_2 = c/b – a/b X_1$ where slope is $ – a/b$ and the intercept on the X_2 axis id c/b.

Let the supporting vector on white circle lie on line $AX_1+ BX_2 − C = +1$. Hence slope of margin line is $ − A/B$. Intercept of line is (C+1)/B. Let the supporting vector on black circle lie on line $AX_1+ BX_2 − C = −1$. Hence slope of margin line is $ − A/B$. Intercept of line is (C−1)/B. The separating line in the middle of the margin lines is $AX_1+ BX_2 − C = 0$. (B > 0, C > 1, and A < 0 so that the slopes and intercepts are correct as in the diagram.) The white circles or points have features that obey $AX_1+ BX_2 − C \geq +1$, i.e., the white circles lie above the margin line $AX_1+ BX_2 − C = +1$. The black circles or points have

[2] See also Vladimir N. Vapnik, (1998), "Statistical Learning Theory", John Wiley, and V. Vapnik and R. Izmailov, (2018), :Rethinking statistical learning theory: learning using statistical invariants, Machine Learning, 1-43. V. Vapnik is a pioneer of the Support Vector Machine methods.

features that obey $AX_1 + BX_2 - C \leq -1$, i.e., the black circles lie below the margin line $AX_1 + BX_2 - C = -1$.

Consider any perpendicular line from a point on $AX_1 + BX_2 - C = +1$ to a point on $AX_1 + BX_2 - C = -1$. The slope of this perpendicular line is $-1/(-A/B) = B/A$. One such perpendicular line is $X_2 = [B/A] X_1$ or $AX_2 - BX_1 = 0$. The intersection of this perpendicular with line $AX_1 + BX_2 - C = +1$ is point with co-ordinates ($A[C+1]/[A^2+B^2]$, $B[C+1]/[A^2+B^2]$). The intersection of this perpendicular with line $AX_1 + BX_2 - C = -1$ is point with co-ordinates ($A[C-1]/[A^2+B^2]$, $B[C-1]/[A^2+B^2]$). Therefore, the distance between these two intersection points is the shortest distance between the two margin lines. This distance is

$$\sqrt{\left(\frac{2A}{A^2+B^2}\right)^2 + \left(\frac{2B}{A^2+B^2}\right)^2} = \sqrt{\frac{2^2}{A^2+B^2}} = \frac{2}{\sqrt{A^2+B^2}} \; .$$

Let vector $W = (A,B)$ and the separating line can be written as $W \bullet X - C = 0$, where X is vector (X_1,X_2), and operation "$\bullet$" denotes the dot product. Then $W \bullet W = A^2 + B^2$ where $W \bullet W$ is also matrix multiplication of transpose of W with W. In Euclidean space, $W \bullet W = \|W\|^2$ under Euclidean norm $\|.\|$. So, the perpendicular distance between two margin lines is $2/\|W\|$ (for the two-dimensional case here). The task in this linear SVM is to find the minimum $\|W\|$ for the maximum margin $2/\|W\|$ of the wide "street".

In the more general p-dimension (p>2), let vector $X_j = (X_1^{(j)}, X_2^{(j)}, X_3^{(j)}, \ldots, X_p^{(j)})$ for the jth sample point in the training data set. $X_j \in R^p$. If subscript j is not highlighted, then the point is any from the training data set. Let $W \in R^P$ be a p-dimensional vector representing coefficients of X in a hyperplane $W \bullet X - C = $ constant. This vector notation of a sample point X_j in the p-dimensional space should not be confused with the 2-dimensional X_2-X_1 axes.

Suppose one margin hyperplane is $W \bullet X - C = +1$ where all training sample points lying on or "above" (i.e., $W \bullet X - C > +1$) it have $Y_j = +1$. The other parallel margin hyperplane is $W \bullet X - C = -1$ where all training sample points lying on or "below" (i.e., $W \bullet X - C < -1$) it have $Y_j = -1$. Then for any point that lies on $W \bullet X - C = +1$ when $Y_j = +1$, $Y_j (W \bullet X_j - C) = 1$. And for any point that lies on $W \bullet X - C = -1$ when $Y_j = -1$, $Y_j (W \bullet X_j - C)$ also equals to 1. Hence, for any point X_j on the margin hyperplanes

$$Y_j (W \bullet X_j - C) - 1 = 0 \qquad\qquad\qquad (4.5)$$

In general, for any point X_j of the training sample,

$$Y_j (W \bullet X_j - C) \geq 1 \text{ for all } j \in [1,N] \qquad\qquad\qquad (4.6)$$

For a general X, the Euclidean distance between point vector X_0 and X that lie on hyperplane $W \bullet X - C = +1$ is $\sqrt{(X_0 - X) \bullet (X_0 - X)}$ subject to $W \bullet X - C = +1$. If the line connecting X_0 and X is perpendicular to $W \bullet X - C = +1$, then X can be found by

$$min_X \; (X_0 - X) \bullet (X_0 - X) - \lambda (W \bullet X - C - 1)$$

where λ is Lagrange multiplier[3] on constraint $W \bullet X - C - 1 = 0$. Solving the quadratic problem (convex) with global minimum, first derivative w.r.t vector X gives $-2((X_0 - X) - \lambda W = 0$. This yields two more equations for solution, viz.,

---

[3] Lagrange multiplier adds the constraints into the objective function to be optimized. First order conditions can be derived directly from a Lagrangian. The Lagrange multiplier λ also serves as the shadow price of the resource

$$W \bullet [-2 (X_0 - X) - \lambda W] = 0 \Rightarrow \lambda = -2 \, W \bullet (X_0 - X)/W \bullet W$$

and $\quad (X_0 - X) \bullet [-2 (X_0 - X) - \lambda W] = 0 \Rightarrow -2 (X_0 - X) \bullet (X_0 - X) = \lambda (X_0 - X) \bullet W$

$\Rightarrow -2 (X_0 - X) \bullet (X_0 - X) = [- 2 \, W \bullet (X_0 - X)/W \bullet W] \, (X_0 - X) \bullet W$

$\Rightarrow (X_0 - X) \bullet (X_0 - X) = [W \bullet (X_0 - X)]^2 / \, W \bullet W = [W \bullet X_0 - (C + 1)]^{\,2} / \, W \bullet W$

Hence the perpendicular distance is $\sqrt{\frac{(W \bullet X_0 - (C+1))^2}{W \bullet W}}$. Now if X_0 lies on parallel hyperplane $W \bullet X - C$

$= -1$, then $W \bullet X_0 - C = -1$, so the perpendicular distance becomes $\sqrt{\frac{((C-1)-(C+1))^2}{W \bullet W}} = \frac{2}{\sqrt{W \bullet W}} = 2/\|W\|$.

Hence this is also the margin length or width of the "street" between the two parallel hyperplanes.

In the linear SVM problem, we maximize the margin or the width of the "street" $2/\|W\|$ to find the supporting vectors, hence also the separating hyperplane $W \bullet X - C = 0$ in the middle of the two parallel margin hyperplanes. This implies minimizing $\|W\|$. The solution would be the same as minimizing $\frac{1}{2} \|W\|^2$ which is mathematically a more convenient objective function. This minimization over the training sample space is constrained by the support vectors in Eq. (4.5).

Hence,

$$min_{\,W,C,\lambda_i} \quad \tfrac{1}{2} W \bullet W - \sum_{i=1}^{N} \lambda_i \, [Y_i \, (\, W \bullet X_i - C) - 1]$$

where there are N constraints, hence N Lagrange multipliers λ_i.
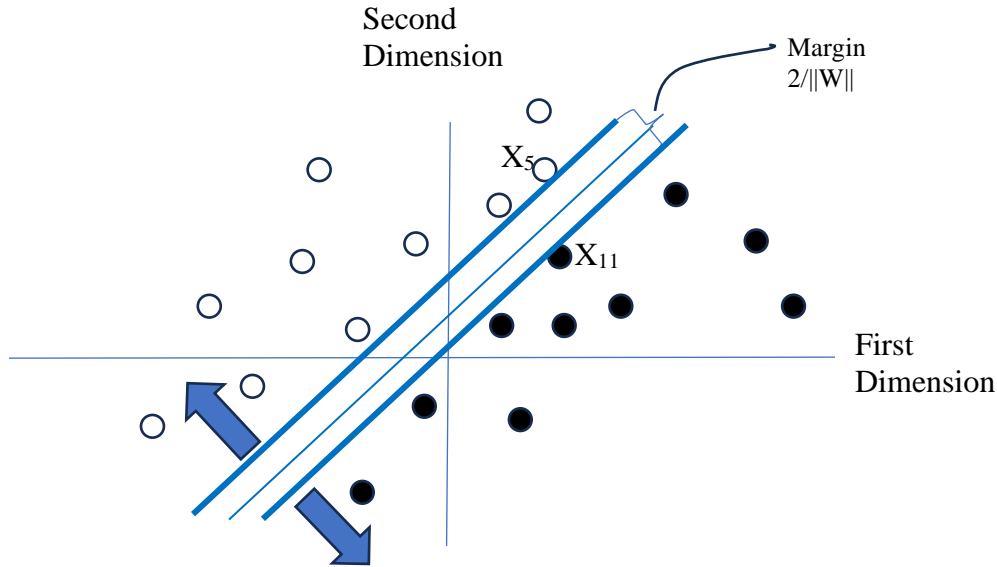

Figure 4.7

Figure 4.7 shows the new notations indicating two training sample points X_5 and X_{11} as the support vectors on the margin hyperplanes separating the two classes. X_5, X_{11}, and all the other sample points

---

constraint, i.e., at optimal solution, it is the amount of increase in the objective function given an increase in the resource.

(circles) are p-dimensional, although we represent only two dimensions in the diagram. X_5 lies on $W \bullet X - C = +1$ while X_{11} lies on $W \bullet X - C = -1$. At these two points, constraint in Eq. (4.5) is satisfied.

Note that if support vectors $W \bullet X - C = +1$ or $W \bullet X - C = -1$, seen as points X_5 and X_{11} draw farther away from each other, like the other non-support vectors, as indicated by the two thick arrows trying to widen the "street", this creates a smaller objective function. This situation when $[Y_i (W \bullet X_i - C) - 1]$ moves from zero toward being positive as in Eq. [4.6] and reducing the objective (increasing "street" width) means that λ_i should be positive to be consistent. Hence in the Lagrangian, $\lambda_i \geq 0$. At the optimal solution to the objective function, only small changes in the support vectors affect the objective, hence at the support vectors X_5 and X_{11}, λ_5 and $\lambda_{11} > 0$. For all the other non-constraining sample points X_j where $Y_j (W \bullet X_j - C) > 1$, $\lambda_j = 0$, so that $\lambda_j[Y_j (W \bullet X_j - C) - 1]$ would not affect the minimized objective.

Thus, without loss of generality, we solve

$$min_{W,C,\lambda_i} \quad \tfrac{1}{2} W \bullet W - \sum_{i=1}^{N} \lambda_i [Y_i (W \bullet X_i - C) - 1]$$

since at non-constraining points j, $\lambda_j = 0$. Thus, there are some λ_j 's $= 0$ in the optimal solution. The Lagrangian simplifies to

$$min_{W,C,\lambda_i} \quad \tfrac{1}{2} W \bullet W - \sum_{i=1}^{N} \lambda_i Y_i (W \bullet X_i - C) + \sum_{i=1}^{N} \lambda_i \qquad (4.7)$$

First order conditions (p derivatives of Lagrangian with respect to each element of vector W, and the derivative of Lagrangian w.r.t. C) are:

$$W - \sum_{i=1}^{N} \lambda_i Y_i X_i = 0 \qquad (4.8)$$

$$\sum_{i=1}^{N} \lambda_i Y_i = 0 \qquad (4.9)$$

First order condition w.r.t. the Lagrange multipliers are

$$[Y_i (W \bullet X_i - C) - 1] = 0 \qquad (4.10)$$

for $\lambda_i > 0$.

Optimal vector W is a linear combination $\lambda_i Y_i$ of the training sample points. Substituting for W in Eq. (4.8) into (4.7), we have the dual optimization problem:

$$max_{\lambda_i,C} \quad \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \ Y_i Y_j (X_i \bullet X_j) - \sum_{i=1}^{N} \lambda_i Y_i X_i \bullet \left(\sum_{j=1}^{N} \lambda_j Y_j X_j \right) + C \sum_{i=1}^{N} \lambda_i Y_i + \sum_{i=1}^{N} \lambda_i$$

Using Eq. (4.9), the optimization becomes

$$max_{\lambda_i} \quad \sum_{i=1}^{N} \lambda_i - \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \ Y_i Y_j (X_i \bullet X_j) \qquad (4.11)$$

subject to $\sum_{i=1}^{N} \lambda_i Y_i = 0$ and $\lambda_i \geq 0$. Note that there are N first order conditions (derivatives w.r.t. λ_i) to solve for the N Lagrange multipliers, although strictly speaking, only the few associated with supporting vectors have non-zero λ_i's that need to be solved. Solving Eq. (4.11) (possibly by numerical methods) will yield optimal $\hat{\lambda}_i$'s so optimal $\hat{W} = \sum_{i=1}^{N} \hat{\lambda}_i Y_i X_i$ and $\hat{C} = \hat{W} \bullet X_i - 1$ for $Y_i = +1$ (should get same $\hat{C}$ in

$\hat{C} = \widehat{W} \bullet X_i + 1$ for $Y_i = -1$). Or similarly we can find $\hat{C} = \frac{1}{m} \sum_{k=1}^{m} (\widehat{W} \bullet X_k - Y_k)$ where m is the number of support vectors.

For a test sample point Z, the prediction rule is that it is in class +1 (white circles) if $\widehat{W} \bullet Z - \hat{C} > +1$, and it is in class −1 (black circles) if $\widehat{W} \bullet Z - \hat{C} < -1$.

Eq. (4.11) can be expressed as

$$max_{\lambda_i} \; \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \; Y_i Y_j \; K(X_i, X_j) \tag{4.12}$$

where $K(X_i, X_j) = X_i \bullet X_j$ is called the linear kernel in the SVM. Eq. (4.12) can also be written in matrix form as the following.

$$max_{\lambda_i} \; \Lambda^T L - \frac{1}{2} V^T \left[K(X_i, X_j) \right] V$$

where $\Lambda^T = (\lambda_1, \lambda_2, \ldots, \lambda_N)$, $L^T = (1, 1, \ldots, 1)$, $V^T = (\lambda_1 Y_1, \lambda_2 Y_2, \ldots, \lambda_N Y_N)$, and $\left[K(X_i, X_j) \right]$ is N × N kernel matrix with $K(X_i, X_j)$ as its ijth element.

Consider the situation in Figure 4.3 when it is not possible to draw a line to separate both classes (white and black circles). Suppose a training sample point with $Y_i = +1$ lies "below" $W \bullet X - C = +1$, i.e., in region $W \bullet X - C < +1$, then $Y_j (W \bullet X_j - C) < +1$. Suppose another training sample point with $Y_i = -1$ lies "above" $W \bullet X - C = -1$, i.e., in region $W \bullet X - C > -1$, then $Y_j (W \bullet X_j - C) < +1$. Hence it is not possible to find a constrained optimal solution to Eq. (4.7) where inequality constraint $Y_j (W \bullet X_j - C) \geq 1$ for all j ∈ [1,N] in Eq. (4.6) can be satisfied.

Intuitively, the linear SVM line to try separating the two classes as much as possible (allowing some classification errors in the training sample, e.g., one or more white circles in black circle area and one or more black circles in white circle area) should in some way minimize violations of the constraints $Y_j (W \bullet X_j - C) \geq 1$. Violations would mean

$$Y_j (W \bullet X_j - C) - 1 < 0 \tag{4.13}$$

for some points X_j with Y_j. Let a violation be denoted as $1 - Y_j (W \bullet X_j - C) = \zeta_i$. In general, we can write $Y_i (W \bullet X_i - C) - 1 + \zeta_i = 0$ where $\zeta_i \geq 0$. The violation can also be written as max $(0, 1 - Y_j (W \bullet X_j - C))$ which is also called a hinge loss function.

The maximum widening of "street" while keeping violations low can then be solved as follows.

$$min_{W,C} \; \frac{1}{2} W \bullet W + \alpha \sum_{i=1}^{N} \zeta_i$$

subject to $Y_i (W \bullet X_i - C) - 1 + \zeta_i = 0$ and $\zeta_i \geq 0$ for i=1,2,..,N.

The solution of W, C gives rise to a soft-margin classifier, i.e., a classifier allowing for some misclassifications. Larger $\alpha > 0$ implies tendency towards the hard-margin classifier as the penalty via α is larger for any violation $\zeta_i > 0$, allowing less misclassifications. Lower α, on the other hand, imposes less penalty on misclassifications and allow more misclassifications $\sum_{i=1}^{N} \zeta_i$ but wider margins or smaller $\|W\|$. (Note that wider margins lead to more violations.) α is a regularization hyperparameter that is given or pre-determined, e.g., $\alpha = 1$. α provides some sort of tradeoff between "wider" margins and more violations or "narrower" margins and less violations. "Narrower" margins we saw earlier is less helpful toward predictions on test data or generalized data.

In the minimization problem we can set up the Lagrangian L as

$$min_{W,C} \ \frac{1}{2} W \bullet W \ + \ \alpha \sum_{i=1}^{N} \zeta_i - \sum_{i=1}^{N} \lambda_i \left[Y_i \left(W \bullet X_i - C \right) - 1 + \zeta_i \right] \qquad (4.14)$$

where $\zeta_i \geq 0$ and $\lambda_i \geq 0$ for i=1,2,..,N. First order conditions or derivatives w.r.t. W and C show the following.

$$W - \sum_{i=1}^{N} \lambda_i Y_i X_i = 0 \qquad (4.15)$$

$$\sum_{i=1}^{N} \lambda_i Y_i = 0 \qquad (4.16)$$

Expression (4.14) can also be written as

$$min_{W,C} \ \frac{1}{2} W \bullet W \ - \ \sum_{i=1}^{N} \lambda_i Y_i \left(W \bullet X_i - C \right) + \sum_{i=1}^{N} \lambda_i + \sum_{i=1}^{N} (\alpha - \lambda_i) \zeta_i \qquad (4.17)$$

This is somewhat similar to expression (4.7) except it now has an extra set of terms $+ \sum_{i=1}^{N} (\alpha - \lambda_i) \zeta_i$. When $\zeta_i > 0$, the objective function to be minimized, ceteris paribus must increase, so $(\alpha - \lambda_i) \geq 0$ or $\lambda_i \leq \alpha$. Hence, $0 \leq \lambda_i \leq \alpha$.

Applying Eqs. (4.15), (4.16) in (4.14), we obtain the dual problem

$$max_{\lambda_i} \ \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \ Y_i Y_j \left(X_i \bullet X_j \right) \qquad (4.18)$$

subject to $\sum_{i=1}^{N} \lambda_i Y_i = 0$ and $0 \leq \lambda_i \leq \alpha$. Eq. (4.18) can also be written in the form

$$max_{\lambda_i} \ \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \ Y_i Y_j \ K \left(X_i, X_j \right) \qquad (4.19)$$

for a given kernel $K(\cdot, \cdot)$.

The SVM method does not explicitly compute the probability of a test point Z being in one class/type or another. However, we can use a logit model to estimate the probability given the predicted class being +1 or −1 against the features used in SVM for the prediction. The predicted probabilities can then be used to build the ROC curve given different probability thresholds.

Other SVM kernels that can be used for classification include the following. The polynomial kernel $K(X_i, X_j) = (\gamma X_i \bullet X_j + c)^d$ where the degree of the polynomial d, gamma γ, and c are hyperparameters. Sometimes a simpler version of $\gamma = 1$ or $1/N$ is used as a default case in many programs. As an illustration, suppose point X_i is $(x_1^{(i)}, x_2^{(i)})$ with two dimensions. Let $K(X_i, X_j) = (X_i \bullet X_j + 1)^2$ where d = 2. Then $K(X_i, X_j)$ = $(x_1^{(i)} x_1^{(j)} + x_2^{(i)} x_2^{(j)})^2 + 1 = 2 x_1^{(i)} x_1^{(j)} + 2 x_2^{(i)} x_2^{(j)} + 2 x_1^{(i)} x_1^{(j)} x_2^{(i)} x_2^{(j)} + (x_1^{(i)} x_1^{(j)})^2 + (x_2^{(i)} x_2^{(j)})^2 + 1 = (\sqrt{2} x_1^{(i)}, (\sqrt{2} x_2^{(i)}, (\sqrt{2} x_1^{(i)} x_2^{(i)}, x_1^{(i) 2}, x_2^{(i) 2}, 1) \bullet (\sqrt{2} x_1^{(j)}, \sqrt{2} x_2^{(j)}, \sqrt{2} x_1^{(j)} x_2^{(j)}, x_1^{(j) 2}, x_2^{(j) 2}, 1)$. Clearly, the polynomial kernel now expands X_i from two to six dimensions. The higher dimension allows for easier linear separation; this is called the "kernel trick".

We can let the dimension expansion (enlarging feature space) be transformation function $\phi (x_1^{(i)}, x_2^{(i)}) = (\sqrt{2} x_1^{(i)}, \sqrt{2} x_2^{(i)}, \sqrt{2} x_1^{(i)} x_2^{(i)}, x_1^{(i) 2}, x_2^{(i) 2}, 1)$. Hence polynomial kernel on $(x_1^{(i)}, x_2^{(i)})$ becomes linear kernel on $(\phi(x_1^{(i)}, x_2^{(i)}), \phi(x_1^{(j)}, x_2^{(j)}))$, i.e., polynomial $K(X_i, X_j) = \phi(x_1^{(i)}, x_2^{(i)}) \bullet \phi(x_1^{(j)}, x_2^{(j)}) = \phi(X_i) \bullet \phi(X_j)$. Without elaborating on some highly technical details, we state that there is a Mercer's Theorem[4]

---

[4] Mercer, J. (1909), "Functions of positive and negative type and their connection with the theory of integral equations", *Philosophical Transactions of the Royal Society A*, **209** (441–458): 415–446.

showing that for symmetrical real-valued $K(X_i,X_j)$, a representation $K(X_i,X_j) = \phi(X_i) \bullet \phi(X_j)$ for function $\phi(.)$ can be found.

Using a non-linear kernel $K(X_i,X_j)$, we are de facto trying to solve for a (linear) separating hyperplane margin in

$$max_{\lambda_i} \ \sum_{i=1}^{N} \lambda_i - \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \ Y_i Y_j \big(\phi(X_i) \bullet \phi(X_j)\big)$$

$$\equiv \ max_{\lambda_i} \ \sum_{i=1}^{N} \lambda_i - \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \ Y_i Y_j \ K\big(X_i, X_j\big) \tag{4.20}$$

for some function $\phi(.)$ associated with the non-linear kernel.

An example of popular non-linear kernels are the Gaussian radial basis function (RBF) $K(X_i,X_j) = \exp(-\gamma \ \|X_i - X_j\|^2)$ or $\exp(-\gamma \ (X_i - X_j) \bullet (X_i - X_j))$ where $\gamma = 1/(2\sigma^2)$. Here γ (gamma) > 0 is a hyperparameter and σ need not be constrained to be standard deviation of the standardized features. In Sklearn, by default, γ = 'score' which is $1/(N\sigma^2)$. Another example would be to set γ to a positive number. The Gaussian RBF $K(X_i,X_j) = \exp(-\gamma \ (X_i - X_j) \bullet (X_i - X_j)) = \exp(-2\gamma) \exp(2\gamma \ X_i \bullet X_j)$ and hence can be written in the form involving exponential of $X_i \bullet X_j$.

Unlike the linear SVM hyperparameter α that determines how much misclassification would be tolerated, high γ means more curvature in the separating curved surface of the two classes. In linear SVM, this separating curved surface is a hyperplane and thus here γ has no effect and is by default a constant such as $\gamma = 1$ or $\gamma = 1/N$. High gamma like high α leads in general to more accuracy in classification but perhaps overfitting and more bias. Notice that if we expand the exponential function into an infinite Taylor series, then the RBF is like expanding X_i into an infinite number of dimensions involving $\|X_i - X_j\|^4$, $\|X_i - X_j\|^6$, $\|X_i - X_j\|^8$, and so on. Hence the RBF is often able to perform well on highly nonlinear data points as it can project them to a higher number of dimensions.

Another non-linear kernel is the sigmoid or hyperbolic tangent kernel $K(X_i,X_j) = \tanh(\gamma X_i \bullet X_j + c)$ where gamma γ and c are hyperparameters. Sometimes a simpler version of $\gamma = 1$ or $\gamma = 1/N$ is used as a default case in many programs. This kernel form is popular in neural networks but is generally not as effective in SVM.

In general, when we apply a particular kernel function as in the constrained minimization in Eq. (4.14), we check for the performance results in prediction accuracy and the classification errors on the test sample data set. There is no need to explicitly determine or check the transformation function of $\phi(.)$ in $K(X_i,X_j) = \phi(X_i) \bullet \phi(X_j)$.

The choice of what type of kernel depends on the nature of the data and the problem. Linear kernel is used (typically the one with soft constraint and applying hyperparameter α, or 'C', in sklearn) when the data is approximately linearly separable. Polynomial kernel is used when the data has a curved border. Gaussian kernel is used when the data appear to have complicated overlaps and no clear boundaries.

For SVM in classification prediction, the training data set is used to estimate optimally the separating hyperplane parameters (including kernel parameters) and associated hyperparameter such as alpha. Then a validation set can be used to fine-tune the hyperparameters by seeking best performance metrics. After that, the optimized algorithm can be applied to the test set to check performance metrics for robustness.

4.4 Support Vector Regression

When the SVM method is used for regression in making numerical prediction, the method is modified to an algorithm called Support Vector Regression (SVR). Whereas the objective in linear regression estimation is to minimize the errors (via metric such as mean squared distances) between the predicted hyperplane and the targets of the training data points, the objective in SVR is to find the estimated hyperplane $Y_j = W \bullet X_j - C$ such that the errors $\widehat{Y}_j - Y_j$ should as far as possible be smaller than or equal to a pre-determined hyperparameter, threshold epsilon $\varepsilon > 0$. Within this epsilon space or tube, no penalty is associated with the error or deviation from the fitted hyperplane.

Just as in SVM, the SVR finds a maximum margin (equivalently minimum $\frac{1}{2}\|W\|^2$) where the margin hyperplanes have vertical displacements $\varepsilon > 0$ on either side of the fitted hyperplane in the middle between the two margin hyperplanes. As seen in Figure 4.8 below (where for illustration a simple two-dimension case is shown), there are two possible fitted lines l_1 and l_2. Each has two margin hyperplanes with vertical displacements ε on either side. However, line l_2 has a wider margin than line l_1, i.e., $m_2 > m_1$. Hence, maximizing margin subject to displacements of epsilon $\varepsilon > 0$ is tantamount to finding a "flatter" fitted hyperplane.

However, as with the case of imperfect separation in SVM classification, there may be points lying outside the epsilon space or tube. These outside points (see * points in Figure 4.9 below) are penalized with a cost α for the error or deviation ζ_i beyond the margin hyperplanes. Thus, if line l_2 is fitted, the added cost is $\alpha \sum_{i=1}^{3} \zeta_i$ in this case. The idea in finding the optimal fitted hyperplane is in

$$min_{W,C,\zeta_i} \ \frac{1}{2} W \bullet W \ + \ \alpha \ \sum_{i=1}^{N} \zeta_i \tag{4.21}$$

subject to $| Y_i - (W \bullet X_j - C) | \le \varepsilon$ and $\zeta_i \ge 0$ for i=1,2,..,N. Note that in SVR, unlike SVM, Y_i is the numerical target value, not the class/type dummy variable of +1 and −1.
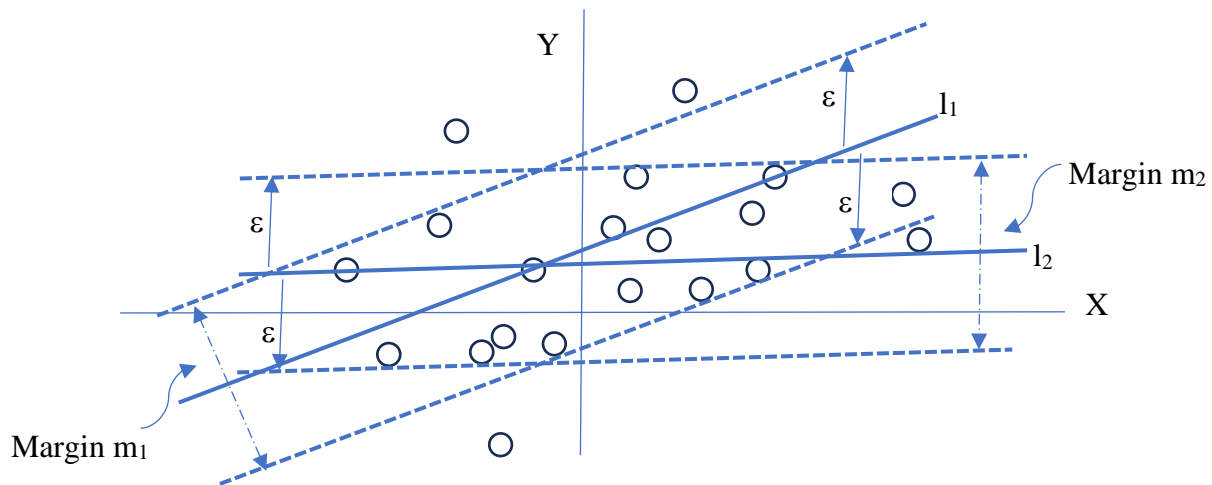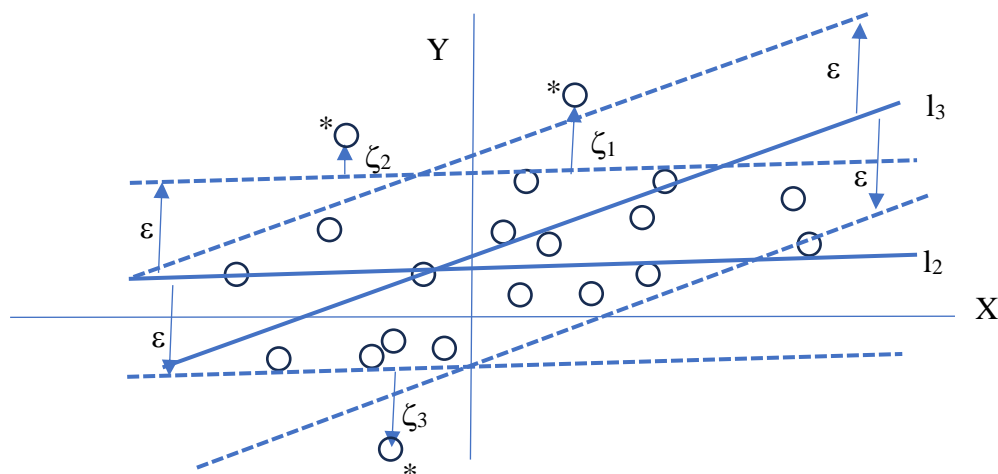

Figure 4.8

Figure 4.9

Note that in Figure 4.9, a better fit with a lower objective function in Eq. (4.21) can be found by increasing/ tuning hyperparameter α (or 'C', in sklearn) so that ζ_i deviations are penalized more and thus are kept smaller with tilting the slope of the fitted line to l_3. In l_3, sum of ζ_i deviations is smaller. Increasing epsilon ε would reduce sum of ζ_i deviations but when the increase is too large and no deviations exist, the fitted slope becomes flat or zero slope. Sometimes increasing epsilon ε may cause prediction to deteriorate if catching too many points in the tube including outliers loses the slope pattern of the majority of the points without the outliers. In this latter case, there is usually poor prediction on test data. When epsilon shrinks toward zero, the SVR approaches a linear regression. In sklearn, the default epsilon is 0.1 and the default α (or 'C") is 1.0.

Just as in SVM, the dual problem of finding the optimal fitted hyperplane in SVR can be formulated in terms of $(X_i \bullet X_j)$ as in Eq. (4.18). Nonlinear kernels $K(X_i,X_j) = \phi(X_i) \bullet \phi(X_j)$ as in Eq. (4.20) can also be used in SVR to obtain fitted hyperplane after the original data points are suitably transformed via $\phi(.)$.

4.5 Worked Examples – Data

The data set is found in public website: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud. It was collected by Worldline in collaboration with the Machine Learning Group of Université Libre de Bruxelles. The dataset shows European credit cardholders' 284,807 transactions over two days in September 2013, out of which there were 492 fraudulent cases, accounting for about 0.17%[5] of the transactions. Credit card fraudulent transactions occur when unauthorized person(s) somehow gain access to use the card to make purchases.

According to the source, the data are numerical continuous variable features that are a PCA transformation from the original confidential client background information. There are 28 such features. In

---

[5] Due to the huge type imbalance, it may be misleading just to consider Confusion Matrix accuracy, the AUC should also be considered.

addition to these, two actual variables were added: "Time" variable indicates time in seconds that elapsed between each transaction and the first transaction in the dataset and "Amount" indicates dollar transaction amount. The target or class/type variable "Type" is a dummy variable that takes the value 1 if the case is fraudulent, and 0 otherwise.[6]

The "Time" variable is not included in the study as it does not appear to be a useful feature. The 28 PCA-transformed variables and the "Amount" are standardized via the StandardScaler and these 29 features are used to predict Type 1 (Fraud) or Type 0 (no Fraud). The prediction results are run in Chapter4-1.ipynb using Naïve Bayes, k-Nearest Neighbor, Support Vector Machines, and also the Logistic Regression algorithms. The main code lines are shown as follows.

```
In [1]:    ### Data from https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
           ### Credit Card Fraud Prediction/Detection
           import pandas as pd
           import numpy as np
In [2]:    dataX=pd.read_csv('creditcard.csv',header=0)
           dataX=dataX.dropna() # Remove missing values or NA
           print(dataX.shape)
           print(list(dataX.columns))

Output:  (284807, 31)
         ['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', '
         V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Type']
```

There are 284,807 records/cases and 31 fields. Columns 2 to 30 will be used as features, while column 31 provides the target variable values or types. 'Type' = 1 indicates fraud case while 'Type' = 0 indicates no-fraud case. In [4], the percentage of fraud case is found to be very small at about 0.173%. The data classifications are imbalanced. To mitigate the imbalance problem that can produce high accuracy in prediction without any algorithm by just always picking the class that has very high proportion, one way is to create a more balanced sample by deleting sample points with the higher proportion of the same targets/classes. Another way is to augment (using resampling schemes) sample points with the lower proportion of the same targets/classes. Here we randomly select 492 sample points from the high density cases with Type 0 to match with the existing 492 sample points from the low density cases with Type 1.

```
In [4]:    data=dataX.groupby("Type").sample(n=492, random_state=1)

           ### creates 492 or n number of each of Type 1 and 0
```

Now there are a total of 984 cases with 50% of the cases involving credit card frauds. The features are normalized (with mean 0 and standard deviation 1) in code line [10] as part of the data preprocessing. Having data of similar scale is important for some machine learning algorithm that are affected by relative magnitudes of features.

```
In [10]:   # Normalization of all features

           from sklearn.preprocessing import StandardScaler #from sklearn import preprocessing done earlier
           scaler = StandardScaler().fit(data1)  # this computes means and sd to standardize
           data2 = scaler.transform(data1)       # this does the standardization with computed means, sds
           data3=pd.DataFrame(data2,index=data1.index,columns=data1.columns)
           # above step converts array from last step back to pandas dataframe, also puts back indexes V's
```

---

[6] A similar study by Suraj Patil, V. Nemade, and P.K. Soni, (2018), "Predictive Modelling For Credit Card Fraud Detection Using Data Analytics", Procedia Computer Science 132, pp.385-395, indicated client's checking account history, pattern of credit card payments, abnormal use and amounts of credit card use, and client's possession of property, car, and insurance could have predictive influence on the probability of a fraud.

Next, the data is split into 80% (787 cases) for training and remaining 20% (197 cases) for testing. We skip validation and assume the test result is done using optimized hyperparameter(s), and that the performance would be similar to using a subset for validation.

Naïve Bayes Algorithm

Code line [15] shows the application of the sklearn Naïve Bayes algorithm to fit the training set data, then perform the prediction on the test data (X_test). The accuracy is then reported as 91.878% using the sklearn metrics on accuracy_score.

```
In [15]: # Train the model
         gnb = GaussianNB()
         gnb.fit(X_train, y_train)

         # Evaluate the model
         y_pred_NB = gnb.predict(X_test)

         from sklearn import metrics
         from sklearn.metrics import accuracy_score
         Accuracy_NB = metrics.accuracy_score(y_test, y_pred_NB)
         print("Naive Bayes Accuracy:",Accuracy_NB)

         Naive Bayes Accuracy: 0.9187817258883249
```

The confusion matrix, the classification report, and the ROC curve are obtained via code lines [16], [17], [18], [19]. ROC area is 96.075%.

```
In [16]: from sklearn.metrics import confusion_matrix
         confusion_matrix = confusion_matrix(y_test, y_pred_NB)
         print(confusion_matrix)

         [[98  5]
          [11 83]]
```

```
In [17]: from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred_NB))

                       precision    recall  f1-score   support

                    0       0.90      0.95      0.92       103
                    1       0.94      0.88      0.91        94

             accuracy                           0.92       197
            macro avg       0.92      0.92      0.92       197
         weighted avg       0.92      0.92      0.92       197
```
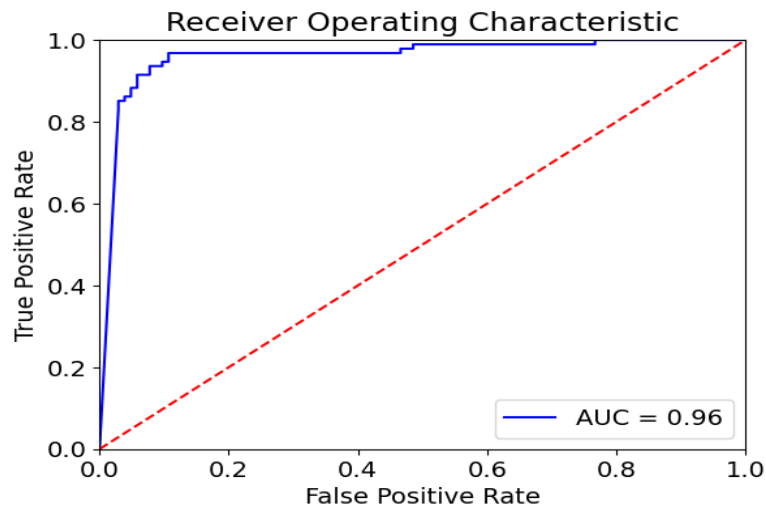
K-Nearest Neighbor Algorithm

Code lines [20], [21], [22] show the application of the sklearn KNeighborsClassifier employing k = 3 to fit the training set data, then perform the prediction on the test data (X_test). The accuracy is then reported as 92.893% using the sklearn metrics on accuracy_score.

```
In [20]: from sklearn.neighbors import KNeighborsClassifier
         kNN= KNeighborsClassifier(n_neighbors = 3, algorithm='auto', metric='minkowski', p=2,
             metric_params=None, n_jobs=None, weights='uniform')
         # argument: Minkowski (default metric) and p=2 gives Euclidean distance
         # default weight is uniform
         # algorithm auto uses 'best' search method to solve min distances
         # n_jobs (default none means 1): number of parallel processes to solve
         kNN.fit(X_train, y_train)
         ## Note the kNN.predict process next involves intense computations

Out[20]: KNeighborsClassifier(n_neighbors=3)

In [21]: y_pred_kNN = kNN.predict(X_test)

In [22]: from sklearn import metrics
         from sklearn.metrics import accuracy_score
         Accuracy_kNN = metrics.accuracy_score(y_test, y_pred_kNN)
         print(Accuracy_kNN)

         0.9289340101522843
```

Code lines [23], [24] show that the training score is higher than the test score which is to be expected since the testing is done on generalized (out-of-sample) data. Both training fitting accuracy score and testing accuracy score are close so there is no overfitting in the training.

```
In [23]: print('Test set score: {:.16f}'.format(kNN.score(X_test, y_test)))
         ## same computation as combined two steps above

         Test set score: 0.9289340101522843

In [24]: # Compare with training score
         print('Training set score: {:.16f}'.format(kNN.score(X_train, y_train)))

         Training set score: 0.9504447268106735
```

The confusion matrix, the classification report, and the ROC curve are obtained via code lines [26], [27], [28], [33], and [34].

```
In [26]: from sklearn.metrics import confusion_matrix
         confusion_matrix = confusion_matrix(y_test, y_pred_kNN)
         print(confusion_matrix)

         [[99  4]
          [10 84]]
```

```
In [27]: from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred_kNN))

                       precision    recall  f1-score   support

                    0       0.91      0.96      0.93       103
                    1       0.95      0.89      0.92        94

             accuracy                           0.93       197
            macro avg       0.93      0.93      0.93       197
         weighted avg       0.93      0.93      0.93       197
```

```
In [28]: Prob1=kNN.predict_proba(X_test)[:,0]
         ### gives the probabilities for target variable being 0
```
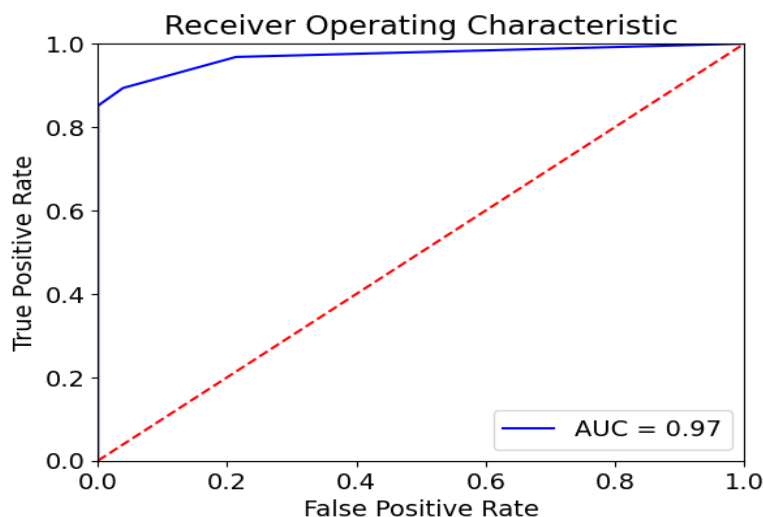

The area under ROC for k=3 is 97.041%, about 1% higher than that of Gaussian Naïve Bayes method. When k is increased to k=5, the area increases marginally to 97.707%, after which this area decreases when k is further increased. The accuracy also increases to 94.923% when k=5, after which it decreases as k increases. We assume that k=5 would have been the optimal hyperparameter to be used if we had used a validation subset.

Support Vector Machine

Code line [57] shows the application of the sklearn SVM employing the linear kernel with hyperparameter $\alpha = 1.0$ ("C" in the ipynb file) to fit the training set data. These hyperparameters are assumed to be optimal from a validation. Then the model is for prediction on the X_test. Code lines [58], [59], [60] shows the confusion matrix, accuracy, and classification report. The accuracy is reported as 96.447%.

Linear Kernel

```
In [57]: from sklearn import svm
         svm1 = svm.SVC(kernel='linear', C = 1.0, probability=True)
         ### C is regularization para. Default C=1.
         svm1.fit(X_train,y_train)
         y_pred_svm = svm1.predict(X_test)
```

```
In [58]: from sklearn.metrics import accuracy_score,confusion_matrix
         confusion_matrix(y_test,y_pred_svm)
```

```
Out[58]: array([[102,   1],
                [  6,  88]], dtype=int64)
```

```
In [59]: accuracy_score(y_test,y_pred_svm)
```

```
Out[59]: 0.9644670050761421
```

```
In [60]: from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred_svm))
```

```
                 precision    recall  f1-score   support

             0       0.94      0.99      0.97       103
             1       0.99      0.94      0.96        94

      accuracy                           0.96       197
     macro avg       0.97      0.96      0.96       197
  weighted avg       0.97      0.96      0.96       197
```

Code lines [61], [62] yield the ROC area and graph. The area is 98.657% which is the higher than those of the Naïve Bayes and the kNN methods.
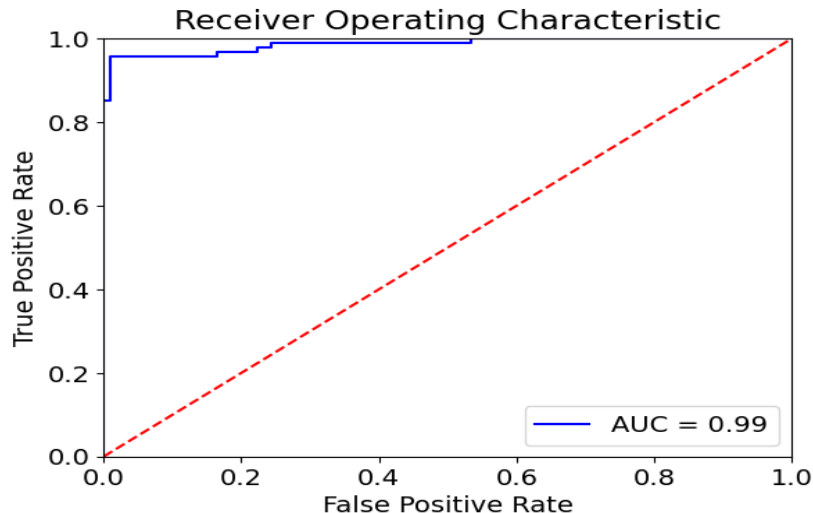
Employing other kernels such polynomial, sigmoid, and rbf in the SVM produced lower accuracies and lower area under ROC curves.

Support Vector Regression

The exercise here is contained in Chapter4-2.ipynb and uses the data set 'california_housing.frame.csv' seen in Chapter 2. As seen in code line [8], after removing outliers and censored data, there are 19615 rows with 8 columns of features and 1 column of target variable 'MedHouseval' (median house value in block in USD).

In code line [17], the features are pre-processed and scaled to within (0,1). In code line [18], the data set is randomly split into 80% or 15,692 training data points and 3,923 test data points. In code lines [20], [21], linear kernel SVR (epsilon = 0.56, α or 'C' = 25) shows a R^2 training score of 61.97% and R^2 test score of 62.14%.

```
In [20]: SVR1 = SVR(kernel = 'linear',epsilon=0.56,C=25)
         SVR1.fit(X_train, y_train)
         y_pred_SVR1_train = SVR1.predict(X_train)
         y_pred_SVR1_test = SVR1.predict(X_test)
```

```
In [21]: r2_score_SVR1_train = r2_score(y_train, y_pred_SVR1_train)
         print('R2_score (train): ', r2_score_SVR1_train)
         ### R2_score (train) is the R-square in the linear regression involving only the training data set
         r2_score_SVR1_test = r2_score(y_test, y_pred_SVR1_test)
         print('R2_score (test): ', r2_score_SVR1_test)

         R2_score (train):  0.6197322462567303
         R2_score (test):  0.6214216496582796
```

Code lines [24] through [27] show the results of other nonlinear kernels. The best result is shown in code lines [26], [27] with the 'rbf' kernel.

```
In [26]: SVR3 = SVR(kernel = 'rbf',epsilon=0.56,C=25)
         SVR3.fit(X_train, y_train)
         y_pred_SVR3_train = SVR3.predict(X_train)
         y_pred_SVR3_test = SVR3.predict(X_test)
```

```
In [27]: r2_score_SVR3_train = r2_score(y_train, y_pred_SVR3_train)
         print('R2_score (train): ', r2_score_SVR3_train)
         ### R2_score (train) is the R-square in the linear regression involving only the training data set
         r2_score_SVR3_test = r2_score(y_test, y_pred_SVR3_test)
         print('R2_score (test): ', r2_score_SVR3_test)

         R2_score (train):  0.7286284928673199
         R2_score (test):  0.7356554348857722
```

R^2 test score is increased to 73.57%. The predicted target versus actual target value plots for the linear kernel and the more effective nonlinear 'rbf' kernel are shown below.

Linear Kernel   Non-linear 'rbf' Kernel
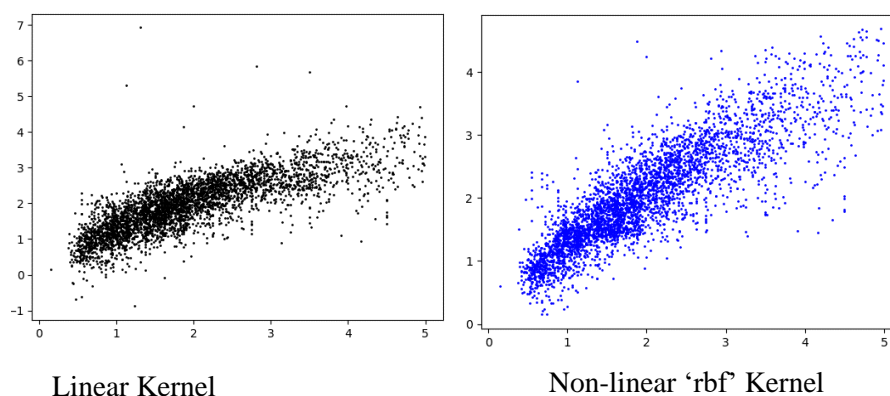
4.6 Summary

Naïve Bayes method is popular as it has lower computational time and the method scales up effectively with multi-class predictions as with binary predictions. It can also work with many features/dimensions. However, its independence assumption may cause issues in classification if the features are highly

correlated. A common mistake is not to ensure that all features present in the testing sample set are also used for fitting in the training data set; use of such a feature in the test set would render it a 'zero frequency' by which it has no part to play in the prediction of the test sample point. Numerical prediction using Naïve Bayes is not a natural concoction and may produce ineffective predictors.

kNN has lazy training as active computations take place only when a test sample point is introduced and then the nearest neighbors are searched within the training data set. Each new test sample point gives rise to new constructions of neighborhood and the resulting prediction. The method is completely non-parametric with no distributional assumptions. It also extends readily to multi-class problems and can be used for both classification and numerical prediction.

However, kNN has drawbacks in some problems. Firstly, since the method uses a distance measure such as Euclidean distance, it is sensitive to scale so that features with outsized or else miniature scale should be standardized or normalized. The algorithm is slow when the dimension or number of features increases as the neighborhood computations become large. In the same vein, tuning the k hyperparameter also involves intense computations when the dimension is large. Imbalanced data may cause bias due to biased neighborhood. It also pays much less attention to outliers not in the neighborhood.

Contrary to kNN, SVM can easily handle many features. The number of features can also be larger than the number of training data points. The ability to use different specifications of kernel functions provides good possibility of finding separating boundaries or separating hyperplanes after kernel transformation of the original data points. The transformation often expands the feature dimension. This can perform reasonably well in complex data patterns.

However, SVM may require high computational time for a large data set. Overfitting often occurs especially when the number of features is large relative to the size of the data set. Regularization is thus important in SVM. SVM may also be sensitive to outliers. SVM may be extendible to multi-class separation although the methods typically involve multiple binary classifications.

4.6 Other References

Alex J. Smola, and B. Schölkopf, (2004), "A tutorial on support vector regression", Statistics and Computing, 14, 199—222.

Eibe Frank, Leonard Trigg, Geoffrey Holmes, and Ian H. Witten, "Technical Note: Naïve Bayes for Regression", Machine Learning, Kluwer Publishers Boston, 1999.

Lipo Wang, "Support Vector Machines: Theory and Applications," ed., Springer, Verlag, 2005.