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 that 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.

Consider 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.

Let Y represent the event that a blue ball is drawn. 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 $P(\{Z=1\} \cap Y) = 10/100$ and P(Y) = 30/100.

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 $P(Y) = P(Y\cap Z) + P(Y\cap Z^c)$. 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 positive, and percentage of non-Z and tested positive (false positive).

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. This illustration here is based on single words; advanced sentiment analysis dives into phrases, sentences, and texts. However, the approach is similar.

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,

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

Using the simplifying assumption on the last line,

$$P(F_1, F_2, \ldots, F_n | Z) = P(F_1|Z) \times P(F_2|Z) \times P(F_3|Z) \times P(F_4|Z) \times P(F_5|Z) \times \ldots\ldots \times P(F_{n-1}| Z) \times P(F_n |Z) = \prod_{j=1}^{n} P\left(F_j|Z\right)$$

Then,

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

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 $\hat{P}\left(F_j|Z = 1\right)$ (percentage of days of occurrences of F_j when next day stock price index increases) and $\hat{P}\left(F_j|Z = 0\right)$ (percentage of days 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 ,…, F_m} ($m \leq n$) 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

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

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

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

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 or cases in the training sample. These are unconditional probability estimates. These estimates add to one. There are several methods for estimating $\hat{P}\left(F_j|Z = 1\right)$ and $\hat{P}\left(F_j|Z = 0\right)$ 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 $P\left(F_j|Z\right)$ 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.

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. 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 number of days of occurrence of F_j /total number of days 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 days, 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 ≤ 1) is added in the estimation of $\hat{P}(F_j\ |Z=1)$ and $\hat{P}(F_j\ |Z=0)$. For example, $\hat{P}(F_j = \text{"boom"}\ |Z=0)$ = $(1+\alpha)/(200+\alpha n)$ where n is the number of features. This adjustment can be non-negligible when n is small.

In the general execution of Eq. (4.2), we assume that the training set is large enough so that $\hat{P}(F_j|Z) > 0$ for every possible F_j (in the dictionary) in each of condition Z = 1 or Z = 0. We also assume that in the test data set (and foreseeable generalized data set), observed Y** for one day or observed dictionary mood words for that day consisting of F_5, F_7 , F_8,…, F_m for example have computed $\hat{P}(F_5\ |Z=1)$, $\hat{P}(F_7\ |Z=1)$, $\hat{P}(F_8\ |Z=1)$,…, $\hat{P}(F_m\ |Z=1)$, and also $\hat{P}(F_5\ |Z=0)$, $\hat{P}(F_7\ |Z=0)$, $\hat{P}(F_8\ |Z=0)$,…, $\hat{P}(F_m\ |Z=0)$ obtained from the training set computations. $\hat{P}(Z=1)$ and $\hat{P}(Z=0)$ are also obtained from the training data set. Then (4.2) can be used to predict the test set Z (whether 1 or 0) based on these test set features Y**.

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. The information Y contains not $\{F_1, F_2, \ldots, F_n\}$, but news - $\{F_1, F_2, \ldots, F_n\}$, analysts' reports - $\{F_1, F_2, \ldots, F_n\}$, social media - $\{F_1, F_2, \ldots, F_n\}$, i.e., including sources and their mood words. We could use instead news-F_j, analysts'-F_j, social media-F_j, for all j, as separate complex features. In this way, the same mood word F_j occurring in different sources are treated as being different pieces of information. We can extend Eq. (4.1) to

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

For estimating $\hat{P}(\text{news} - F_j|Z=1)$, we can use their frequency counts of number of days of occurrence of news-F_j /total number of days 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 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 , j=1,2,…,n, 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 (associated lagged features $\{G_j\}$) is

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

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 (using all of sample data) 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) for example when we are dealing with time-sequenced data, it may be more robust to first standardize the training and the test data features separately 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):

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

where Y ≡ {$F_1, F_2, \dots, F_u, G_1, G_2, \dots, G_v$}. 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

$$\arg\max_{Z=1,0} P(\{Z\}|F_1, F_2, \dots, F_u, G_1, G_2, \dots, G_V)$$
$$= \arg\max_{Z=1,0} \prod_{j=1}^{u} \hat{P}(F_j|Z) \prod_{j=1}^{v} \frac{1}{\sqrt{2\pi\hat{\sigma}_{zj}^2}} \exp\left(-\frac{1}{2}\left[\frac{G_j - \hat{\mu}_{zj}}{\hat{\sigma}_{zj}}\right]^2\right) \times \hat{P}(Z)$$

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} 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), and (V_d,L_d,I_d). 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. The feature is used to predict the outcome Z = 1 or Z = 0 the following trading day.

Under condition of Z=1 (next day index rises), the probabilities of combined events (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), are estimated by frequency out of the total of all combined events in the training data, i.e., each with probability $\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3, \hat{\theta}_4, \hat{\theta}_5, \hat{\theta}_6, \hat{\theta}_7, \hat{\theta}_8$, where $\sum_{i=1}^{8} \hat{\theta}_i = 1$. Similarly, under condition of Z = 0 (next day index falls or maintains), the probabilities of combined events (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), are estimated by frequency out of the total of all combined events in the training data, i.e., each with probability $\hat{\gamma}_1, \hat{\gamma}_2, \hat{\gamma}_3, \hat{\gamma}_4, \hat{\gamma}_5, \hat{\gamma}_6, \hat{\gamma}_7, \hat{\gamma}_8$, where $\sum_{i=1}^{8} \hat{\gamma}_i = 1$. Assume these combined events are independent of each other and the session in which they occurred.

Suppose for a test sample point (day), the combined event of E = {(V_u,L_u,I_u), (V_u,L_u,I_u), (V_d,L_u,I_u)} are observed in the previous day sessions. The multinomial probability of this happening conditioned on Z = 1 the next day is

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

where $x_1 + x_2 + \ldots + x_8 = n$. Here, n = 3, $x_1 = 2$, $x_5 = 1$, and the remaining x_j's are zeros. The multinomial probability of this happening conditioned on Z = 0 the next day is

$$\text{P(E | Z = 0)} = n! \times \left[\frac{\left(\hat{\gamma}_1^{\,x_1}\,\hat{\gamma}_2^{\,x_2}\,\hat{\gamma}_3^{\,x_3}\,\hat{\gamma}_4^{\,x_4}\,\hat{\gamma}_5^{\,x_5}\,\hat{\gamma}_6^{\,x_6}\,\hat{\gamma}_7^{\,x_7}\,\hat{\gamma}_8^{\,x_8}\right)}{x_1!x_2!x_3!x_4!x_5!x_6!x_7!x_8!}\right].$$

The predictor of Z becomes

$$\arg\max_{Z=1,0} P(\{Z\}|E) = \arg\max_{Z=1,0} [P(E|Z)] \times \hat{P}(Z) \tag{4.4}$$

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, using the trained $\hat{\theta}_j$, $\hat{\gamma}_j$ for j =1,2,…,8 and also $\hat{P}(Z)$ from the training set data.

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. In GNB, the data set possibly 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 estimated parameters are then applied to predict cases from the test features. If there are hyperparameters, such as the number of categories, we assume validation has been done and that we are working on a re-grouped training set using the optimal hyperparameters and applying the trained or estimated model to perform predictions on the test data set. 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^{**}. We may need to also estimate probability of the features. 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.

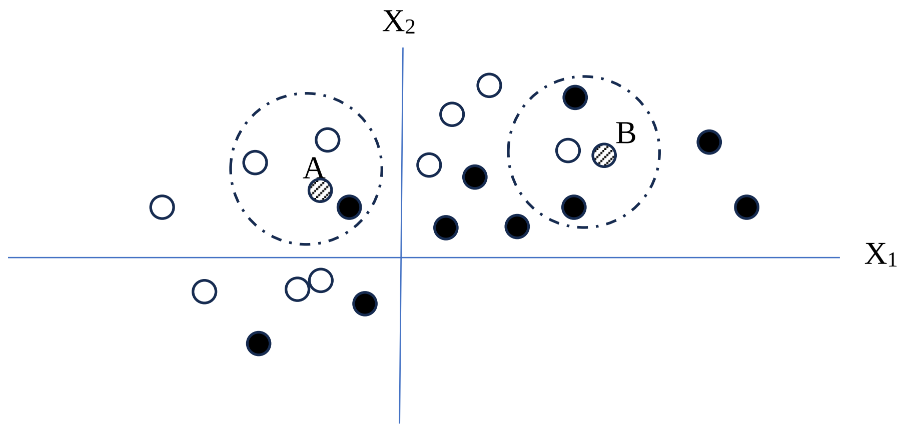


Figure 4.1

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]). This transformation is done over the whole sample for cross-sectional data. 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

$$\sqrt{\sum_{k=1}^{2}(X_{uk} - X_{vk})^2}$$

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 $\sum_{k=1}^{2}|X_{uk} - X_{vk}|$, the Minkowski distance $\left(\sum_{k=1}^{2}|X_{uk} - X_{vk}|^p\right)^{1/p}$ for some $p \geq 1$, 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 so that it can flip sides. 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.

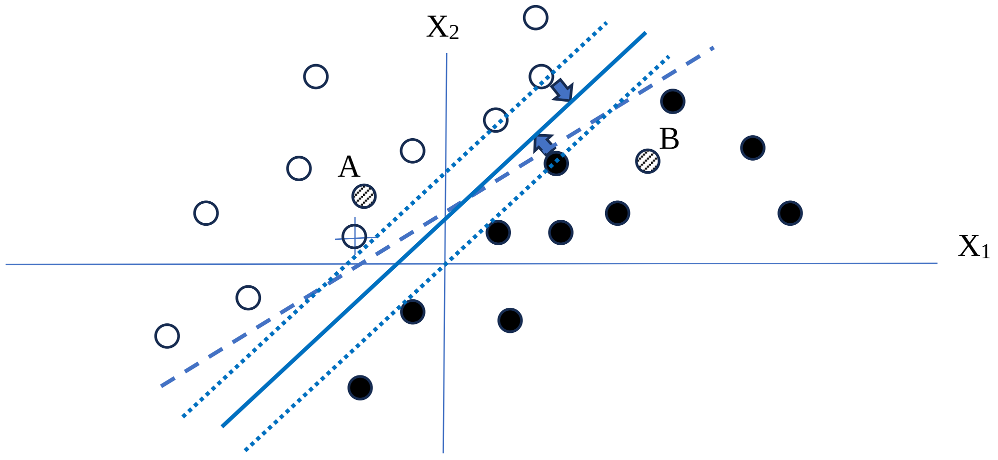


Figure 4.2

[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.

Suppose we try to predict the targets/labels/classes of the cases or instances 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 (linear or nonlinear) 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.

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. The linear SVM provides for a solid line boundary with the maximum margin (sum of perpendicular distances from the nearest feature points on either class to the line). Any shift in the boundary (see dashed line) would end up having at least one colored circle being closer to the boundary, e.g., the white circle shown with a cross. The latter means that the margin or gap has shrunk and is not an optimal boundary.

In general, this boundary is called the separating hyperplane or optimal decision plane (to separate the two classes). It separates the two classes as far as is possible. The separating hyperplane comes with two parallel hyperplanes ("margin" hyperplanes) on either side (denoted by the dotted lines) that touch the closest points (from the boundary) of the two classes. These 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. To recall, the perpendicular gap or distance between the dotted lines, or similarly, twice the distance between a supporting vector and the boundary line is called the margin.

Once the SVM has fitted the training data set, i.e., determined the boundary to separate the classes in the training data set, and supposing validation was done to fine-tune the hyperparameters, 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.

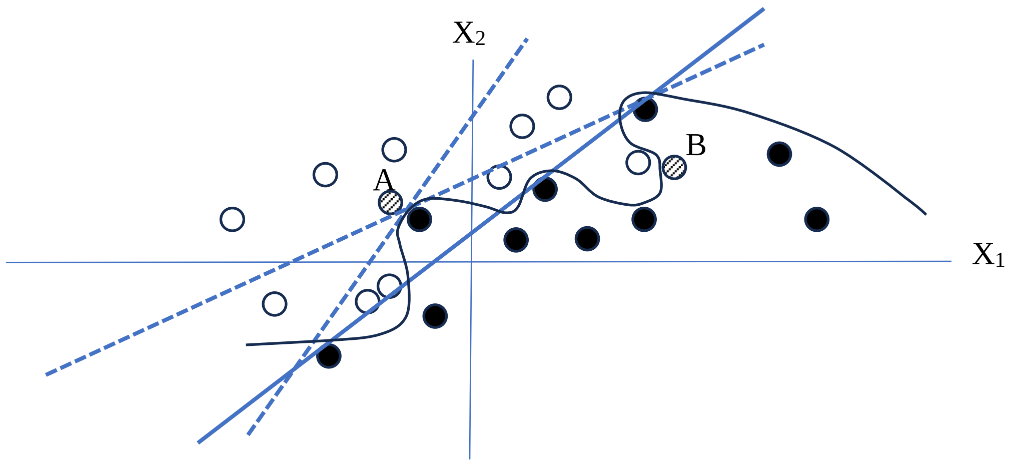


Figure 4.3

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 of a decision boundary, a test data sample point t's features are subject to the transformation so that $\phi(X_1^{(t)}, X_2^{(t)})$ 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".

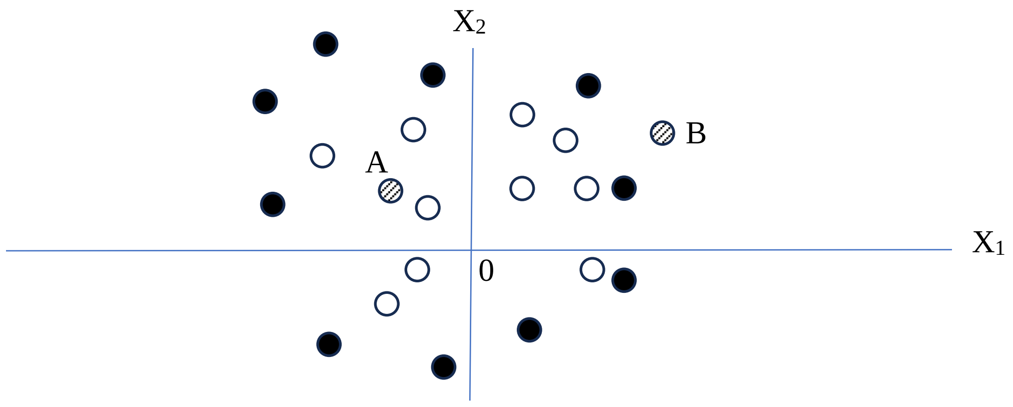


Figure 4.4

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 a 3-dimensional representation of the targets/labels in Figure 4.5.

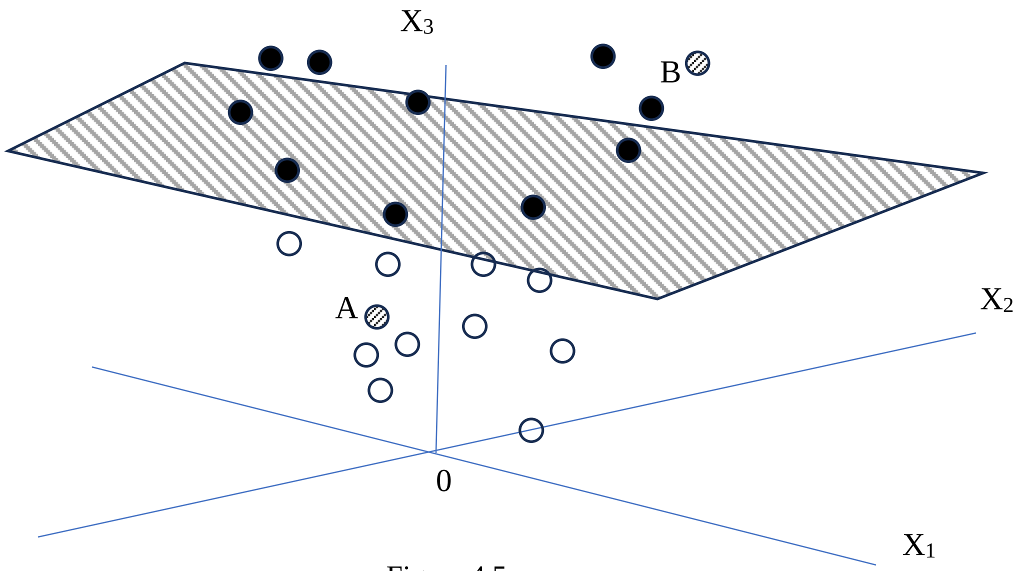


Figure 4.5

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.) 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.

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 j^{th} sample point category/type, i.e., let $Y_j = +1$ or -1 depending on which of 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 on the lines. The double-arrowed line in Figure 4.6 shows the extent of the margin.

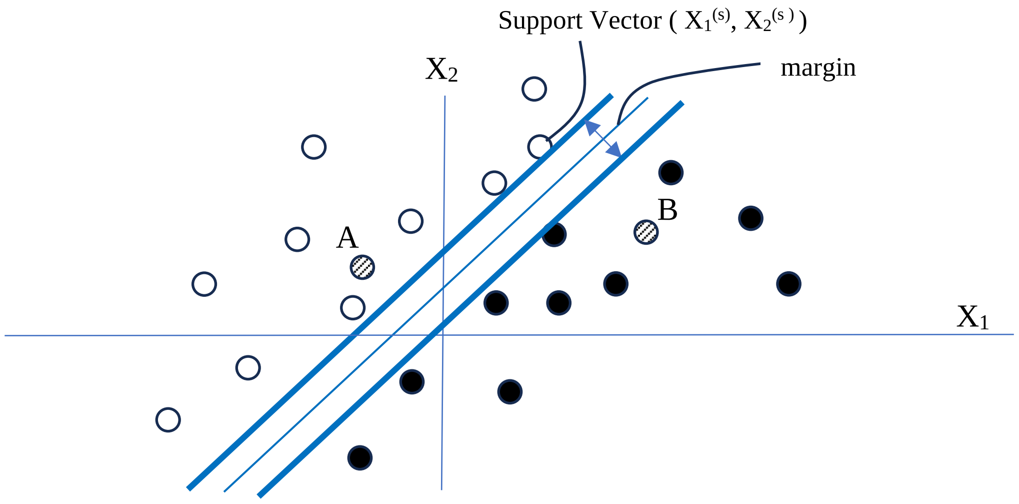


Figure 4.6

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 is c/b.

Let the supporting vector on white circle lie on line $AX_1+ BX_2 - C = +\delta$. Hence slope of "margin" line is $-A/B$. Intercept of line is $(C+\delta)/B$. Let the supporting vector on black circle lie on line $AX_1+ BX_2 - C = -\delta$. Hence slope of margin line is $-A/B$. Intercept of line is $(C-\delta)/B$. 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 $AX_1+ BX_2 - C \geq +\delta$, i.e., the white circles lie above (on the positive side of) the "margin" line $AX_1+ BX_2 - C = +\delta$. The black circles or points have features that obey $AX_1+ BX_2 - C \leq -\delta$, i.e., the black circles lie below (on the negative side of) the "margin" line $AX_1+ BX_2 - C = -\delta$.

Consider any perpendicular line from a point on $AX_1 + BX_2 - C = +\delta$ to a point on $AX_1 + BX_2 - C = -\delta$. 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 $AX_1 + BX_2 - C = +\delta$ is the point with co-ordinates ($A[C+\delta]/[A^2+B^2]$, $B[C+\delta]/[A^2+B^2]$). The intersection of this perpendicular with line $AX_1 + BX_2 - C = -\delta$ is the point with co-ordinates ($A[C-\delta]/[A^2+B^2]$, $B[C-\delta]/[A^2+B^2]$). Therefore, the distance between these two intersection points is the shortest distance between the two margin lines. This distance is

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

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 $2\delta/\|W\|$ (for the two-dimensional case here). The task in this linear SVM is to find the minimum $\|W\|$ for the maximum margin $2\delta/\|W\|$ of the wide "street" sandwiched between the two "margin" lines. This is to separate the two sets of colored circles as much as possible in order to provide a clear decision boundary for test or generalized cases.

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 $X_j \equiv (X_1^{(j)}, X_2^{(j)}, X_3^{(j)}, \ldots, X_p^{(j)}) \in R^p$. In addition, each point belongs to one of two classes denoted by $Y_j = +1$ or $Y_j = -1$. (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 or weights on X, and W_0 denote a constant vector. Suppose the separating hyperplane is $W^T X + W_0 = 0$. Then a unit vector perpendicular[3] to the separating hyperplane is $W/\|W\|$. Similarly, $-W/\|W\|$ is an opposite unit vector perpendicular to the same separating hyperplane.

If X_j is a point in R^p and X_0 is its perpendicular projection onto the separating hyperplane (X_0 is on the separating hyperplane), then $X_j = X_0 + \gamma_j W/\|W\|$ where $\gamma_j > 0$ is the distance of the perpendicular projection of X_j. Let X_j be a sample point or vector on the positive side or class $Y_j = +1$.

Now $X_0 = X_j - \gamma_j W/\|W\|$. Since X_0 is on the separating hyperplane, $W^T X_0 + W_0 = 0$. Thus, $W^T (X_j - \gamma_j W/\|W\|) + W_0 = 0$. Or, $W^T X_j + W_0 = \gamma_j W^T W/\|W\| = \gamma_j \|W\|$. Note that $\gamma_j \|W\|$ is equal to δ in the earlier two-dimensional case.

Hence, for the points on the positive side with class $Y_j = +1$,

$$\gamma_j = \frac{W^T X_j + W_0}{\|W\|}\,.$$

[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.

[3] Cosine law states that in a triangle with sides having lengths a, b, c, with c the largest, then $c^2 = a^2 + b^2 - 2ab \cos\theta$ where θ is the angle between the sides having lengths a and b. But vectors u and v are such that the vector $v - u$ forms the third side of a triangle. Then $\|v - u\|^2 = \|u\|^2 + \|v\|^2 - 2\,\|u\|\,\|v\| \cos\theta$ by the cosine law. Or, $(v-u) \bullet (v-u) = u \bullet u + v \bullet v = 0$ if and only if (iff) u and v are perpendicular, i.e., $\cos\theta = 0$. Hence $u \bullet v = 0$ iff u and v are perpendicular. Now, any two points X_a and X_b on the separating hyperplane obeys $W^T X_a + W_0 = 0$ and $W^T X_b + W_0 = 0$, so $W^T (X_a - X_b) = 0$. But $(X_a - X_b)$ is a vector lying on the hyperplane. Hence W is a perpendicular vector to $(X_a - X_b)$. Thus W is perpendicular to the separating hyperplane. W has length $\sqrt{(W \bullet W)} = \|W\|$. Hence $W/\|W\|$ has unit length.

If, however, X_j is a sample point or vector on the negative side or class $Y_j = -1$. Then $X_j = X_0 - \gamma_j W/||W||$ where $\gamma_j > 0$ is the distance of the perpendicular projection of X_j. Now $X_0 = X_j + \gamma_j W/||W||$. Since X_0 is on the separating hyperplane, $W^T X_0 + W_0 = 0$. Thus, $W^T (X_j + \gamma_j W/||W||) + W_0 = 0$. Or, $W^T X_j + W_0 = -\gamma_j W^T W/||W|| = -\gamma_j ||W||$.

Hence, for the points on the negative side with class $Y_j = -1$,

$$\gamma_j = -\frac{W^T X_j + W_0}{||W||}.$$

Assume the classes are separable by the separating hyperplane. For class $Y_j = +1$ points on the positive side of the separating hyperplane, we let $\gamma_j = \frac{W^T X_j + W_0}{||W||} > 0$ be a positive signed distance. For class $Y_j = -1$ points on the negative side of the separating hyperplane, we let $\gamma_j = -\frac{W^T X_j + W_0}{||W||} < 0$ be a negative signed distance. We can call the distances $\gamma_j > 0$ as positive SVM scores, and the distances $\gamma_j < 0$ as negative SVM scores.

Then for the points on the positive side with class $Y_j = +1$,

$$Y_j \gamma_j = Y_j \frac{W^T X_j + W_0}{||W||} > 0.$$

For the points on the negative side with class $Y_j = -1$, negative Y_j multiplied by negative $\frac{W^T X_j + W_0}{||W||}$ gives

$$Y_j \gamma_j = Y_j \frac{W^T X_j + W_0}{||W||} > 0.$$

For all sample points j in the training data set, their absolute perpendicular distances $\gamma_j = \left|\frac{W^T X_j + W_0}{||W||}\right|$ from the separating hyperplane should be above or equal to a minimum $\gamma > 0$. Twice γ is the margin of the SVM. Hence,

$$Y_j \frac{W^T X_j + W_0}{||W||} \geq \gamma \quad \text{for all j.}$$

But a constant c times W, W_0 gives $(cW^T X_j + cW_0)/||cW|| = (W^T X_j + W_0)/||W||$. Hence there is no unique solution (W, W_0) linking a support vector X_s to minimum projection distance γ. Suppose the solution γ exists and is unique given the sample data set. We can find a unique solution (W, W_0) by fixing $||W|| = 1/\gamma$ given the data set. This is fixing $||W|| \gamma = 1$.

Suppose the solution to a SVM separating hyperplane $W^T X + W_0 = 0$ is

$$\max_{W, W_0, \gamma} \gamma \qquad \text{subject to } Y_j \frac{W^T X_j + W_0}{||W||} \geq \gamma \text{ for all j.}$$

For unique solution (W, W_0), the fixing of $||W|| \gamma = 1$ allows the program to be re-written as:

$$\max_{W, W_0} 1/||W|| \qquad \text{subject to } Y_j (W^T X_j + W_0) \geq 1 \text{ for all j.}$$

This is equivalent to a quadratic program:

$$\min_{W, W_0} \frac{1}{2}||W||^2 \quad \text{subject to } Y_j (W^T X_j + W_0) \geq 1 \text{ for all j.} \tag{4.5}$$

To solve the constrained optimization shown above more readily, we make use of the Lagrange duality. This involves the Lagrange primal form and the dual Lagrange form to facilitate numerical computations of a solution. The Lagrange duality concept and solution is explained in the following sub-section.

Lagrange Duality Solution

A numerical solution of W, W_0 (each is p-dimensional) may be too cumbersome as the inequality constraints $Y_j(W^T X_j + W_0) \geq 1$ needs to be checked for each j (with $p \times 1$ X_j and 1×1 Y_j) given a potential solution (W, W_0). For this context, a neater solution is to make use of the Lagrangian. Recall program (4.5) is to maximize the margin of the SVM with the separating hyperplane as $W^T X + W_0 = 0$ for some optimal W and W_0. The primal representation of program (4.5) is

$$\min_{W,W_0} \max_{\lambda \geq 0} \ \frac{1}{2}\|W\|^2 + \sum_{j=1}^{N} \lambda_j \left(1 - Y_j(W^T X_j + W_0)\right) \tag{4.6}$$

where N×1 vector $\lambda = (\lambda_1, \lambda_2, \lambda_3, \ldots\ldots, \lambda_N)^T$, and $\lambda_j \geq 0$ (defined to be positive) is a Lagrange multiplier on the inequality constraint $1 - Y_j(W^T X_j + W_0) \leq 0$. The Lagrange multiplier acts like a penalty if the constraint is violated such as when $1 - Y_j(W^T X_j + W_0) > 0$. In this case $\lambda_j > 0$ is selected to be sufficiently high, hence the max function, to penalize the large positive value of $1 - Y_j(W^T X_j + W_0)$, in order to force the subsequent minimization over (W,W_0) to then ensure the constraint apply. λ_j is also the "shadow price" of the constraint $1 - Y_j(W^T X_j + W_0) \leq 0$. It is the amount of decrease in the objective function $\frac{1}{2}\|W\|^2$ with a marginal relaxation of the constraint. It would be incorrect to use minimization over λ_j since a violation in $1 - Y_j(W^T X_j + W_0) > 0$ means $\lambda_j = 0$. This is contradictory as zero shadow price means the constraint is already met, which is not the case here. Note that the Lagrangian and use of the Lagrange multipliers allow optimization in principle over an unconstrained program.

The dual Lagrange form of (4.6) is

$$\max_{\lambda \geq 0} \min_{W,W_0} \ \frac{1}{2}\|W\|^2 + \sum_{j=1}^{N} \lambda_j \left(1 - Y_j(W^T X_j + W_0)\right) \tag{4.7}$$

Note that we typically can construct a dual for a primal program. The dual program $\max_{\lambda \geq 0} \min_{\Theta} L(\Theta, \lambda) \leq \min_{\Theta} \max_{\lambda \geq 0} L(\Theta, \lambda)$, the primal program. An example in which inequality could occur is when the primal goes to infinity with a $\lambda_j \uparrow \infty$ due to a stubborn violation of constraint. The dual Lagrange form in (4.7) has a finite solution where

$$\min_{W,W_0 | \lambda} \ \frac{1}{2}\|W\|^2 + \sum_{j=1}^{N} \lambda_j \left(1 - Y_j(W^T X_j + W_0)\right) \tag{4.8}$$

is performed with a given λ as a kind of first stage optimization before the second stage optimization over $\lambda \geq 0$ (some elements of λ can be = 0). For our context of solving the SVM problem, when an optimal interior solution exists, $\max_{\lambda \geq 0} \min_{\Theta} L(\Theta, \lambda) = \min_{\Theta} \max_{\lambda \geq 0} L(\Theta, \lambda)$, for a general parameter Θ and Lagrangian L.

In the first stage solution in (4.8), the first order conditions for minimum are:

$$\frac{\partial L(\lambda)}{\partial W} = W - \sum_{j=1}^{N} \lambda_j Y_j X_j = 0$$

and
$$\frac{\partial L(\lambda)}{\partial W_0} = \sum_{j=1}^{N} \lambda_j Y_j = 0$$

Fixing these optimal values for W* in (4.8), we have the second stage optimization in $\lambda \geq 0$:

$$\max_{\lambda\geq 0|W^*} \frac{1}{2}\left(\sum_{j=1}^{N}\lambda_j \mathrm{Y_j X_j}\right)\bullet\left(\sum_{j=1}^{N}\lambda_j \mathrm{Y_j X_j}\right)+\sum_{j=1}^{N}\lambda_j\left(1-\mathrm{Y_j}\left(\sum_{j=1}^{N}\lambda_j \mathrm{Y_j X_j}\right)^{\mathrm{T}}\mathrm{X_j}\right)-\mathrm{W_0}\sum_{j=1}^{N}\lambda_j \mathrm{Y_j}$$

$$\equiv \max_{\lambda\geq 0|W^*} \tfrac{1}{2}\left(\sum_{j=1}^{N}\sum_{k=1}^{N}\lambda_j\,\lambda_k \mathrm{Y_j Y_k X_j^T X_k}\right)+\sum_{j=1}^{N}\lambda_j-\sum_{j=1}^{N}\lambda_j\,\mathrm{Y_j}\left(\sum_{j=1}^{N}\lambda_j \mathrm{Y_j X_j}\right)^{\mathrm{T}}\mathrm{X_j}$$

$$\equiv \max_{\lambda\geq 0|W^*} \tfrac{1}{2}\left(\sum_{j=1}^{N}\sum_{k=1}^{N}\lambda_j\,\lambda_k \mathrm{Y_j Y_k X_j^T X_k}\right)+\sum_{j=1}^{N}\lambda_j-\sum_{j=1}^{N}\sum_{k=1}^{N}\lambda_j\,\lambda_k\,\mathrm{Y_j Y_k X_j^T X_k}$$

$$\equiv \max_{\lambda\geq 0|W^*} \sum_{j=1}^{N}\lambda_j-\tfrac{1}{2}\sum_{j=1}^{N}\sum_{k=1}^{N}\lambda_j\,\lambda_k\,\mathrm{Y_j Y_k X_j^T X_k} \qquad (4.9)$$

subject to $\sum_{j=1}^{N}\lambda_j \mathrm{Y_j}=0$ and $\lambda_j \geq 0$ for all j.

The numerical solution of dual in (4.9) is tractable given observations $\mathrm{Y_j Y_k X_j^T X_k}$ for every j, k. The solutions λ_j^* are then applied to obtain W* = $\sum_{j=1}^{N}\lambda_j^*\mathrm{Y_j X_j}$. Note that W* is solution to the separating hyperplane $\mathrm{W^{*T}}$ X + $\mathrm{W_0}$* = 0. Note that for uniqueness of solution in W*, $\mathrm{W_0}$*, with setting $\gamma\ \|\mathrm{W}^*\| = 1$, and $\mathrm{Y_j}\left(\mathrm{W^{*T}X_j}+\mathrm{W_0^*}\right)\geq 1$ for all j, we have support vector(s) on the positive side ($\mathrm{Y_j}$ = +1) "margin" hyperplane as $\mathrm{W^{*T}X_j}+\mathrm{W_0^*}$ = 1 or min $\left(\mathrm{W^{*T}X_j}+\mathrm{W_0^*}\mid \mathrm{Y}_j=+1\right)=1$, or min $\left(\mathrm{W^{*T}X_j}\mid \mathrm{Y}_j=+1\right)=1-\mathrm{W_0^*}$.

Support vector(s) on the negative side ($\mathrm{Y_j}$ = –1) "margin" hyperplane are $\mathrm{W^{*T}X_j}+\mathrm{W_0^*}$ = –1, or max $\left(\mathrm{W^{*T}X_j}+\mathrm{W_0^*}\mid \mathrm{Y}_j=-1\right)=-1$, or max $\left(\mathrm{W^{*T}X_j}\mid \mathrm{Y}_j=-1\right)=-1-\mathrm{W_0^*}$. Then $\mathrm{W_0^*}$ can be found as

$$\mathrm{W_0^*}=-\left[\min\left(\mathrm{W^{*T}X_j}\mid \mathrm{Y}_j=+1\right)+\max\left(\mathrm{W^{*T}X_j}\mid \mathrm{Y}_j=-1\right)\right]/2.$$

The score of any sample point can be measured as $\mathrm{W^{*T}X_j}+\mathrm{W_0^*}$ which is the distance of $\gamma_\mathrm{j}=\frac{\mathrm{W^{*T}X_j+W_0^*}}{\|\mathrm{W}^*\|}$ in terms of $\gamma = 1/\|\mathrm{W}^*\|$, hence $\gamma_\mathrm{j}/\gamma = \mathrm{W^{*T}X_j}+\mathrm{W_0^*}$. This score can be positive (on the positive side of the separating hyperplane) or negative (on the negative side of the separating hyperplane).

When it comes to prediction of a class given a generalized data point with features $\mathrm{X_g}$ (including predictions on test data), the SVM score of the generalized data point is measured using the trained SVM, i.e., score = $\mathrm{W^{*T}X_g}+\mathrm{W_0^*}$. If the score is ≥ +1, the point or case is predicted to be in the class $\mathrm{Y_g}$ = +1. If the score is ≤ –1, the point or case is predicted to be in the class $\mathrm{Y_g}$ = –1.

Non-Linear Kernels in SVM

So far we have used a linear separating hyperplane, $\mathrm{W^T X_0 + W_0} = 0$ based on the p-dimensional feature vector $\mathrm{X_k} \equiv (\mathrm{X_1^{(k)}, X_2^{(k)}, X_3^{(k)}, \ldots, X_p^{(k)}}) \in \mathrm{R^p}$. Suppose the optimization in (4.9) fails as a linear separating hyperplane cannot be found. In (4.9), the term $\mathrm{X_i^T X_j}$ involving the dot product of $\mathrm{X_i}$ and $\mathrm{X_j}$ is called a linear kernel on $\mathrm{X_i}$ and $\mathrm{X_j}$.

We saw earlier how we can possibly use a nonlinear transformation (nonlinear kernel) to transform the features or feature vectors $\mathrm{X_i}$ and $\mathrm{X_j}$. Nonlinear SVM kernels that can be used for classification include the following.

The polynomial kernel $\mathrm{K(X_i,X_j)} = (\gamma \mathrm{X_i \bullet X_j}+c)^d$ where the degree of the polynomial d, gamma γ, and c are hyperparameters. Sometimes a simpler version of γ = 1 or γ = 1/N is used as a default case in many programs. As an illustration, suppose point $\mathrm{X_i}$ is $(x_1^{(i)}, x_2^{(i)})$ with two dimensions. Let $\mathrm{K(X_i,X_j)} = (\mathrm{X_i \bullet X_j} +1)^2$ where d = 2. Then $\mathrm{K(X_i,X_j)} = (x_1^{(i)}x_1^{(j)} + x_2^{(i)}x_2^{(j)} + 1)^2 = 2\ x_1^{(i)}x_1^{(j)} + 2\ x_2^{(i)}x_2^{(j)} + 2\ x_1^{(i)}x_1^{(j)}x_2^{(i)}x_2^{(j)} + (x_1^{(i)}x_1^{(j)})^2 + (x_2^{(i)}x_2^{(j)})^2 +1 = (\surd 2\ x_1^{(i)}, \surd 2\ x_2^{(i)}, \surd 2\ x_1^{(i)}x_2^{(i)}, x_1^{(i)\,2}, x_2^{(i)\,2}, 1) \bullet (\surd 2\ x_1^{(j)}, \surd 2\ x_2^{(j)}, \surd 2\ x_1^{(j)}x_2^{(j)},$

$x_1^{(j)\,2}$, $x_2^{(j)\,2}$, 1). Clearly, the polynomial kernel now expands X_i from two to six dimensions. The higher dimension allows for more space in-between sample point features and hence 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 2-dimensional X_i, X_j , $K(X_i,X_j) = \phi(x_1^{(i)},x_2^{(i)}) \bullet \phi(x_1^{(j)},x_2^{(j)}) = \phi(X_i) \bullet \phi(X_j)$. The latter is a linear kernel on $\phi(X_i)$, $\phi(X_j)$.

Without elaborating on some highly technical details, we state that there is a Mercer's Theorem[4] 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 on linear $\phi(X_i)$, $\phi(X_j)$ for some function $\phi(.)$ associated with the non-linear kernel. See (4.10).

$$\max_{\lambda\geq 0|W^*} \quad \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\,\lambda_j\, Y_i Y_j\, \phi(X_i)\bullet \phi(X_j) \tag{4.10}$$

Another popular non-linear kernel is 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, $\gamma = 1/(N\sigma^2)$. Another example would be to set γ to a positive number. High gamma 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 $K(X_i,X_j)$ 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.

The choice of what type of kernel depends on the nature of the data and the problem. Linear kernel is used 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.

In general, when we apply a particular kernel function as in the constrained optimization in Eq. (4.10), we check for the performance results in prediction accuracy and the classification errors on the validation or test sample data set. There is no need to explicitly determine or check the transformation function of $\phi(.)$ in $K(X_i,X_j) = \phi(X_i) \bullet \phi(X_j)$.

For SVM in classification prediction, the training data set is used to estimate optimally the separating hyperplane parameters with pre-selected sets of kernel parameters and associated hyperparameters. Then a validation set can be used to fine-tune the hyperparameters by seeking best performance metrics. After that, the fine-tuned hyperparameters can be used to train a re-grouped or enlarged training data set, and then this trained model is applied to the test set to check performance metrics for robustness.

Linear Non-Separability in SVM

Suppose the optimization in (4.9) still fails as a linear separating hyperplane cannot be found despite suitable transformations of features or feature vector X_i into $\phi(X_i)$ with perhaps a higher dimension. We can see this in Figure 4.7 whereby any transformations $\phi(X_i)$ of a sample point into two dimensions (Z_1, Z_2) could not allow linear separation.

[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.

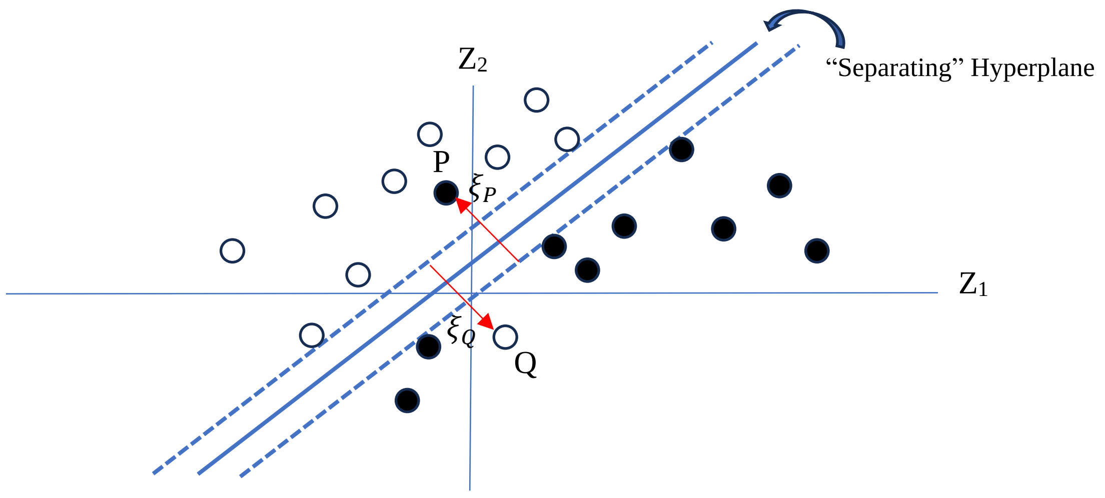


Figure 4.7

In general, denote the sample point feature vector as Z_j. In forming the "separating" hyperplane or optimal decision boundary, there will still be support vectors touching the "margin" lines whereby according to (4.7), $Y_j\left(W^{*T}Z_j + W_0^*\right) = 1$. And most sample points would have $Y_j\left(W^{*T}Z_j + W_0^*\right) > 1$. However, as seen in Figure 4.7, there would be points such as P, Q that violate the constraints. In particular, the distance of P away from its "margin" into the opposite side is ξ_P, and the distance of Q away from its "margin" into the opposite side is ξ_Q.

To avoid such violations, we put slack variables ξ_j into the constraints such that $Y_j\left(W^{*T}Z_j + W_0^*\right) \geq 1 - \xi_j$, and $\xi_j \geq 0$ for all j. This is also called soft margin classification. If all $\xi_j = 0$, then we are back to the solution with separation or hard margin classification.

The dual program to solve in the context of allowing some points to have $\xi_j > 0$, similar to (4.8), is

$$\min_{W,W_0,\xi|\lambda,\mu} \quad \frac{1}{2}\|W\|^2 + \alpha\sum_{j=1}^{N}\xi_j + \sum_{j=1}^{N}\lambda_j\left(1 - \xi_j - Y_j\left(W^TZ_j + W_0\right)\right) - \sum_{j=1}^{N}\mu_j\xi_j \qquad (4.11)$$

given λ and μ, where $\alpha > 0$ is a hyperparameter denoting the penalty to put on $\sum_{j=1}^{N}\xi_j > 0$. N×1 vector $\lambda = (\lambda_1, \lambda_2, \lambda_3, \ldots\ldots, \lambda_N)^T$, and $\lambda_j \geq 0$ (defined to be positive) is a Lagrange multiplier on the inequality constraint $Y_j\left(W^TZ_j + W_0\right) \geq 1 - \xi_j$ for all j. N×1 vector $\mu = (\mu_1, \mu_2, \mu_3, \ldots\ldots, \mu_N)^T$, and $\mu_j \geq 0$ (defined to be positive) is a Lagrange multiplier on the inequality constraint $\xi_j \geq 0$ for all j. N×1 vector $\xi = (\xi_1, \xi_2, \xi_3, \ldots\ldots, \xi_N)^T$. A large penalty α tends to reduce $\sum_{j=1}^{N}\xi_j$ as a choice, i.e., small error rates of training misclassifications, while a small penalty α and large $\sum_{j=1}^{N}\xi_j$ results in larger margin and large error rates of training misclassifications. α is to be fine-tuned for optimal performance in a validation data set before final testing is done. If we re-arrange the objective function as $\frac{1}{2}\|W\|^2 + \alpha'\sum_{j=1}^{N}\left(1 - Y_j\left(W^TZ_j + W_0\right) + \eta^2\right)$ for some real η and given λ_j's, or equivalently in the minimizer to find W, W_0 , $\frac{1}{2\alpha'}\|W\|^2 + \sum_{j=1}^{N}\left(1 - Y_j\left(W^TZ_j + W_0\right)\right)^+$ where the superscript '+" denotes the argument taking a value larger or equal to zero, then $\left(1 - Y_j\left(W^TZ_j + W_0\right)\right)^+$ is called a hinge loss function.

Program (4.11) can be re-written as:

$$\min_{W,W_0,\xi|\lambda,\mu} \quad \frac{1}{2}\|W\|^2 + \sum_{j=1}^{N}\xi_j\left(\alpha - \lambda_j - \mu_j\right) + \sum_{j=1}^{N}\lambda_j\left(1 - Y_j\left(W^TZ_j + W_0\right)\right) \qquad (4.12)$$

The first order conditions for minimum are:

$$\frac{\partial L(\lambda,\mu)}{\partial W} = W - \sum_{j=1}^{N} \lambda_j \mathrm{Y_j} \mathrm{Z_j} = 0$$

$$\frac{\partial L(\lambda,\mu)}{\partial W_0} = \sum_{j=1}^{N} \lambda_j \mathrm{Y_j} = 0$$

and

$$\frac{\partial L(\lambda,\mu)}{\partial \xi} = \alpha - \lambda - \mu = 0.$$

Fixing these optimal values for W* in (4.12), we have the second stage optimization in $\lambda \geq 0$:

$$\max_{\lambda\geq 0|W^*} \quad \sum_{j=1}^{N} \lambda_j - \frac{1}{2}\sum_{j=1}^{N}\sum_{k=1}^{N} \lambda_j\, \lambda_k\, \mathrm{Y_j Y_k Z_j^T Z_k} \qquad (4.13)$$

subject to $\sum_{j=1}^{N} \lambda_j \mathrm{Y_j} = 0$, $\alpha = \lambda_j + \mu_j$, $\lambda_j \geq 0$ and $\mu_j \geq 0$ for all j. The constraints here can be simplified to $\sum_{j=1}^{N} \lambda_j \mathrm{Y_j} = 0$ and $\alpha \geq \lambda_j \geq 0$ for all j. The solutions W*, W_0* can be similarly obtained as in the separating context. Validation or test sample points on the positive side of the “separating” hyperplane $\mathrm{W}^{*\mathrm{T}}\mathrm{Z_j} + \mathrm{W}_0^* = 0$ are predicted as $Y_j = +1$, whereas points on the negative side of the “separating” hyperplane are predicted as $Y_j = -1$.

Other Aspects of SVM

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. A common method is to use the SVM score for each sample point as the feature for predicting the class using the logit model. The predicted probabilities can then be used to build the ROC curve given different probability thresholds.

The SVM method is natively a binary classification algorithm as its separating hyperplane offers two obvious sides of the classifications. However, SVM may be extendible to multi-class separation although the methods typically involve multiple binary classifications in a heuristic approach. These approaches are applicable to ML algorithms that are natively binary in its classifications such as the logistic or logit prediction.

Suppose there are n > 2 classes for prediction. In the One-vs.-Rest (OvR) or One-vs.-All (OvA) approach, a SVM binary classification is performed for each of the n classes versus all the remaining classes combined. The training data set will be utilized n times where each time, it is a prediction of class j versus the rest (non class j). This builds up n prediction models. Let the trained jth model provide predicted probability P_j of a generalized validation or test case or instance being of class j (and $1 - P_j$) being not of class j. For all j = 1, 2, …., n, find j with the highest P_j. That j is the predicted class for the instance in the multi-class prediction problem.

In the same n class prediction problem, another heuristic approach is the One-vs.-One (OvO) method. Here a binary classification is performed for each pair of the n classes. There are n(n-1)/2 such pairs. The classification model training for each pair will use only data that contain these two classes. In each of the pair classifications, the predicted class is noted for an instance from the validation or test data set. Of the n(n-1)/2 events each with a predicted class, the class with the highest number of times of being predicted (between 0 and n-1) is the predicted class for the instance in the multi-class prediction problem.

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). Just as in multiple linear regression where the objective 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 also to minimize errors between the predicted hyperplane and the training data targets but setting the errors to non-zero values only when they exceed a given hyperparameter epsilon $\varepsilon > 0$. Within this epsilon space or tube, no penalty is associated with the error or deviation of actual target value Y_j from the fitted hyperplane $\widehat{Y}_j = W \bullet X_j - W_0$.

Just as in SVM, the SVR finds a maximum margin (equivalently minimum $\frac{1}{2}||W||^2$) where the "margin" hyperplanes have given 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 one-dimension feature 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. The pre-determined epsilon ε is large enough to contain all sample points within the epsilon tube, so the maximization of margin has zero penalty. Hence the margin is maximized by line L_2. Line L_2 has a wider margin than line L_1, i.e., $m_2 > m_1$. Hence, maximizing margin subject to displacements of $\varepsilon > 0$ has a tendency to lead to a flatter fitted hyperplane.

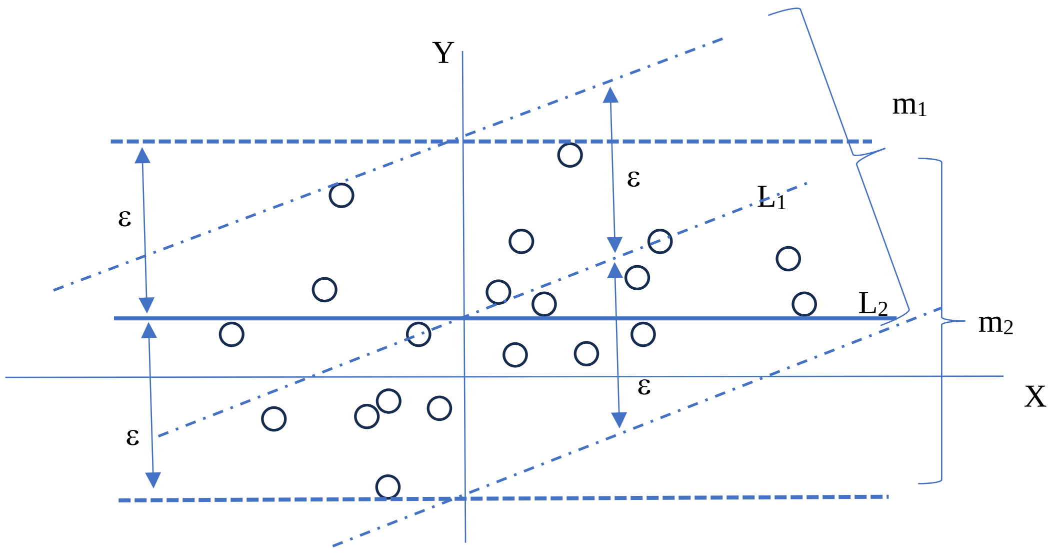


Figure 4.8

However, when epsilon ε is smaller, it may not be possible to contain all the sample points within the epsilon tube. 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

$$\min_{W, W_0, \zeta_i} \frac{1}{2} W \bullet W + \alpha \sum_{i=1}^{N} \zeta_i \tag{4.14}$$

subject to | $Y_i - (W \bullet X_j - W_0)$ | $- \varepsilon - \zeta_i \le 0$ 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.

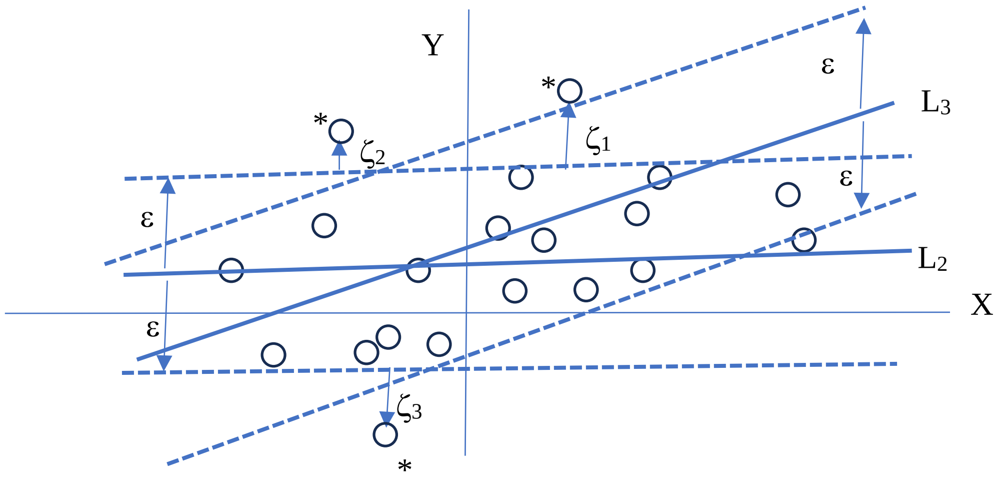


Figure 4.9

Note that in Figure 4.9, a better fit with a lower objective function in Eq. (4.14) can be found in line L_3. The hyperplane L_3 has lowest $\frac{1}{2} W \bullet W + \alpha \sum_{i=1}^{N} \zeta_i$ as the tilting of the slope and intercept shift adjust the epsilon tube such that the outside points now have less deviation from the “margin” lines and hence less penalties.

In general, 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. In this latter case, there is usually poor prediction on test data. On the other hand, when epsilon shrinks toward zero, the SVR can become overly sensitive to outliers. It approaches a linear regression. In sklearn, the default epsilon is 0.1 and the default α (or “C” in sklearn) is 1.0.

Just as in SVM, the dual problem of finding the optimal fitted hyperplane in SVR can be formulated in terms of $(X_i \bullet X_j)$. Nonlinear kernels $K(X_i,X_j) = \phi(X_i) \bullet \phi(X_j)$ can also be used in SVR to obtain fitted hyperplane after the original data points are suitably transformed via ϕ(.). The latter may help in better fitting with lower penalty costs in the objective function.

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

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

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 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 classif -ications 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
```

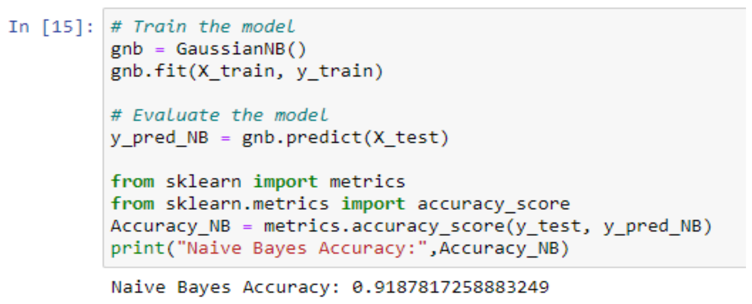
[6] A similar study by Suraj Patil, V. Nemade, and P.K. Soni, (2018), "Predictive Modelling For Credit Card Fraud Detection Using Data Analytics", Procedia Computer Science 132, pp.385-395, indicated client's checking account history, pattern of credit card payments, abnormal use and amounts of credit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

```
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
```

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), or that validation and hyperparameter tuning was already done.

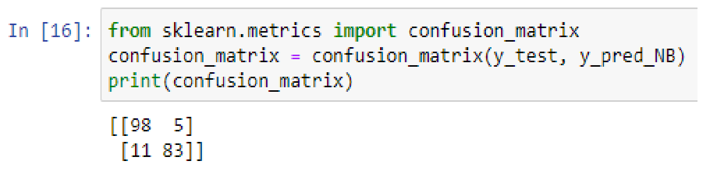
Naïve Bayes Algorithm

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.

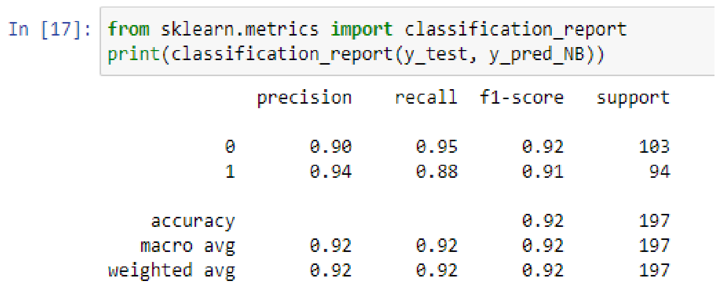


```
In [15]: # Train the model
         gnb = GaussianNB()
         gnb.fit(X_train, y_train)

         # 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
```

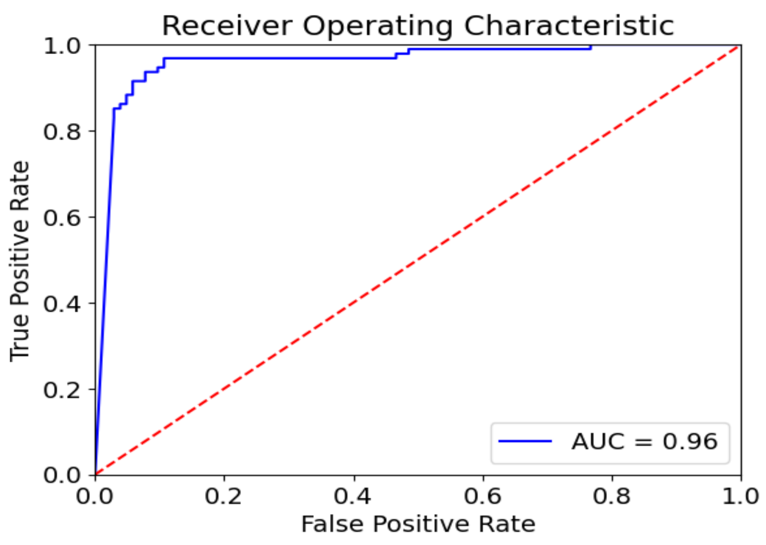
The confusion matrix, the classification report, and the ROC curve are obtained via code lines [16], [17], [18], [19]. ROC area is 96.075%.



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

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

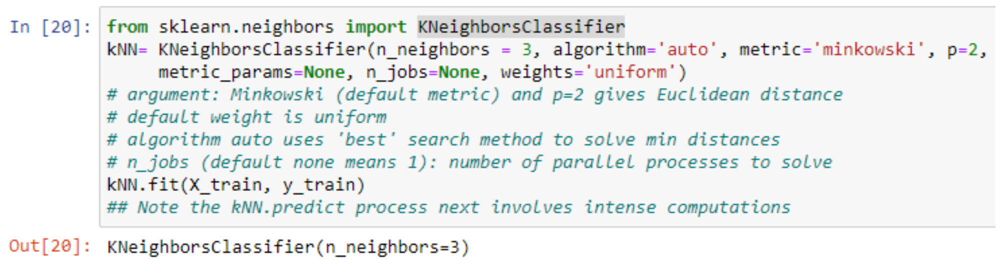


```
In [17]: from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred_NB))

                       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
```

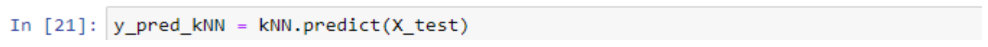


K-Nearest Neighbor Algorithm

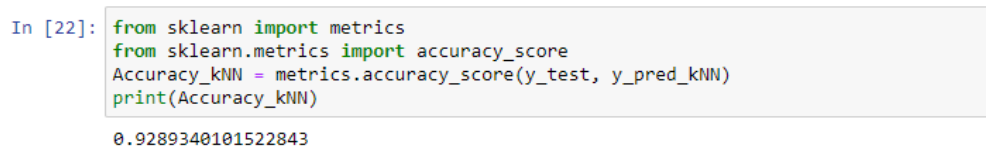
Code lines [20], [21], [22] show the application of the sklearn KNeighborsClassifier employing k = 3 to fit the training set data, then perform the prediction on the test data (X_test). The accuracy is then reported as 92.893% using the sklearn metrics on accuracy_score.



```
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)
```

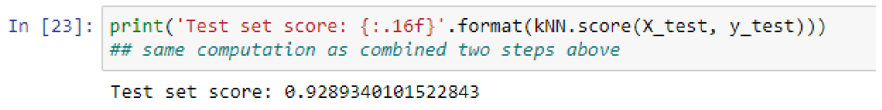


```
In [21]: y_pred_kNN = kNN.predict(X_test)
```

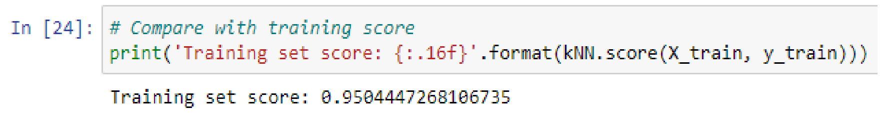


```
In [22]: from sklearn import metrics
         from sklearn.metrics import accuracy_score
         Accuracy_kNN = metrics.accuracy_score(y_test, y_pred_kNN)
         print(Accuracy_kNN)

         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
```



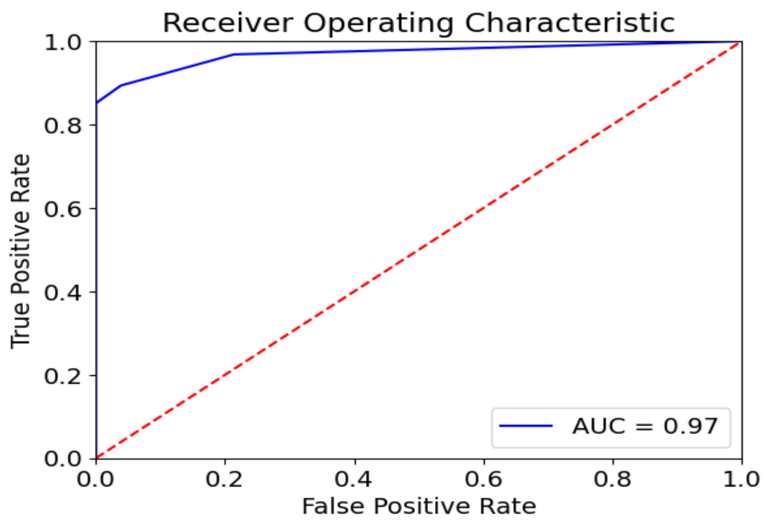
```
In [24]: # Compare with training score
         print('Training set score: {:.16f}'.format(kNN.score(X_train, y_train)))

         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.

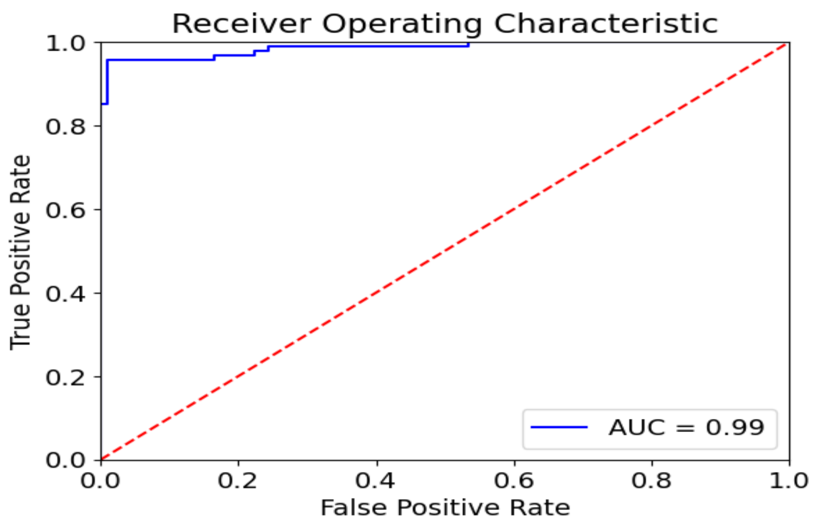
Support Vector Machine

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 earlier 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’. 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)
```

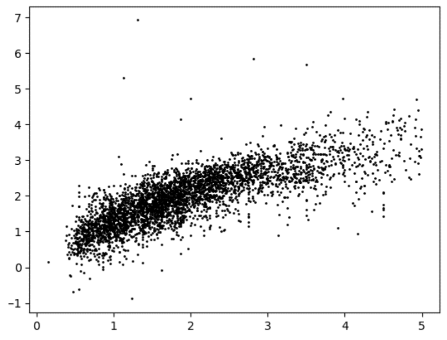
```
In [21]: r2_score_SVR1_train = r2_score(y_train, y_pred_SVR1_train)
         print('R2_score (train): ', r2_score_SVR1_train)
         ### R2_score (train) is the R-square in the linear regression involving only the training data set
         r2_score_SVR1_test = r2_score(y_test, y_pred_SVR1_test)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

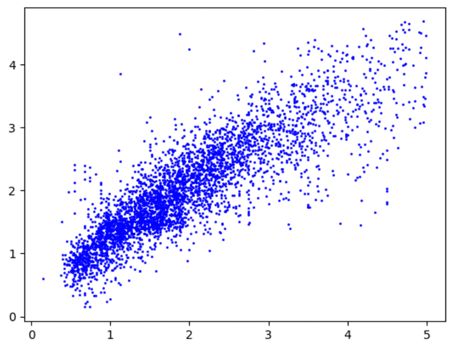
```
In [26]: SVR3 = SVR(kernel = 'rbf',epsilon=0.56,C=25)
         SVR3.fit(X_train, y_train)
         y_pred_SVR3_train = SVR3.predict(X_train)
         y_pred_SVR3_test = SVR3.predict(X_test)
```

```
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 Concluding Thoughts

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 or regression 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.

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.