QF634 APPLIED QUANTITATIVE RESEARCH METHODS
LECTURE 4

Lecturer: Prof Lim Kian Guan

Naïve Bayes, kNN, SVM Algorithms

Naïve Bayes Algorithm

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

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

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.

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

Naïve Bayes Algorithm

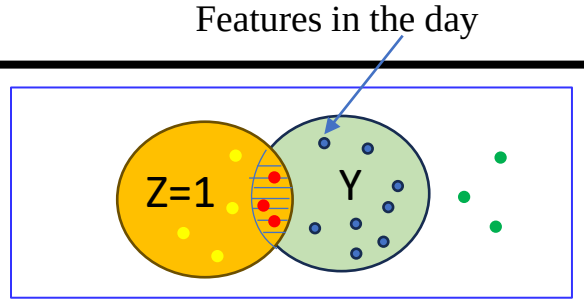
- Think of event {Z=1} as a class/type, e.g., those who are marksmen, and event {Z=0} as the rest of the universe, i.e., those who are not marksmen.
- Z is a random variable that can take the value 1 or else 0, i.e., there is a binary classification. Bayes' Theorem can be expressed as:

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

where Z^c is complement of event Z, i.e., $Z^c = \Omega \setminus Z$ (Ω less Z), and $P(Y) = P(Y \cap Z) + P(Y \cap Z^c)$. In our notations, $Z^c \equiv \{Z=0\}$. This latter version of Bayes' Theorem is useful in the context of updating prior information or probability of Z, P(Z).

- Example. Medical testing, where a patient has apriori (without any testing for further information) a probability P(Z) = x of having the medical condition Z. x (%) could be obtained from some statistical averages over reports of such medical conditions in various countries. A medical test on the patient could produce information Y, e.g., a positive test case. It may not be easy to obtain total information across all countries of the % of all tests producing Z and Y, i.e., $P(Z \cap Y)$. However, it may be easier to obtain summary reports on $P(Y|Z)$ and $P(Y|Z^c)$, e.g., percentage of cases that have Z and tested for Y, and percentage of non-Z and tested (false positive) for Y.

Naïve Bayes Algorithm



- The Naïve Bayes classification algorithm works using the Bayes Theorem:

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

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 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. (Assume next stock price index remaining constant has zero probability or let {Z=0} include case of no change.)
- Let features F_1, F_2, …, 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.
- All the features found in a day constitute event $\{F_1, F_2, \ldots, F_n\}$ = Y, the evidence or signal for the day. More convenient to denote elements F_k collected in set as in Venn Diagram.
- 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. Red dots are mood words that coincide with Z=1 in that day. Blue dots are mood words that coincide with Z=0 in that day. Yellow dots and green dots are words not captured in the Y information vocabulary.

Naïve Bayes Algorithm

$$P(Z \mid F_1, F_2, \ldots, F_n) = P(F_1, F_2, \ldots, F_n \mid Z) \times P(Z) / P(Y) \quad \text{where } Y = \{F_1, F_2, \ldots, F_n\}$$

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

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

- By Bayes' Theorem,

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

Naïve Bayes Algorithm

- Using the simplifying assumption on the last line,

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

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

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

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

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

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

Naïve Bayes Algorithm

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

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

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

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

Accuracy of the algorithm is evaluated based on the performance of predictors on each test data point.

- There are several methods for estimating $\hat{P}(F_j|Z=1)$ and $\hat{P}(F_j|Z=0)$ for every j in the training data set, depending on what we assume is the probability distribution of the features F_j.
- Some common distribution assumptions are the multinomial, the categorical, and Gaussian/normal distributions. 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.

Naïve Bayes Algorithm

- In the sentiment analyses using mood words as features, it is natural to assume that the features F_j, j=1,2,…,n follow the **categorical distribution**. This means that for any sample point (day), given Z, the probability of a feature F_j occurring is $P(F_j|Z)$ and the probability of the feature not occurring is $1 - P(F_j|Z)$. Thus, any feature F_1 through F_n can either occur or not occur. The categorical variable of F_j or no F_j can be hot-encoded as dummy variable 1 or 0. If there is only one feature in the dataset, then this is also a Bernoulli Naïve Bayes. In any one day, different categorical variables F_1, F_2, etc. can occur together.
- We make the simplification that a dictionary of mood words is created for the study such that all the mood words in this dictionary appear at least once in the training data set in the cases Z=1 and in the cases Z=0. In other words, we do not have a situation of 'zero frequency'.
- Suppose there are other non-natural language measures of features G_j that are continuous, such as volatility (j=1), volume traded on index futures (j=2), lagged index change (j=3), etc., so that it is not feasible to count category frequency to estimate $\hat{P}(G_j|Z)$. In such a situation, we can use the Gaussian Naïve Bayes (GNB) method which employs the Gaussian or normal probability distribution. GNB assumes that $P(G_j|Z)$ (Z=1 or Z=0) follows a Gaussian/normal probability distribution:

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

where $\mu_{zj} = E(G_j)$ and $\sigma_{zj} = \sqrt{var(G_j)}$.

Naïve Bayes Algorithm

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

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

- Note that with GNB, there is no need to first standardize/normalize the features first before training if the test data features also have the same mean and variance as those in the training data since $\hat{\mu}_{zj}, \hat{\sigma}_{zj}$ estimates from the training data would suitably represent parameters from the test data set.
- However, if the mean and variance from the test data set are different from those in the training data set (even if features from both sets are Gaussian), it may be more robust to first standardize the training and the test data features before applying the GNB method. After standardizing, the $\hat{\mu}_{zj}, \hat{\sigma}_{zj}$ estimates of the standardized features are 0 and 1 respectively, for both the training and test set data.
- For mixed type of data in features, e.g., some features F_j are categorical while others G_j are continuous variables, we can apply the NB method as follows. Following Eq. (4.1):

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

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

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

Naïve Bayes Algorithm

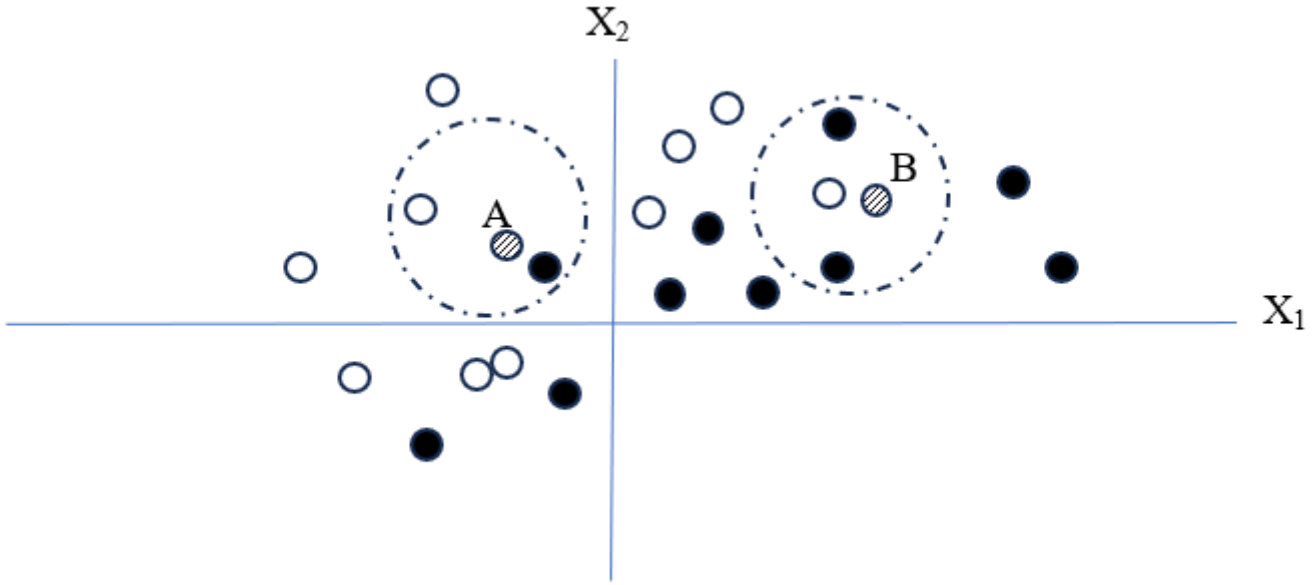
When sample data is split into training and test data sets, the Naïve Bayes algorithm is first applied to the training data set to compute the classification accuracy.

More features can be added or selected so that the training set can be adjusted to optimize the accuracy of a balanced data set.

In GNB, the training data set also provides estimation of the feature parameters of means and variances in order to compute the conditional probabilities for predicting the labels or targets.

If there are no competing models or no hyperparameters to optimize, the optimized and estimated parameters are then applied to predict cases from the test features. The predictions are compared with the test labels to find the various performance metrics as well as the ROC AUC.

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



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

k-Nearest Neighbor Algorithm

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

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

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

k-Nearest Neighbor Algorithm

- 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).

k-Nearest Neighbor Algorithm

- 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 to avoid a tie. We may have to re-run the algorithm several times for different hyperparameter values of k (tuning k) before we reach a satisfactory prediction or classification accuracy and good AUC.
- When k is decreased toward 1, the prediction for each case becomes unstable as prediction is based only on what happens to be the nearest neighbor of the test sample point. The prediction is myopic and may miss the correct affiliation as it may happen that there are more of the incorrect categories that are closer.
- But as k is increased, the prediction becomes more stable due to majority voting/averaging phenomenon, but this prediction can worsen as k becomes too large when the expanding neighborhood gets in more of both categories or types of the sample points.
- The 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.

k-Nearest Neighbor Algorithm

- The k-NN algorithm becomes slower as the number of features used for computing distances increases.
- 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.
- 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.

Support Vector Machine

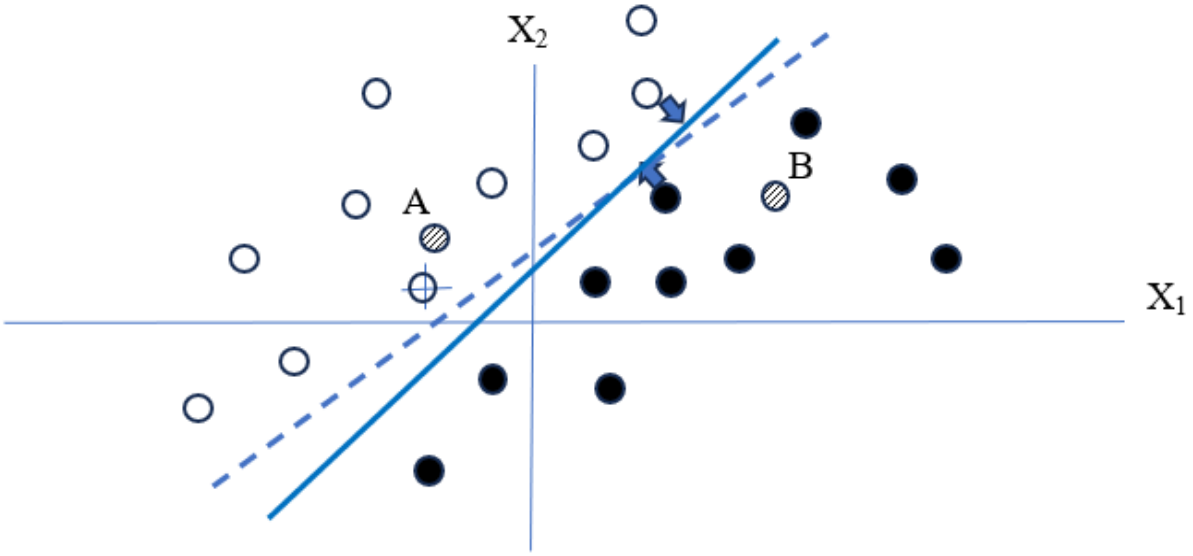


Figure 4.2

In Figure 4.2, the maximum separation is attained via the solid boundary line with positive slope.

The nearest white circle and the nearest black circle to the boundary have equal distances represented by the arrows.

Any shift in the boundary (see dotted line) would end up having at least one colored circle being closer to the boundary, e.g., the white circle shown with a cross.

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).

The two closest points from both classes are called the support vectors as they help determine the boundary. If these support vectors are shifted, then the boundary will also change.

Suppose we try to predict the targets/labels/classes of the cases represented as black or white circles. Diagram shows a simple example of cases with two features with values X_1 and X_2. In the training of the SVM, a boundary is drawn to separate the two classes/groups as far as is possible -- the colored circles are easily separated by a linear straight line. Once the SVM has fitted the training data set, i.e., determined the boundary to separate the classes in the training data set, then the test data sample points, e.g., A and B, can be easily predicted as being in the white circle and the black circle group/class respectively.

Support Vector Machine

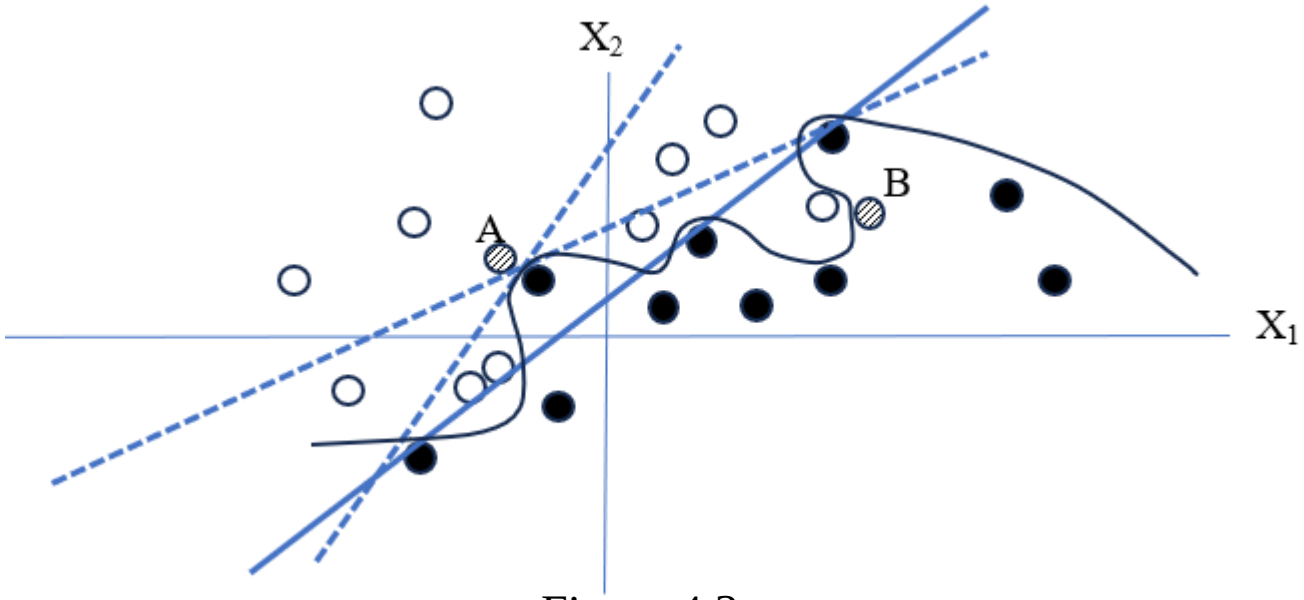


Figure 4.3

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.

Two ways when outright linear separation is not possible.

- Firstly, we could still apply linear boundary as in the solid line but allow some classification errors in training.
- Secondly, we may be able to find a polynomial function (shown in the curved line) to separate the colored circles.

Support Vector Machine

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".

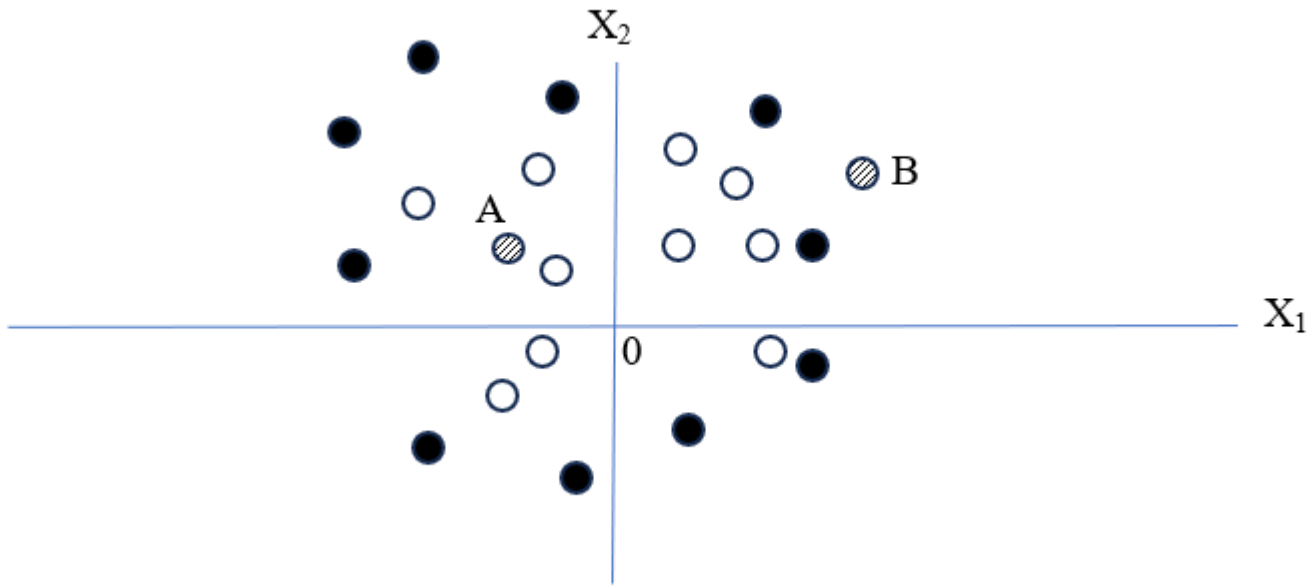
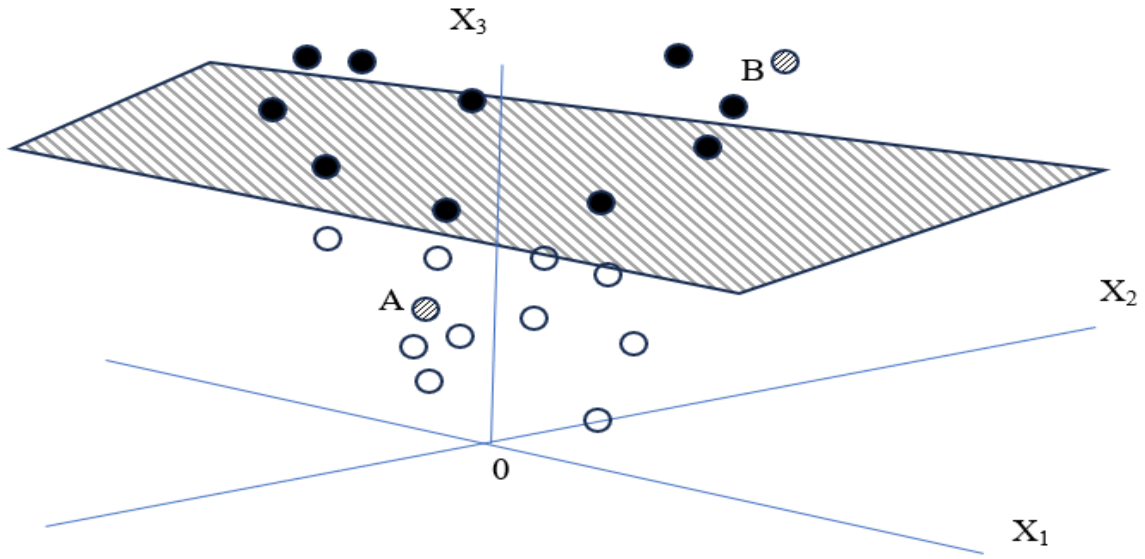


Figure 4.4

Support Vector Machine

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 j^{th} training data set sample point, where the third coordinate is $X_3 = X_1^2 + X_2^2$. By extending the feature space, we can obtain the following 3-dimensional representation of the circles. 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.

Support Vector Machine

- The theory of SVM can be explained as follows. Suppose a training sample consists of N number of data or sample points. Each sample point has p features and the j^{th} sample point can be characterized in Euclidean space as a p-dimensional vector $(X_1^{(j)}, X_2^{(j)}, X_3^{(j)}, \ldots, X_p^{(j)})$.
- Each point belongs to one of two classes. 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 on the binary class/type.

Consider again the two-dimensional features in an earlier example. We want to draw a hyperplane (a straight line in 2-dimension) to separate the black circles (target/label $Y_j = -1$) from the white circles (target/label $Y_j = +1$).

Moreover, we want a straight line with the widest margin (wide "street") with the two margin lines (shown as bold lines) touching the support vectors or the points closest to (or on) the lines.

The double-arrowed line in Figure 4.6 shows the extent of the margin.

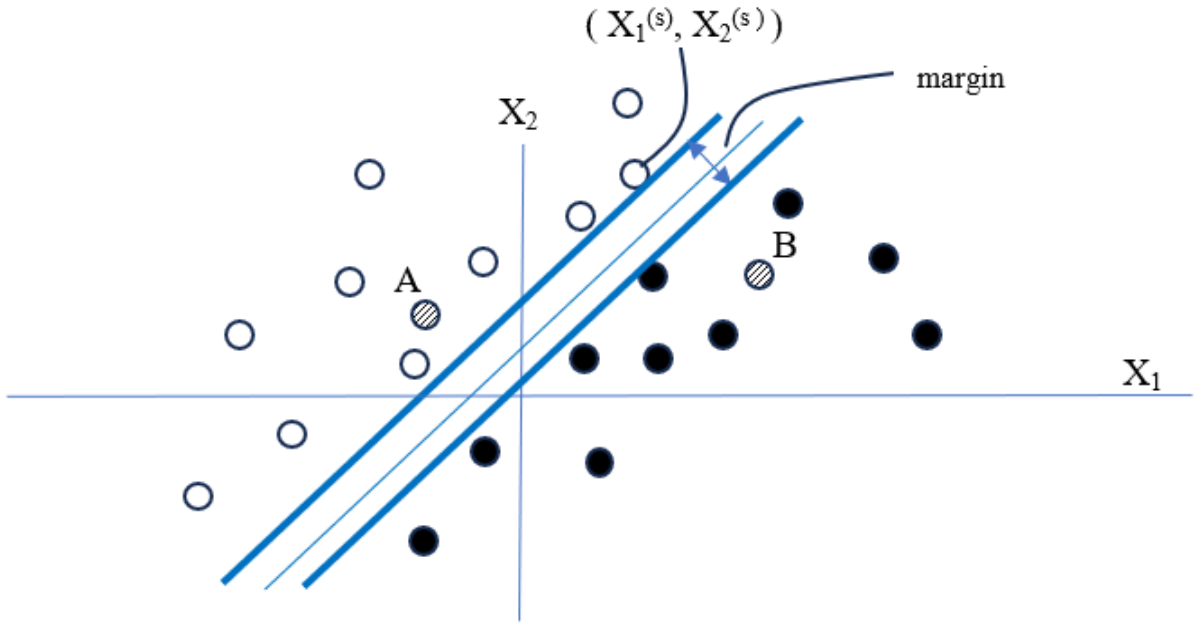


Figure 4.6

Support Vector Machine

- In the 2-D case here, the straight lines can be represented algebraically as equations $aX_1+bX_2 = c$ for different values of c. This is a more general representation (allowing b to be possibly 0) of the familiar $X_2 = c/b - a/b\ X_1$ where slope is $-a/b$ and the intercept on the X_2 axis id c/b.
- Let the supporting vector on white circle lie on line $AX_1+ BX_2 - C = +1$. Hence slope of margin line is $-A/B$. Intercept of line is $(C+1)/B$. Let the supporting vector on black circle lie on line $AX_1+ BX_2 - C = -1$. Hence slope of margin line is $-A/B$. Intercept of line is $(C-1)/B$.
- The white circles or points have features that obey $AX_1+ BX_2 - C \geq +1$, i.e., the white circles lie above the margin line $AX_1+ BX_2 - C = +1$. The black circles or points have features that obey $AX_1+ BX_2 - C \leq -1$, i.e., the black circles lie below the margin line $AX_1+ BX_2 - C = -1$.
- Consider any perpendicular line from a point on $AX_1+ BX_2 - C = +1$ to a point on $AX_1+ BX_2 - C = -1$. The slope of this perpendicular line is $-1/(-A/B) = B/A$. One such perpendicular line is $X_2 = [B/A]\ X_1$ or $AX_2 - BX_1 = 0$. The intersection of this perpendicular with line $AX_1+ BX_2 - C = +1$ is point with co-ordinates ($A[C+1]/[A^2+B^2]$, $B[C+1]/[A^2+B^2]$).
- The intersection of this perpendicular with line $AX_1+ BX_2 - C = -1$ is point with co-ordinates ($A[C-1]/[A^2+B^2]$, $B[C-1]/[A^2+B^2]$). Therefore, the distance between these two intersection points is the shortest distance between the two margin lines. This distance is

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

Support Vector Machine

- In the more general p-dimension (p>2), let vector $X_j = (X_1^{(j)}, X_2^{(j)}, X_3^{(j)}, \ldots, X_p^{(j)})$ for the j^{th} sample point in the training data set. $X_j \in R^p$. Let $W \in R^P$ be a p-dimensional vector representing coefficients of X in a hyperplane $W \bullet X - C$ = constant.
- Suppose one margin hyperplane is $W \bullet X - C = +1$ where all training sample points lying on or "above" (i.e., $W \bullet X - C > +1$) it have $Y_j = +1$.
- The other parallel margin hyperplane is $W \bullet X - C = -1$ where all training sample points lying on or "below" (i.e., $W \bullet X - C < -1$) it have $Y_j = -1$.
- Then for any point that lies on $W \bullet X - C = +1$ when $Y_j = +1$, $Y_j (W \bullet X_j - C) = 1$. And for any point that lies on $W \bullet X - C = -1$ when $Y_j = -1$, $Y_j (W \bullet X_j - C)$ also equals to 1. Hence, for any point X_j on the margin hyperplanes

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

- In general, for any point X_j of the training sample, $Y_j (W \bullet X_j - C) \geq 1$ for all $j \in [1,N]$ (4.6)
- For a general X, the Euclidean distance between point vector X_0 and X that lie on hyperplane $W \bullet X - C = +1$ is

$$\sqrt{(X_0 - X)(X_0 - X)} \quad \text{subject to } W \bullet X - C = +1.$$

If the line connecting X_0 and X is perpendicular to $W \bullet X - C = +1$, then X can be found by

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

where λ is Lagrange multiplier on constraint $W \bullet X - C - 1 = 0$.

Support Vector Machine

- Solving the quadratic problem (convex) with global minimum, first derivative w.r.t vector X gives $-2((X_0 - X) - \lambda W = 0$. This yields two more equations for solution, viz.,

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

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

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

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

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

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

- This is the margin length or width of the "street" between the two parallel hyperplanes. In the linear SVM problem, we maximize the margin or the width of the "street" $2/||W||$ to find the supporting vectors, hence also the separating hyperplane $W\bullet X - C = 0$ in the middle of the two parallel margin hyperplanes. This implies minimizing $||W||$.

Support Vector Machine

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

  where there are N constraints, hence N Lagrange multipliers $\lambda_i \geq 0$.

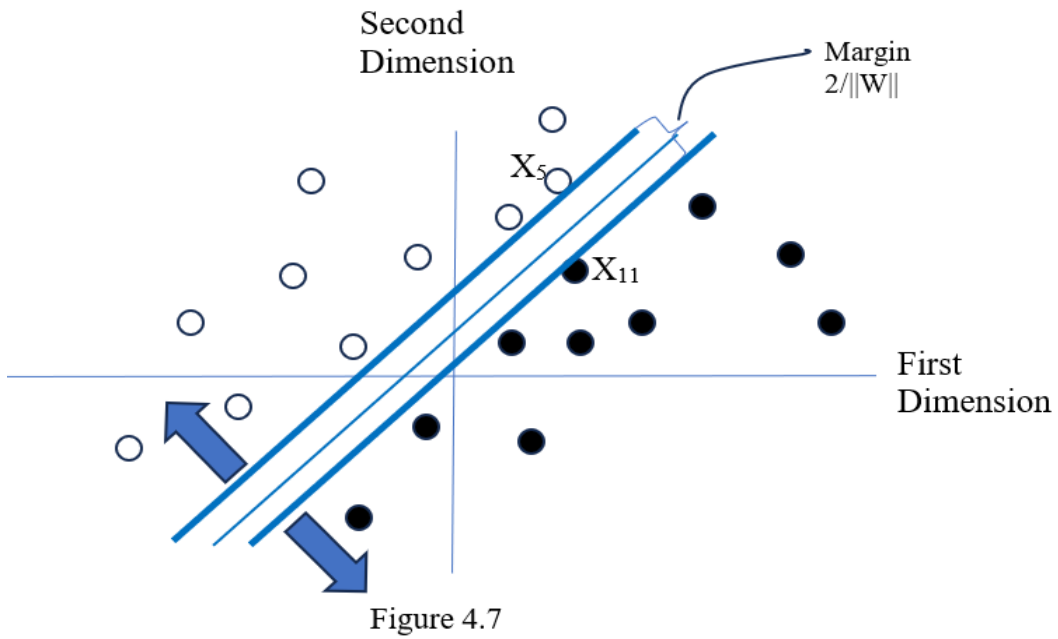


Figure 4.7

Figure 4.7 shows the new notations indicating two training sample points X_5 and X_{11} as the support vectors on the margin hyperplanes separating the two classes. X_5 lies on $W \bullet X - C = +1$ while X_{11} lies on $W \bullet X - C = -1$.

At these two points, constraint in Eq. (4.5): $Y_j (W \bullet X_j - C) - 1 = 0$ is satisfied.

Note that if support vectors $W \bullet X - C = +1$ or $W \bullet X - C = -1$, seen as points X_5 and X_{11} draw farther away from each other, like the other non-support vectors, as indicated by the two thick arrows trying to widen the "street", this creates a smaller objective function.

At the support vectors X_5 and X_{11}, λ_5 and $\lambda_{11} > 0$. For all the other non-constraining sample points X_j where $Y_j (W \bullet X_j - C) > 1$, $\lambda_j = 0$, so that $\lambda_j [Y_j (W \bullet X_j - C) - 1]$ would not affect the minimized objective.

Support Vector Machine

Without loss of generality, we solve

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

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

The Lagrangian simplifies to

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

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

$$\mathrm{W} - \sum_{i=1}^{N}\lambda_i\,Y_iX_i = 0 \qquad (4.8)$$

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

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

$$[Y_i\,(\mathrm{W}X_i - \mathrm{C}) - 1] = 0 \qquad (4.10)$$

for $\lambda_i > 0$.

Support Vector Machine

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

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

Using Eq. (4.9), the optimization becomes

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

subject to $\sum_{i=1}^{N} \lambda_i Y_i = 0$ and $\lambda_i \geq 0$.

Note that there are N first order conditions (derivatives w.r.t. λ_i) to solve for the N Lagrange multipliers, although strictly speaking, only the few associated with supporting vectors have non-zero λ_i's that need to be solved.

Solving Eq. (4.11) (possibly by numerical methods) will yield optimal $\hat{\lambda}_i$'s so optimal $\widehat{W} = \sum_{i=1}^{N} \hat{\lambda}_i Y_i X_i$ and $\hat{C} = \frac{1}{m} \sum_{k=1}^{m} (\widehat{W} X_k - Y_k)$ where m is the number of support vectors.

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

Eq. (4.11) can be expressed as $max_{\lambda_i} \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j Y_i Y_j K(X_i, X_j)$ in Eq. (4.12) where $K(X_i,X_j) = X_i \bullet X_j$ is called the linear kernel in the SVM.

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

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

$$Y_j (W \bullet X_j - C) - 1 < 0 \qquad (4.13)$$

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

Support Vector Machine

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

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

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

The solution of W, C gives rise to a soft-margin classifier, i.e., a classifier allowing for some misclassifications.

Larger $\alpha > 0$ implies tendency towards the hard-margin classifier as the penalty via α is larger for any violation $\zeta_i > 0$, allowing less misclassifications.

Lower α, on the other hand, imposes less penalty on misclassifications and allow more misclassifications $\sum_{i=1}^{N} \zeta_i$ but wider margins or smaller $||\mathrm{W}||$. (Note that wider margins lead to more violations.)

α is a regularization hyperparameter that is given or pre-determined, e.g., $\alpha = 1$. α provides some sort of tradeoff between "wider" margins and more violations or "narrower" margins and less violations. "Narrower" margins we saw earlier is less helpful toward predictions on test data or generalized data.

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

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

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

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

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

Expression (4.14) can also be written as

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

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

Support Vector Machine

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

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

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

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

for a given kernel K(·,·).

The SVM method does not explicitly compute the probability of a test point Z being in one class/type or another.

However, we can use a logit model to estimate the probability given the predicted class being +1 or −1 against the features used in SVM for the prediction.

The predicted probabilities can then be used to build the ROC curve given different probability thresholds.

Support Vector Machine

- Other SVM kernels that can be used for classification include the following. The polynomial kernel $K(X_i,X_j) = (\gamma X_i \bullet X_j + c)^d$ where the degree of the polynomial d, gamma γ, and c are hyperparameters.
- As an illustration, suppose point X_i is $(x_1^{(i)}, x_2^{(i)})$ with two dimensions. Let $K(X_i,X_j) = (X_i \bullet X_j + 1)^2$ where d = 2. Then $K(X_i,X_j) = (x_1^{(i)}x_1^{(j)} + x_2^{(i)}x_2^{(j)})^2 + 1 = 2\, x_1^{(i)}x_1^{(j)} + 2\, x_2^{(i)}x_2^{(j)} + 2\, x_1^{(i)}x_1^{(j)}x_2^{(i)}x_2^{(j)} + (x_1^{(i)}x_1^{(j)})^2 + (x_2^{(i)}x_2^{(j)})^2 + 1 = (\sqrt{2}\, x_1^{(i)}, (\sqrt{2}\, x_2^{(i)}, (\sqrt{2}\, x_1^{(i)}x_2^{(i)}, x_1^{(i)\,2}, x_2^{(i)\,2}, 1) \bullet (\sqrt{2}\, x_1^{(j)}, \sqrt{2}\, x_2^{(j)}, \sqrt{2}\, x_1^{(j)}x_2^{(j)}, x_1^{(j)\,2}, x_2^{(j)\,2}, 1)$. Clearly, the polynomial kernel now expands X_i from two to six dimensions. The higher dimension allows for easier linear separation; this is called the "kernel trick".
- We can let the dimension expansion (enlarging feature space) be transformation function $\phi(x_1^{(i)}, x_2^{(i)}) = (\sqrt{2}\, x_1^{(i)}, \sqrt{2}\, x_2^{(i)}, \sqrt{2}\, x_1^{(i)}x_2^{(i)}, x_1^{(i)\,2}, x_2^{(i)\,2}, 1)$. Hence polynomial kernel on $(x_1^{(i)}, x_2^{(i)})$ becomes linear kernel on $(\phi(x_1^{(i)}, x_2^{(i)}), \phi(x_1^{(j)}, x_2^{(j)}))$, i.e., polynomial $K(X_i,X_j) = \phi(x_1^{(i)}, x_2^{(i)}) \bullet \phi(x_1^{(j)}, x_2^{(j)}) = \phi(X_i) \bullet \phi(X_j)$.
- Without elaborating on some highly technical details, we state that there is a Mercer's Theorem 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.
- An example of popular non-linear kernels are the Gaussian radial basis function (RBF) $K(X_i,X_j) = \exp(-\gamma\, ||X_i - X_j||^2)$ or $\exp(-\gamma\, (X_i - X_j) \bullet (X_i - X_j))$ where $\gamma = 1/(2\sigma^2)$. Here γ (gamma) > 0 is a hyperparameter and σ need not be constrained to be standard deviation of the standardized features. In Sklearn, by default, γ = 'score' which is $1/(N\sigma^2)$. High γ means more curvature in the separating curved surface of the two classes.

Support Vector Machine

- 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.
- In general, when we apply a particular kernel function as in the constrained minimization in Eq. (4.14), we check for the performance results in prediction accuracy and the classification errors on the test sample data set. There is no need to explicitly determine or check the transformation function of $\phi(.)$ in $K(X_i,X_j) = \phi(X_i) \bullet \phi(X_j)$.
- The choice of what type of kernel depends on the nature of the data and the problem. Linear kernel is used (typically the one with soft constraint and applying hyperparameter α, or 'C', in sklearn) when the data is approximately linearly separable. Polynomial kernel is used when the data has a curved border. Gaussian kernel is used when the data appear to have complicated overlaps and no clear boundaries.

Support Vector Regression

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

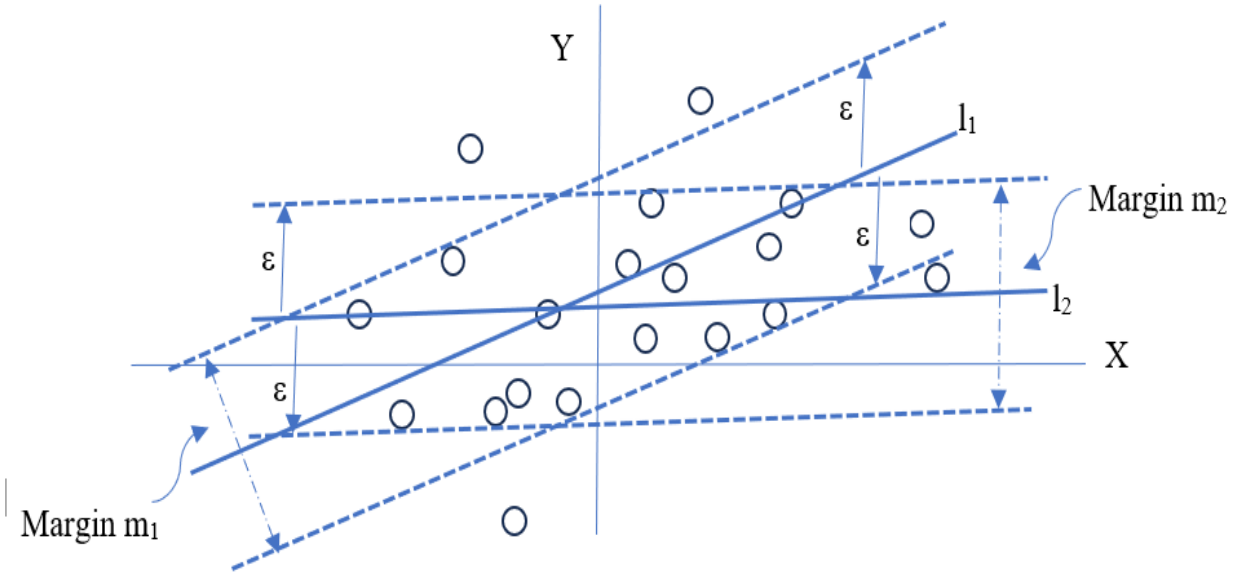


Figure 4.8

Just as in SVM, the SVR finds a maximum margin (equivalently minimum $\frac{1}{2}||W||^2$) where the margin hyperplanes have vertical displacements $\varepsilon > 0$ on either side of the fitted hyperplane in the middle between the two margin hyperplanes.

As seen in Figure 4.8, there are two possible fitted lines l_1 and l_2. Each has two margin hyperplanes with vertical displacements ε on either side. However, line l_2 has a wider margin than line l_1, i.e., $m_2 > m_1$. Hence, maximizing margin subject to displacements of epsilon $\varepsilon > 0$ is tantamount to finding a "flatter" fitted hyperplane.

Support Vector Regression

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

$$min_{W,C,\zeta_i} \ \tfrac{1}{2} W \bullet W + \alpha \sum_{i=1}^{N} \zeta_i \qquad (4.21)$$

subject to $| Y_i - (W \bullet X_j - C) | \leq \varepsilon$ and $\zeta_i \geq 0$ for i=1,2,..,N.

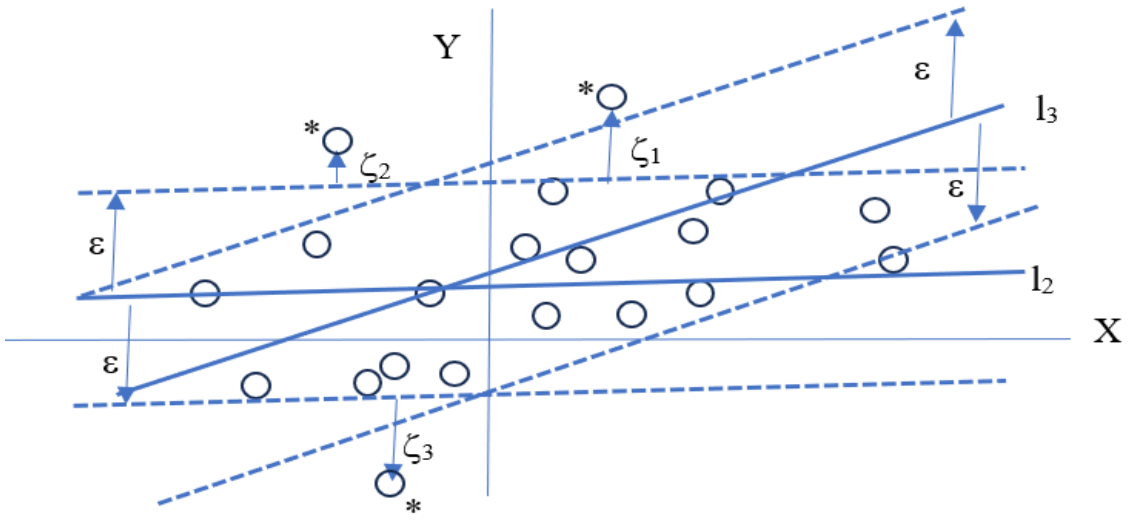


Figure 4.9

In Figure 4.9, a better fit with a lower objective function in Eq. (4.21) can be found by increasing hyperparameter α (or 'C', in sklearn) so that ζ_i deviations are penalized more and thus are kept smaller with tilting the slope of the fitted line to l_3. In l_3, sum of ζ_i deviations is smaller. Increasing epsilon ε would reduce sum of ζ_i deviations but when the increase is too large and no deviations exist, the fitted slope becomes flat or zero slope. Sometimes increasing epsilon ε may also cause prediction to deteriorate. Just as in SVM, the dual problem of finding the optimal fitted hyperplane in SVR can be formulated in terms of $(X_i \bullet X_j)$ as in Eq. (4.18). Nonlinear kernels $K(X_i,X_j) = \phi(X_i) \bullet \phi(X_j)$ as in Eq. (4.20) can also be used in SVR to obtain fitted hyperplane after the original data points are suitably transformed via ϕ(.).

Worked Example

- 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% of the transactions. Credit card fraudulent transactions occur when unauthorized person(s) somehow gain access to use the card to make purchases.
- According to the source, the data are numerical continuous variable features that are a PCA transformation from the original confidential client background information. There are 28 such features. In 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. Eventually, "Amount" is also added as a feature. The target or class/type variable "Type" is a dummy variable that takes the value 1 if the case is fraudulent, and 0 otherwise.
- 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
- Due to the huge type imbalance, it may be misleading just to consider Confusion Matrix accuracy, the AUC should also be considered.
- Other studies indicate 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 (client is target of fraudulent card transactions due to lost or stolen card or hacked card details). These features are used to predict Type 1 (Fraud occurrence) or Type 0 (no Fraud occurrence).
- 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.

Worked Example

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook

- 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%.
- 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 [10]:  # Normalization of all features

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

Next, the data is split into 80% (787 cases) for training and remaining 20% (197 cases) for testing.

We skip validation and assume the test result is done using optimized hyperparameter(s), and that the performance would be similar to using a subset for validation.

Naïve Bayes Algorithm

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook

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%.

Naïve Bayes Algorithm

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook

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

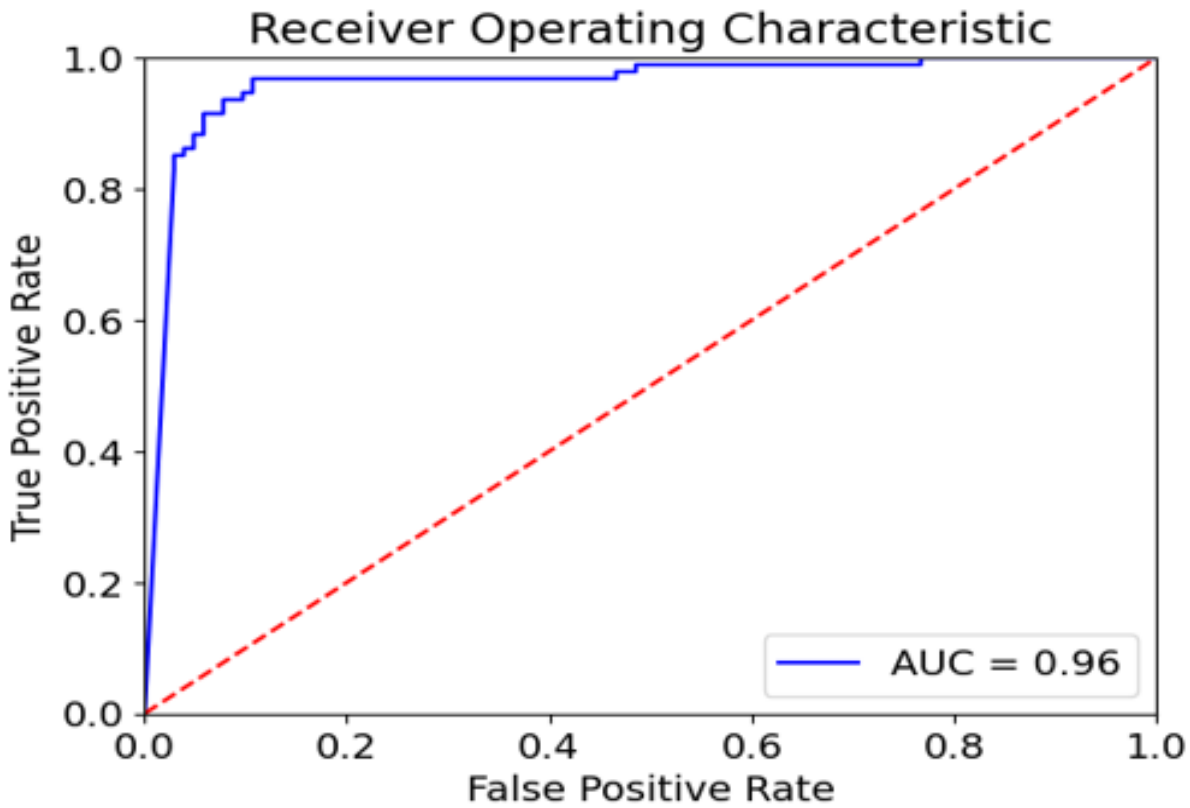
```
[[98  5]
 [11 83]]
```

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
```

Naïve Bayes Algorithm

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook



K-Nearest Neighbor Algorithm

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook

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
```

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook

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
```

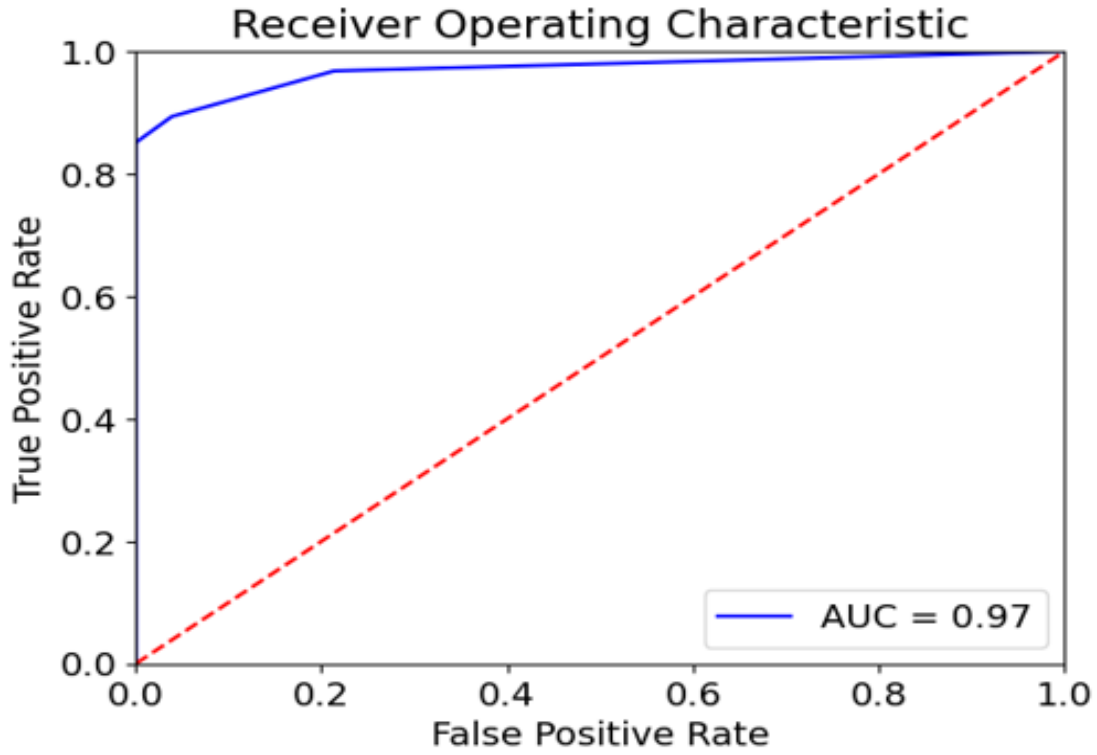
```
Test set score: 0.9289340101522843
```

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

```
Training set score: 0.9504447268106735
```

K-Nearest Neighbor Algorithm

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook



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

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook

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

Linear Kernel

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

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

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

In [59]: accuracy_score(y_test,y_pred_svm)

Out[59]: 0.9644670050761421
```

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook

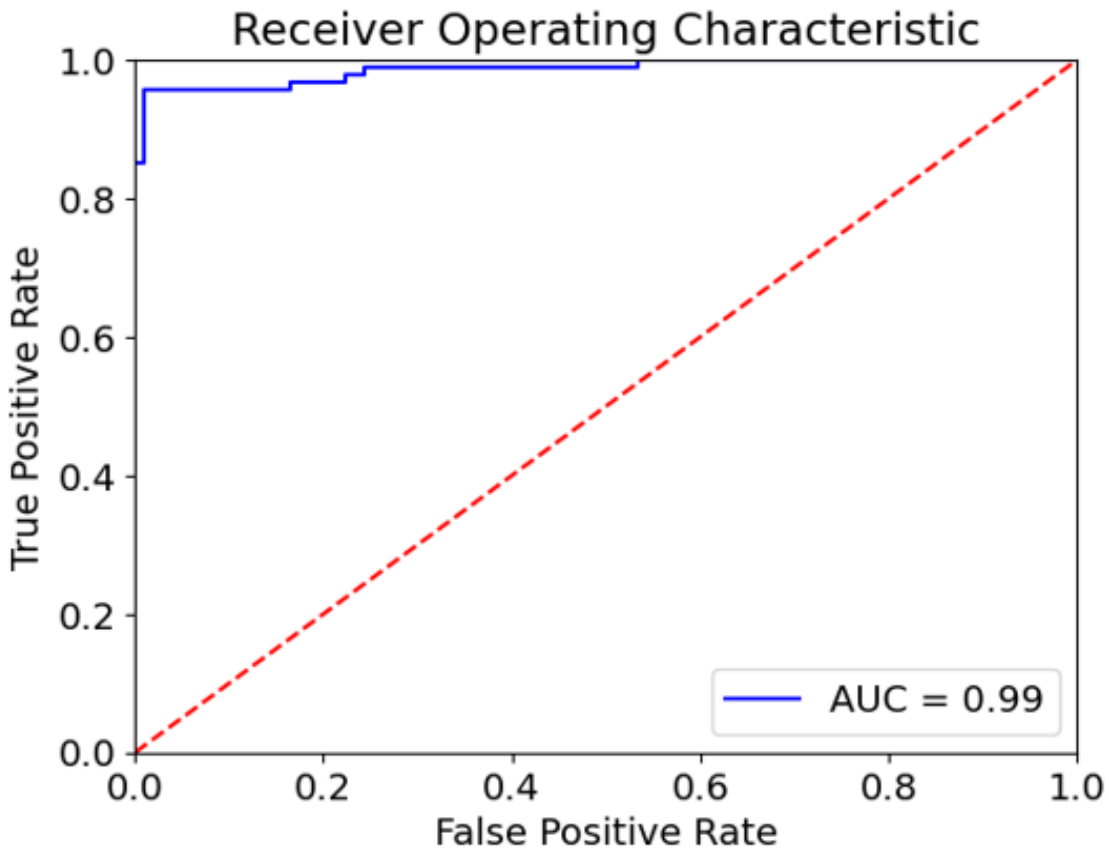
```
In [60]: from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred_svm))
```

```
              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.

Support Vector Machine

Please upload Chapter4-1.ipynb and follow the computing steps in Jupyter Notebook



Employing other kernels such polynomial, sigmoid, and rbf in the SVM produced lower accuracies and lower area under ROC curves.

Support Vector Regression

Please upload Chapter4-2.ipynb and follow the computing steps in Jupyter Notebook

- Data set 'california_housing.frame.csv' . Predicting MedHouseVal from 8 features.
- In code line [17], the features are preprocesses 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%.

- 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
```

Summary

- Naïve Bayes
  - Lower computational time. Scales up with multi-class predictions. Works well with many features/dimensions.
  - Features independence assumption can cause poor performance. Not natural extension to numerical prediction.
- kNN
  - Non-parametric easy to understand. Lazy training. Can extend to both classification and numerical prediction. Extendible to multi-class problem.
  - Requires feature standardizations. Algorithm can be slow when dimension increases.
- SVM
  - Contrary to kNN, suited to handling many features or high dimension. Dimension can be larger than sample size. Nonlinear kernels allow good performance in complex data patterns.
  - May require high computational time for large data set. Overfitting can occur easily when number of features is large relative to sample size. Performance depends importantly on regularization (hyperparameter tuning).

In-Class Practice Exercise (not graded):

Chapter4-3.ipynb

Repeat the exercise done in Chapter4-1 except now with the total data set of imbalanced data

End of Class