CHAPTER 4

UNIT IV

Naïve Bayes and Support Vector Machine

Syllabus

Bayes' Theorem, Naïve Bayes' Classifiers, Naïve Bayes in Scikit- learn - Bernoulli Naïve Bayes, Multinomial Naïve Bayes, and Gaussian Naïve Bayes.

Support Vector Machine (SVM) : Linear Support Vector Machines, Scikit - learn implementation- Linear Classification, Kernel based classification, Non-linear Examples. Controlled Support Vector Machines, Support Vector Regression.

4.1 Introduction to Naïve Bayes

- Naïve Bayes is a classification algorithm based on Bayes Theorem and the Maximum A Posteriori (MAP) hypothesis.
- It is useful for large data sets.
- It is also called as Idiot Bayes and Simple Bayes.
- Naïve Bayes makes an assumption that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is know as class conditional independence.
- The above assumption is made to simplify the computation and in this sense it's called as Naïve(it means the data attributes are independent).

4.1.1 Bayes Theorem

- Consider $X=\{x_1,x_2,x_3...x_n\}$ as a sample, whose components represent values made on a set of n attributes.
- In Bayes Theorem X is considered evidence. Let H be a Hypothesis, such that data X belongs to a class C.
- In classification, the goal is to determine P(H|X), the probability that the hypothesis H holds given the evidence (observed data sample X) .
- In other words, the probability that sample X belongs to class C.
- P(H|X) is called as a Posteriori probability of H conditioned on X, and is given as :

$$P(H|X) = \frac{P(X|H)\,P(H)}{P(X)}$$

Where P(H) is the a priori Probability which is independent of X.

P(X) is the a priori probability of X.

P(X|H) is the a posteriori probability of X conditioned on H.

- Let us understand this concept with the help of an example.

- Suppose we have a data set of customers. In this dataset age and income are the attributes and buys_computer as a class with values as Yes and No.
- Let us consider a sample X{ age : 45 years, income : Rs. 50,000}, we have to classify this unseen tuple as buys_computer = yes or buys_computer = No.
- P(H|X) is the probability that the customer X will buy a computer given his age and income, in our case age = 45 and income = Rs. 50,000.
- P(H) is the probability any customer will buy a computer regardless of his age and income, it is independent of X.
- P(X) is the probability from the data set that the customer is 45 years old and earns Rs. 50,000.
- P(X|H) is the probability that a customer , is 45 years old and he earns Rs. 50,000, given that he will buy a computer.

4.1.2 Maximum A Posteriori (MAP) Hypothesis

Based on Bayes Rule, we can compute the Maximum a Posteriori Hypothesis for the data:

$$h_{map} = \arg \max_{h \in H} P(h|D)$$

$$= \arg \max_{h \in H} \frac{P(D|h)\,P(h)}{P(D)} = \arg \max_{h \in H} P(D|h)\,P(h)$$

H : set of all Hypothesis

P(D) can be dropped as the probability of the data is constant(and independent of hypothesis)

4.1.3 Maximum Likelihood

- Now assume that all hypotheses are equally probable a priori, i.e., $P(h_i) = P(h_j)$ for all $h_i, h_j \in H$.
- This is called assuming a uniform prior. It simplifies computing the posterior.

$$h_{ML} = \arg \max_{h \in H} P(D|h)$$

- This hypothesis is called maximum likelihood hypothesis.

4.2 Naïve Bayes Classifier

Following is the working of Naïve Bayesian Classifier :

1. Let D be a training set of tuples along with their class labels. Each tuple is represented by an n-dimensional vector, $X = (x_1, x_2, x_3, ...x_n)$, consisting of n measurements on the tuple from n attributes $A_1, A_2, ...A_n$.
2. Consider there are m classes, $C_1, C_2, ...C_m$. Given a tuple X the classifier will predict that X belongs to a class having highest posterior probability conditioned on X which is given as

$$P(C_i \mid X) > P(C_j \mid X) \quad \text{for } 1 \cdot j \cdot m, j \neq i$$

Hence we maximize $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is called the maximum posteriori hypothesis. By Bayes theorem we have

$$P(C_i \mid X) = \frac{P(X|C_i)\,P(C_i)}{P(X)}$$

3. As P(X) is constant for all classes, only the $P(X|C_i)\,P(C_i)$ needs to be maximized. In case the prior probabilities of class are not known then it is likely that the probabilities are equal i.e. $P(C_1) = P(C_2) = P(C_3) = \ldots = P(C_m)$ and therefore only $P(X|C_i)$ needs to be maximized else $P(X|C_i)\,P(C_i)$ needs to be maximized. The class prior probabilities can be calculated as

$$P(C_i) = |C_{i,D}| / |D|$$

Where $|C_{i,D}|$ is the number of training tuples of class C_i in D

4. It is computationally expensive to compute $P(X|C_i)$, given a data set with many attributes. To reduce the computational complexity, naïve assumption of class conditional independence is made (No dependence relationships among attributes), thus

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

$$= P(x_1 \mid C_i) \times P(x_2 \mid C_i) \times \ldots\ldots \times P(x_n \mid C_i)$$

These probabilities can be estimated $P(x_1|C_i)$, $P(x_2|C_i)$, $P(x_3|C_i)$,...$P(x_n|C_i)$ from the training tuples. x_k is the value of attribute A_k for tuple X.

Computation of $P(X|C_i)$ is dependent on whether the attribute is categorical or continuous – valued.

(a) If A_k is Categorical : $P(x_k \mid C_i)$ is the number of tuples of class C_i in D having the value x_k for A_k, divided by $|C_{i,D}|$, the number of tuples of class C_i in D.

(b) If A_k is Continuous : A continuous valued attribute has a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}};$$

$$P(x_k \mid C_i) = g(x_k, \mu C_i, \sigma C_i)$$

Compute μC_i and σC_i, which are the mean (i.e., average) and standard deviation, respectively, of the values of attribute A_k for training tuples of class C_i. Use these values in above equation to calculate $P(x_k|C_i)$

5. For Prediction of class label of X ,$P(X|C_i)P(C_i)$ is evaluated for each class C_i . The classifier predicts that the class label of tuple X is the class C_i if and only if

$P(X|C_i)P(C_i) > P(X|C_j)P(C_j)$ for $1 \le j \le m, j \ne i$.

Which means the predicted class label is the class C_i for which $P(X|C_i)P(C_i)$ is the maximum.

4.2.1 Example of Naïve Bayesian Classifier

Predict a class label of an unknown sample using Naive Bayesian classification on the following training dataset from all electronics customer database.

Given a training set, we can compute the probabilities as follows :

- The classification problem may be formalized using a-posterior probabilities :

  $P(C|Y)$ is the probability that the sample tuple $Y = \langle y_1, \ldots, y_k \rangle$ is of class C.
- Assign to sample Y the class label C such that $P(C|Y)$ is maximal.

Age	Income	Student	Credit_rating	Class: buys_computer
<=30	High	No	Fair	No
<=30	High	No	Excellent	No
31...40	High	No	Fair	Yes
>40	Medium	No	Fair	Yes
>40	Low	Yes	Fair	Yes
>40	Low	Yes	Excellent	No
31...40	Low	Yes	Excellent	Yes
<=30	Medium	No	Fair	No
<=30	Low	Yes	Fair	Yes
>40	Medium	Yes	Fair	Yes
<=30	Medium	Yes	Excellent	Yes
31...40	Medium	No	Excellent	Yes
31...40	High	Yes	Fair	Yes
>40	Medium	No	Excellent	No

The unknown sample is x={ age= "<=30" , Income= "Medium", Student= "Yes", Credit rating= "Fair"}

Age	
P(<=30|Yes) = 2/9	P(<=30|No) = 3/5
P(31...40|Yes) = 4/9	P(31...40| No) = 0
P(>40|Yes) = 3/9	P(>40| No) = 2/5
Income	
P(High|Yes) = 2/9	P(High|No) = 2/5
P(Medium|Yes) = 4/9	P(Medium| No) = 2/5
P(low|Yes) = 3/9	P(low| No) = 1/5
Student	
P(No|Yes) = 3/9	P(No|No) = 4/5
P(yes|Yes) = 6/9	P(yes|No) = 1/5
Credit_Rating	
P(fair|Yes) = 6/9	P(fair|No) = 2/5
P(excellent|Yes) =3/9	P(excellent|No) = 3/5

P(Yes) = 9/14
P(No) = 5/14

An unseen sample X = < age= "<=30",Income= "Medium", Student= "yes", Credit rating= "Fair" >

$P(X|Yes) \cdot P(Yes) = P(Age"<=30"|Yes) \cdot P(Income= "Medium"|Yes) \cdot P(Student="yes"|Yes) \cdot P(Credit\_Rating="fair"|Yes) \cdot P(Yes)$

$= 2/9 \cdot 4/9 \cdot 6/9 \cdot 6/9 \cdot 9/14 = 0.028$

$P(X|No) \cdot P(No) = P(Age"<=30"|No) \cdot P(Income="Medium"|No) \cdot P(Student="yes"|No) \cdot P(Credit\_Rating ="fair"|No) \cdot P(No)$

$= 0.007$

Since 0.028 > 0.007, Therefore the naive Bayesian classifier predicts Buys computer = "Yes" for sample X

4.2.2 Properties of Bayes Classifiers

1. **Instrumentality**
   - The prior and likelihood can be updated dynamically with each training example.
   - Flexible and robust to errors.
2. **Combines prior knowledge and observed data**

   Given the training data, the prior probability of a hypothesis is multiplied with probability of the hypothesis.
3. **Probabilistic hypotheses**

   The output of Bayes classifier includes classification as well as probability distribution over all classes.
4. **Meta-classification**
   - The output of several classifiers can be combined together.
   - For example the probabilities of all classifiers predicted for a given class can be multiplied together.

4.3 Bayes in Scikit-Learn

- Based on the same number of different probabilistic distributions, scikit – learn provides implementation of three variants of Naïve bayes : Bernoulli Naïve Bayes, Multinomial Naïve bayes and Gaussian Naïve Bayes.
- Bernoulli naïve bayes is a binary distribution and this type is useful when a feature is present or absent.
- Multinomial Naïve Bayes is a discrete distribution and is useful when a feature needs to represented using a whole number.
- The Gaussian distribution is a continuous distribution and is characterised by mean and variance.

4.4 Bernoulli Naïve Bayes

- Consider X is a random variable having Bernoulli distribution.
- It can assume only two values (say for e.g. 0 and 1) and their probability is given as follows :

$$P(X) = \begin{cases} p & \text{if } X = 1 \\ q & \text{if } X = 0 \end{cases}$$

Where $q = 1 - p$ and $0 < p < 1$

Example : Implementation of Bernoulli Naïve Bayes in Scikit-Learn

Generating a dummy data set :

```
from sklearn.datasets import make_classification
nb_samples = 300
X, Y = make_classification(n_samples=nb_samples, n_features=2, n_informative=2, n_redundant=0)
```

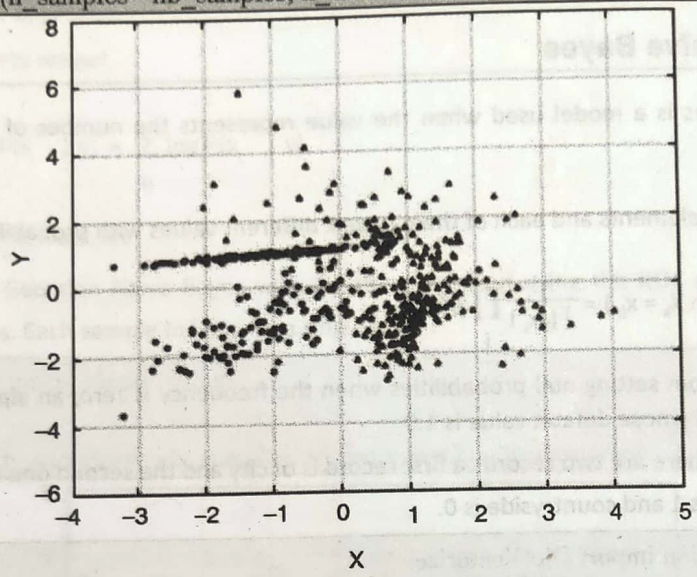

Fig. 4.4.1 : Dummy data generated

- Bernoulli naive Bayes needs binary feature vectors.
- The binarize parameter in BernoulliNB class allows using a threshold that can be used internally to transform the features to binary
- Let us use 0.0 as the binary threshold.

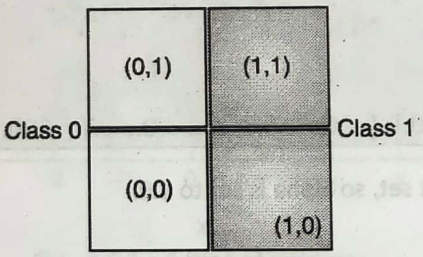

Fig. 4.4.2 : Internally binarized data

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection
import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25)
bnb= BernoulliNB(binarize=0.0)
bnb.fit(X_train, Y_train)
print(bnb.score(X_test, Y_test))
```

Output

```
0.85333333333333339
```

Checking for Prediction

```
data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
print(bnb.predict(data))
```

Output

```
array([0, 0, 1, 1])
```

4.5 Multinomial Naïve Bayes

- Multinomial Naïve Bayes is a model used when the value represents the number of occurrences of a term or its relative frequency.
- If a feature vector has n elements and each of them have k different values with probability p_k then

$$P(X_1 = x_1 \cap X_2 = x_2 \cap .... \cap X_k = x_k) = \frac{n!}{\prod_i x_i!} \prod_i p_i^{x_i}$$

- To prevent the model from setting null probabilities when the frequency is zero, an alpha parameter (called Laplace smoothing factor) is used whose default value is 1.0
- Example in scikit-learn. There are two records a first record is of city and the second one of countryside.
- The output class for city is 1 and countryside is 0.

```
from sklearn.feature_extraction import DictVectorizer
data = [ {'house': 100, 'street': 50, 'shop': 25, 'car': 100, 'tree': 20}, {'house': 5, 'street': 5, 'shop': 0, 'car': 10, 'tree': 500, 'river': 1} ]
dv = DictVectorizer(sparse=False)
X = dv.fit_transform(data)
Y = np.array([1, 0])
>>>X
#OUTPUT
array([[ 100., 100., 0., 25., 50., 20.], [ 10., 5., 1., 0., 5., 500.]])
```

- The term river is missing from the first set, so alpha is set to 1.0.
- Training the Multinomial Naïve Bayes

```
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()
mnb.fit(X, Y) MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Testing the model

```
test_data = data = [ {'house': 80, 'street': 20, 'shop': 15, 'car': 70, 'tree': 10, 'river': 1}, {'house': 10, 'street': 5, 'shop': 1, 'car': 8, 'tree': 300, 'river': 0} ]
mnb.predict(dv.fit_transform(test_data))
#Output
array([1, 0])
```

4.6 Gaussian Naïve Bayes

- Gaussian Naïve Bayes is used when the values are continuous whose probabilities can be modeled using a Gaussian Distribution.
- Conditional Probabilities are also Gaussian Distributed, mean and variance needs to be estimated of each of them using maximum likelihood approach.
- Using the Gaussian property we get,

$$L(\mu; \sigma^2; x_i \mid y) = \log \prod_k P(x_i^{(k)} \mid y) = \sum_k \log P(x_i^{(k)} \mid y)$$

k- refers to the sample in the data set

- **An example, comparing** Gaussian Naive Bayes with logistic regression using the ROC curves. The dataset has 300 samples with two features. Each sample belongs to a single class :

```
from sklearn.datasets import make_classification
nb_samples = 300
X, Y = make_classification(n_samples= nb_samples, n_features=2, n_informative=2, n_redundant=0)
```

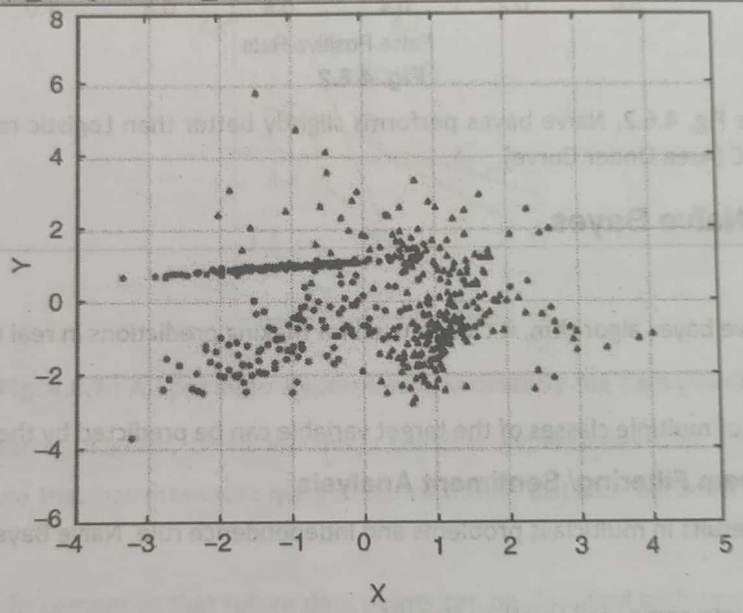

Fig. 4.6.1 : A plot of dataset

Training both the models and generating ROC curves

```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25) >>>gnb = GaussianNB()
gnb.fit(X_train, Y_train)
Y_gnb_score = gnb.predict_proba(X_test)
lr = LogisticRegression()
```

```
lr.fit(X_train, Y_train)
Y_lr_score = lr.decision_function(X_test)
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(Y_test, Y_gnb_score[:, 1])
fpr_lr, tpr_lr, thresholds_lr = roc_curve(Y_test, Y_lr_score)
```

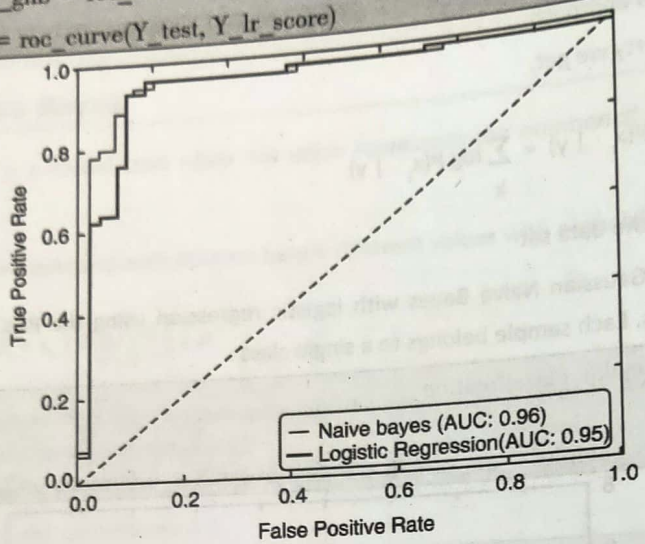

Fig. 4.6.2

As can be seen from the Fig. 4.6.2, Naïve bayes performs slightly better than Logistic regression, both the classifiers have similar accuracy and AUC (Area Under Curve).

4.7 Application of Naïve Bayes

1. **Real-time Prediction**

   Due to high speed of Naïve bayes algorithm, it can be used for making predictions in real time.

2. **Multi-class Prediction**

   The posterior probability of multiple classes of the target variable can be predicted by the algorithm.

3. **Text classification/ Spam Filtering/ Sentiment Analysis**

   - Due to their better results in multiclass problems and independence rule, Naïve Bayes classifier is suitable for text classification.
   - It has a high success rate compared to other algorithms.
   - Due to its success rates they are widely used in spam filtering and sentiment analysis applications.

4. **Recommendation System**

   Naïve Bayes can be combined together with collaborative filtering to make a recommendation system in machine learning and data mining to filter the unseen information and predict the users likes and dislikes.

Algorithm Advantages

- The algorithm performs well for multiclass prediction.
- The prediction of class on test data is faster and it is also easy to apply.
- A Naïve Bayes classifier performs better when compared to other classifiers like e.g. Logistic regression when the assumption of independence holds, it also requires less amount of training data.
- The algorithm performs better for categorical input variable(s) when compared to numerical variable(s). The numerical variable assumes a normal distribution.

Algorithm Disadvantages

- The model will be unable to make a prediction if the categorical variable has a category in test data, which was not available in training data.
- In such a case the model assigns a zero (0) probability. This is called as "Zero Frequency". This problem can be overcome with a smoothing technique. One of the simplest method used for smoothing is called Laplace estimation.
- The assumption of independent predictors in Naïve Bayes algorithm is a limitation as in real world it is almost impossible to get a set of predictors which are independent.

4.8 Introduction to Support Vector Machine (SVM)

- Support vector Machine is another method used for classification.
- It can classify both Linear as well as Non Linear Data.
- The objective of SVM is to find a hyperplane (A decision boundary separating the tuples of one class from another) in an N - dimensional space (where N represents the number of features) that distinctly classify the data points.

Example :

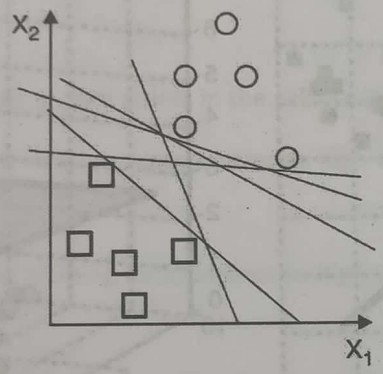

Fig. 4.8.1 : All possible Hyperplanes to classify the data points

- As we can see in the above diagram, there are many possible hyperplanes that could be chosen, but the main objective is to find a plane that has maximum margin i.e. maximum distance between data points of both classes as shown in the Fig. 4.8.2.
- Maximizing provides reinforcement so that future data points can be classified with more confidence.

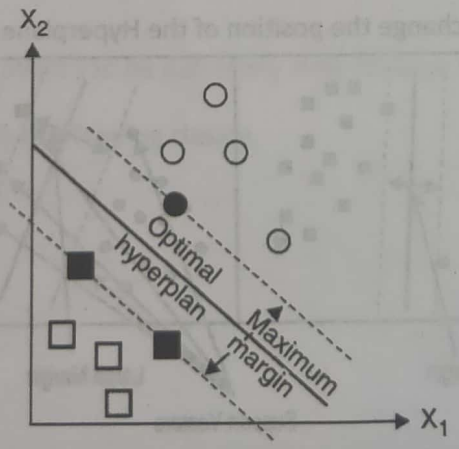

Fig. 4.8.2 : Optimal hyperplane having maximum margin

Working of Support Vector Machine

- SVM uses a Non Linear mapping to transform the original training data into a higher dimension.
- In this dimension it searches for a linear optimal separating hyperplane.
- It finds this hyperplane using support vectors (essentially training tuples) and margins (defined by support vectors).
- In SVM, If the output of the Linear function is greater than 1, it is identified as one class and if the output is -1 it is identified with another class.

Hyperplanes

- Hyperplanes are decision boundaries that are used to classify the data points.
- The dimension of the hyperplane depends depends upon the number of features.
  - Number of Input features is 2 : then hyperplane is a Line
  - Number of Input features is 3 : the hyperplane is a two dimensional plane
- Difficult to imagine a hyperplane when input features are more than 3.

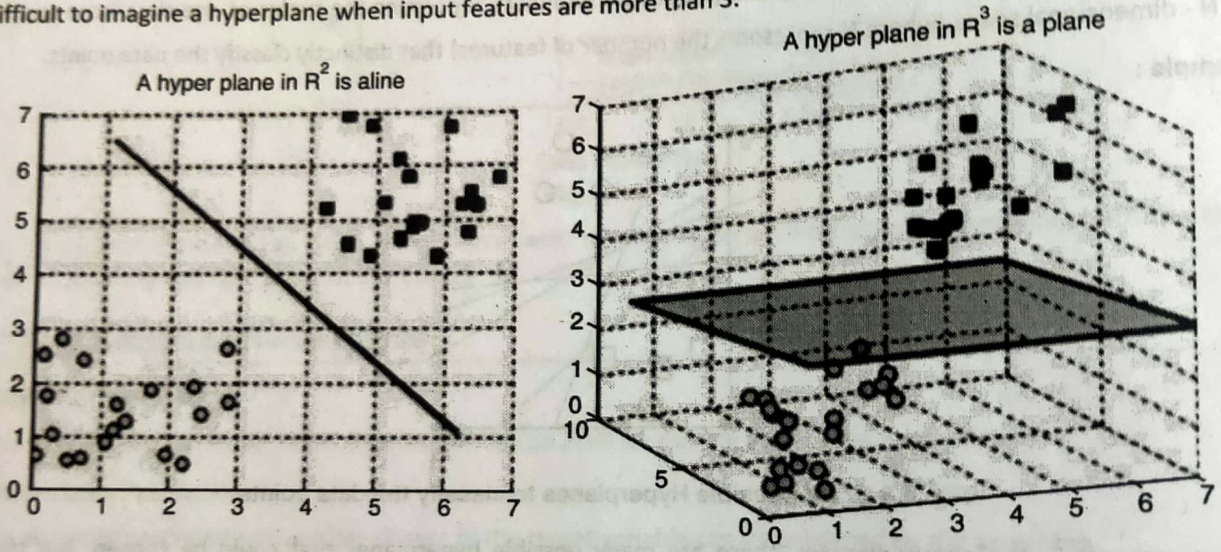

Fig. 4.8.3 : Hyperplane as a Line and a two dimensional plane

Support Vectors

- Support vectors are data points that are closer to hyperplane.
- These support vectors are used to maximize the margin of the classifier.
- Deletion of these support vectors will change the position of the Hyperplane.

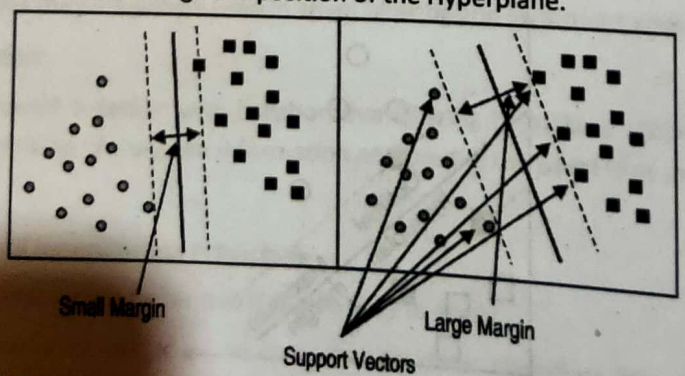

Fig. 4.8.4 : Support vectors and Margin separating the data points

4.8.1 Tuning Hyperparameters

Kernel

- Kernel is used to transform the given data into the required form.
- Different types of kernel functions may be used in Support vector machine, such as Linear, Polynomial, Radial Basis function and sigmoid kernel.
- Polynomial and RBF kernels compute the separation line in higher dimension, they are also useful for non-linear hyperplane.
- For separating the classes that are either curved or non-linear, more complex kernels can be used.

Regularization

- A penalty parameter called as C is used in Scikit-learn library as a regularization parameter.
- The term C represents misclassification or error term, which tells the SVM optimization by how much the error is bearable. This is how the trade-off between the decision boundary and misclassification term is controlled.
- A small-margin hyperplane is created by a small value of C and a larger value of C creates a larger-margin hyperplane.

Gamma

- The value of gamma decides the number of data points in the calculation of separating line.
- A low value of gamma will consider only nearby points whereas a high value of gamma will consider all the data points in the calculation.
- A low value of gamma will loosely fit the training data, whereas a high value will overfit the training data.

Advantages of SVM

- SVM is a highly accurate method due to the ability of modeling complex non-linear decision boundaries.
- When compared to other methods, SVM is less prone to overfitting.
- The support vectors of the model provide a description of the learned model.
- SVM can be used for prediction (numeric) and also as a classifier.

Disadvantages

- The training time of even the fastest SVMs can be extremely slow on large data sets.
- Less effective on noisier datasets with overlapping classes.

Applications

- Handwritten digit recognition.
- Object recognition.
- Speaker identification.
- Benchmark time-series prediction tests.

4.8.2 Problems Faced by SVM when used with Real Datasets

Unbalanced Data

- Success of SVM is very limited when the real world data sets are imbalanced, where the number of negative instances exceeds positive instances.
- These types of data sets may be found in applications like gene profiling, medical diagnosis and credit card fraud detection.
- In general classifiers perform poorly on imbalanced data sets, as they are designed to generalise from sample data and output the simplest hypothesis that best fits the data.
- Another factor includes making the classifier too specific which will result in making it sensitive to noise and more prone to learn an erroneous hypothesis, soft margin algorithm in SVM modifies the behaviour of the algorithm to make it more immune to noisy instances, such approaches work well with balanced data sets but in case of imbalanced data sets, the simplest hypothesis is that it classifies every instance as negative.

Multilabel classification

- SVM was originally designed for binary classification.
- Several approaches have been proposed to extend SVM to multiclass classification by combining several binary classifiers.
- It is computationally expensive to solve a multiclass problem than a binary problem with same number of data.

Large scale data

- The time an memory requirements of SVM surge with an increase in data size which make it impractical to be use for a moderate problem as the data reaches to the extent of hundreds of thousands.
- To address this problem literature has proposed algorithms such as clustering-based SVM, which removes the clustered data points far away from support vectors.

Semi-supervised learning

- In semi-supervised learning, the technique makes use of both labelled as well as unlabelled data.
- The goal is to employ large collection of unlabelled data with labelled data to improve generalisation performance
- Optimization is a challenge in SVM for semi-supervised learning.
- The technique can get trapped in a bad local optima.

4.9 Linear Support Vector Machines

- Let us consider a two class problem where the classes are linearly separable.
- Consider D as the data set which can be given by

$(X_1, y_1), (X_2, y_2), \ldots, (X_{|D|}, y_{|D|})$,

Where X_i is the set of training tuples and y_i the associated class labels which can take + 1 or – 1 as its values.

- The Linearly separable classes as shown in the Fig. 4.9.1.

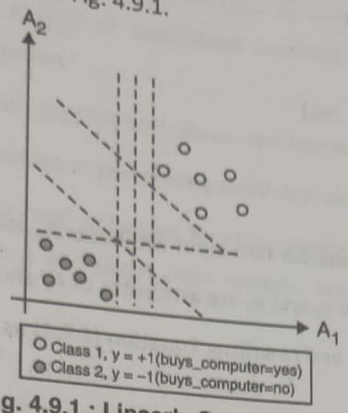

Fig. 4.9.1 : Linearly Separable Data

- Goal here is to find the best possible separating line (considering data as 2D) having minimum classification error on previously unseen tuple.
- Incase if the data is 3D, the goal would be to find the best separating hyperplane with minimum classification error.
- So to Generalise the concept for N dimensions, the Goal is to find the best Hyperplane.
- Support Vector Machine works by finding the maximum marginal hyperplane.

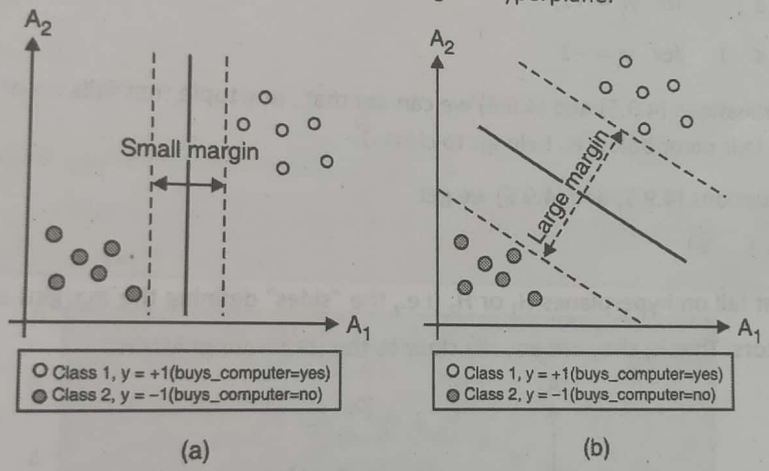

Fig. 4.9.2 : Two possible hyperplanes separating the data

- From Fig. 4.9.2 we can see that there are two possible hyperplanes separating the data, one with smaller margin and the other with larger margin.
- The hyperplane with larger margin can classify the data more better in the future as compared to the smaller margin hyperplane.
- This is the reason SVM searches the hyperplane with largest margin, that is Maximal Margin Hyperplane (MMH).

Definition of Margin : It is the shortest distance from a hyperplane to one side of its margin is equal to the shortest distance from the hyperplane to the other side of its margin, where the "sides" of the margin are parallel to the hyperplane.

...(4.9.1)

- A separating hyperplane can be given as

$$W \cdot X + b = 0$$

W : It is a weight vector, $W = \{w_1, w_2, w_3, \ldots w_n\}$

n : It is the number of attributes

b : It is a scalar (called as bias)

- To understand the classification let us consider two input attributes A_1 and A_2
- Training tuples are 2D, $(X = (x_1, x_2))$ where x_1 and x_2 are attributes of A_1 and A_2
- Considering w_0 as the additional weight, and rewriting Equation (4.9.1) as

$$w_0 + w_1 x_1 + w_2 x_2 = 0 \quad \ldots(4.9.2)$$

- Any point that lies above the hyperplane satisfies the following equation

$$w_0 + w_1 x_1 + w_2 x_2 > 0 \quad \ldots(4.9.3)$$

- And any point that lies below the hyperplane satisfies the following equation

$$w_0 + w_1 x_1 + w_2 x_2 < 0 \quad \ldots(4.9.4)$$

- Adjusting the weight, and hyperplanes defining the sides of the margin can be written as

$$H_1 : w_0 + w_1 x_1 + w_2 x_2 \geq 1 \quad \text{for } y_i = +1, \quad \ldots(4.9.5)$$

$$H_2 : w_0 + w_1 x_1 + w_2 x_2 \leq -1 \quad \text{for } y_i = -1 \quad \ldots(4.9.6)$$

From the above two equations (4.9.5) and (4.9.6) we can say that , any tuple that falls on or above H_1 belongs to class + 1 and any tuple that falls on or below H_2 belongs to class -1.

Combining the two equations (4.9.5) and (4.9.6) we get

$$Y_i (w_0 + w_1 x_1 + w_2 x_2) \geq 1, \quad \forall i \quad \ldots(4.9.7)$$

- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the "sides" defining the margin) satisfy Equation (4.9.7) and are called support vectors. That is, they are equally close to the (separating) MMH.

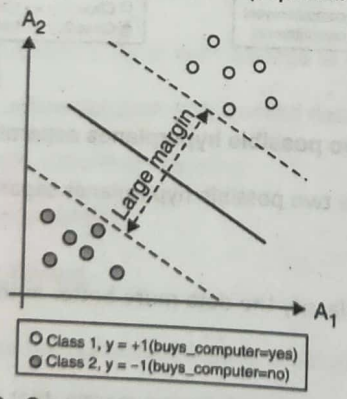

Fig. 4.9.3 : Support vectors with thick boundary

- The distance from the separating hyperplane to any point on H_1 is $1/||W||$, where $||W||$ is the Euclidean norm of $\sqrt{W}$ W. By definition, this is equal to the distance from any point on H_2 to the separating hyperplane. Therefore, the maximal margin is $2/||W||$.

4.10 Scikit-Learn Implementation

- A Support Vector Machine (SVM) is a type of supervised machine learning classification algorithm used for classification, regression and outlier detection.
- The support vector machines in scikit-learn support both dense and sparse sample vectors as input.
- SVC, NuSVC and LinearSVC are classes capable of performing multiclass classification on a dataset.
- SVC and NuSVC implements "one-against-one" approach for classification , on the other hand LinearSVC implements "one-vs-the-rest" multiclass strategy, thus training n class models. Incase of only two classes only one model is trained.

4.10.1 Linear Based Classification

For example In Scikit-learn, a simple classification task in which two classes of points are well separated.

Standard Imports

```
%matplotlib inline
import numpy as np import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set()

from sklearn.datasets.samples_generator import make_blobs
X, y =make_blobs(n_samples=50, centers=2,random_state=0, cluster_std=0.60) plt.scatter(X[:, 0], X[:, 1],
                    c=y, s=50, cmap='autumn');
```

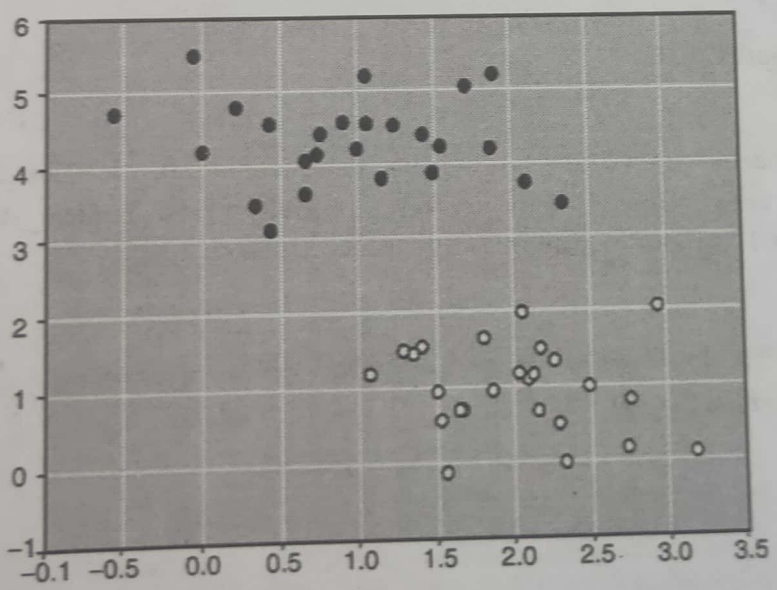

Fig. 4.10.1 : Plot showing data which is linearly separable

Draw Lines that can divide the data into two separate classes

```
xfit=np.linspace(– 1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plt.plot([0.6],[2.1],'x',color='red',markeredgewidth=2,markersize=10)  for m, b in [(1, 0.65),(0.5, 1.6),(– 0.2,2.9)]:
plt.plot(xfit, m *xfit+ b, '– k')
```

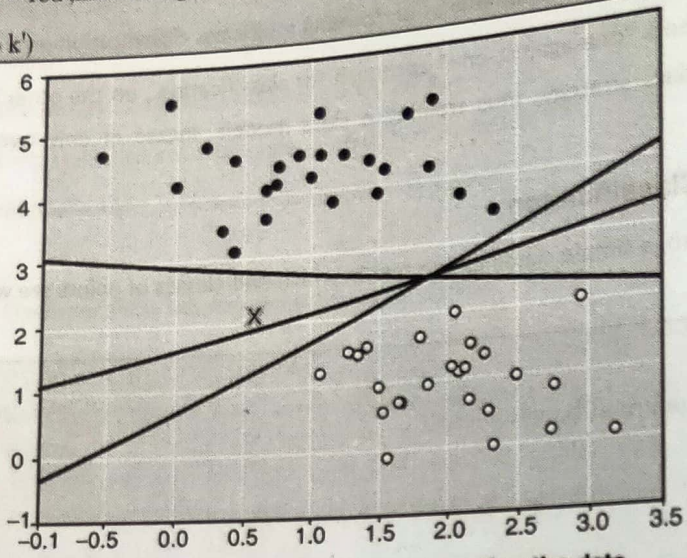

Fig. 4.10.2 : Plot showing Lines separating the data

Rather than simply drawing a zero-width line between the classes, we can draw around each line a *margin* of some width, up to the nearest point.

Fitting a support vector machine

Use Scikit-Learn's support vector classifier to train an SVM model on this data.

```
from sklearn.svm import  SVC
# "Support vector classifier"
model= SVC(kernel='linear', C=1E10)
model.fit(X, y)
SVC(C=10000000000.0, cache_size=200, class_weight=None, coef0=0.0,   decision_function_shape=None,
degree=3, gamma='auto', kernel='linear',   max_iter=-1, probability=False, random_state=None,
shrinking=True,   tol=0.001, verbose=False)

def plot_svc_decision_function(model, ax=None, plot_support=True):
"""Plot the decision function for a 2D SVC"""
if ax isNone:
ax=plt.gca()
xlim=ax.get_xlim()
ylim=ax.get_ylim()
# create grid to evaluate model
```

```
x=np.linspace(xlim[0], xlim[1], 30)
y=np.linspace(ylim[0], ylim[1], 30)
Y, X =np.meshgrid(y, x)
xy=np.vstack([X.ravel(), Y.ravel()]).T
P =model.decision_function(xy).reshape(X.shape)
# plot decision boundary and margins
ax.contour(X, Y, P, colors='k',
    levels=[-1, 0, 1], alpha=0.5,
      linestyles=['--', '-', '--'])
# plot support vectors
if plot_support:
    ax.scatter(model.support_vectors_[:, 0],
    model.support_vectors_[:, 1],
    s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);


model.support_vectors_
```

Output

```
array([[ 0.44359863,  3.11530945],       [ 2.33812285,  3.43116792],       [ 2.06156753,  1.96918596]])
```

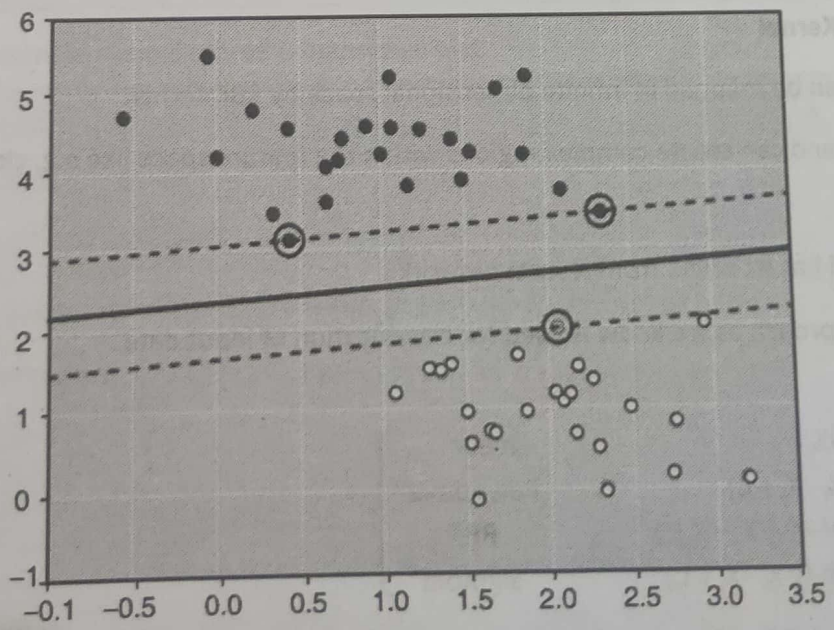

Fig. 4.10.3 : Support vectors

4.10.2 Kernel Based Classification

- Some problems cannot be solved using Linear hyper plane as shown in following examples.
- In this type of cases, SVM uses a kernel trick to transform the input space to a higher dimensional as shown in the Fig. 4.10.4(b).
- The data points are plotted on x-axis and z-axis, now these points can be easily segregated.

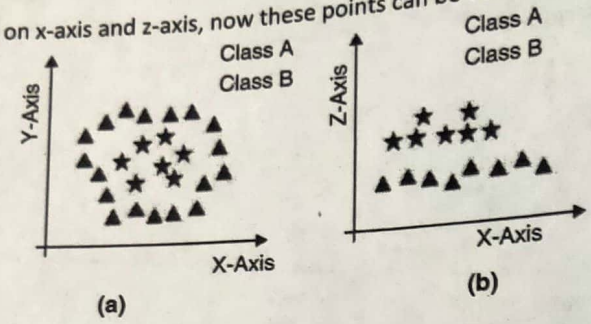

Fig. 4.10.4

- The kernel takes a low dimensional space and transforms it into higher dimensional space.
- It converts a non-separable problem into a separable problem by adding more dimensions to it.
- This is most useful in non-linear separation problem.
- Following are the different kernels used in SVM :

1. **Linear Kernel**
   - A Linear kernel is a dot product of any two given observations.
   - The product between two vectors is the sum of the multiplication of each pair of input values.
2. **Polynomial Kernel**
   - A more generalized form of Linear kernel is Polynomial kernel.
   - The polynomial kernel can differentiate between curved or nonlinear input space.
3. **Radial Basis Function Kernel**
   - The input space can be mapped in infinite dimensional space by RBF kernel.
   - It is a local kernel and can create complex regions within the feature space like e.g. closed polygons in 2D space.
4. **Sigmoid Kernel**
   - The sigmoid kernel has its origin from Neural networks.
   - Neural network approach as we know is used for classification of input data.

Kernel Functions

$$K(X_i, X_j) = \begin{cases} X_i \cdot X_j & \text{Linear} \\ (\gamma X_i \cdot X_j + C)^d & \text{Polynomial} \\ \exp(-\gamma \mid X_j - X_j \mid^2) & \text{RBF} \\ \tanh(\gamma X_i \cdot X_j + C) & \text{Sigmoid} \end{cases}$$

where $K(X_i, X_j) = \phi(X_i) \cdot \phi(X_j)$

4.10.3 Non Linear Examples

A Non Linear example

```
from sklearn.datasets.samples_generator import make_circles
X, y =make_circles(100, factor=.1, noise=.1)
clf= SVC(kernel='linear').fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf, plot_support=False);
```

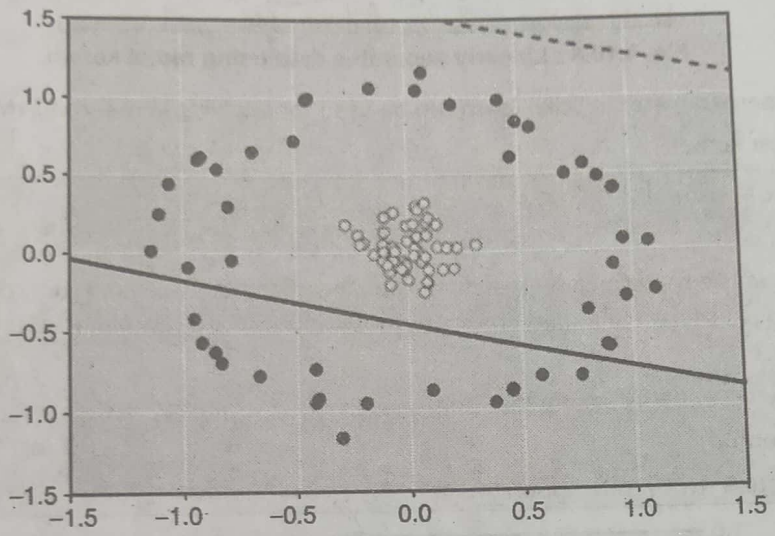

Fig. 4.10.5 : Non Linear separable data

From the Fig. 4.10.5 it is clear that the data is not linearly separable, for this we can use a kernel. (e.g. radial basis function).

```
r=np.exp(– (X **2).sum(1))
```

Lets visualize this extra dimension using a three dimensional plot

```
from mpl_toolkits import mplot3d
def plot_3D(elev=30, azim=30, X=X, y=y):
ax=plt.subplot(projection='3d')
ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=50, cmap='autumn')
ax.view_init(elev=elev, azim=azim)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('r')
interact(plot_3D, elev=[–90, 90], azip=(–180, 180),
X=fixed(X), y=fixed(y));
```

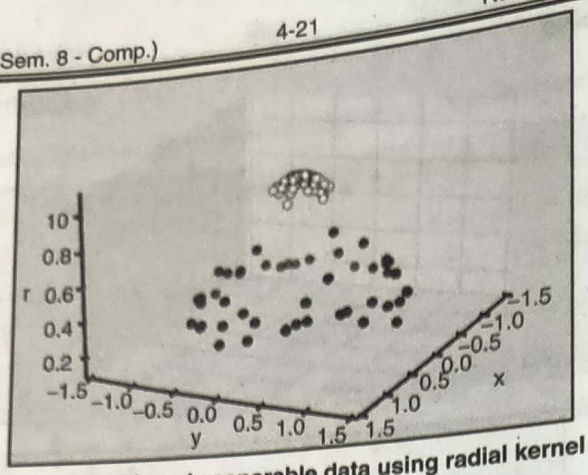

Fig. 4.10.6 : Linearly separable data using radial kernel

The kernel model hyperparameter in Scikit-learn can be used for applying kernalized SVM. i.e. for changing the Linear kernel to RBF or some other kernel.

```
clf= SVC(kernel='rbf', C=1E6) clf.fit(X, y)

SVC(C=1000000.0, cache_size=200, class_weight=None, coef0=0.0,  decision_function_shape=None,
degree=3, gamma='auto', kernel='rbf',  max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=300, lw=1, facecolors='none');
```

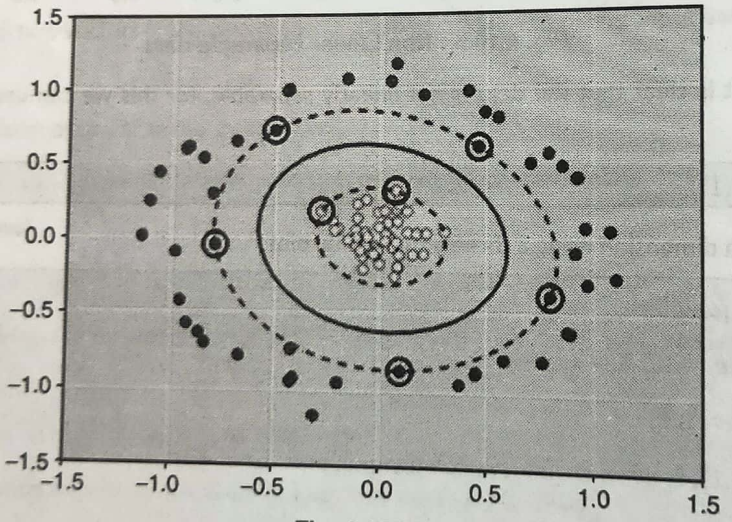

Fig. 4.10.7

4.11 Controlled Support Vector Machines

- While working with real data sets, SVM classifier can result in many support vectors, which can slow down the whole process.
- To find a tradeoff between precision and the number of support vectors, scikit learn implementation provides NuSVC implementation where the parameter nu(bounded between 0 – not included-and 1) can be used to control the number of support vectors.

4.12 Support Vector Regression

- The method of Support Vector machines can be extended to solve regression problems. This is called as Support Vector Regression.
- As discussed above the model produced by Support Vector machine for classification uses only a subset of training data, as the cost function for building the model does not consider the data that lie beyond the margin.
- Similarly the model produced by Support vector regression also depends on a subset of training data as the cost function for building the model ignores any data close to the model prediction.
- In Scikit-learn, there are three different implementations of Support vector regression SVR, NuSVR and LinearSVR.
- LinearSVR is faster than SVR but only considers Linear Kernels.
- NuSVR implements slightly different from SVR and LinearSVR.
- Example of SVR in Scikit-learn

```
from sklearn import svm
>>>X = [[0, 0], [2, 2]]
>>>y= [0.5, 2.5]
>>>clf=svm.SVR()
>>>clf.fit(X, y)
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,gamma='auto_deprecated', kernel='rbf',
max_iter=-1, shrinking=True,tol=0.001, verbose=False)
>>>clf.predict([[1, 1]]) array([1.5])
```

Review Questions

Q.1 Explain Bayes Theorem.

Q.2 Explain Naïve Bayes classifier.

Q.3 Explain the different variants of Naïve Bayes in scikit learn library.

Q.4 Explain Support vector Machines.

Q.5 Explain Linear support vector machine and its implementation in Scikit learn.

Q.6 Explain Linear and kernel based classification in Support vector machines.

Q.7 Explain Support vector regression.

❑❑❑