Lecture 8 - Feb 5

Classification
Bayes and Naïve Bayes Classifiers

Reading

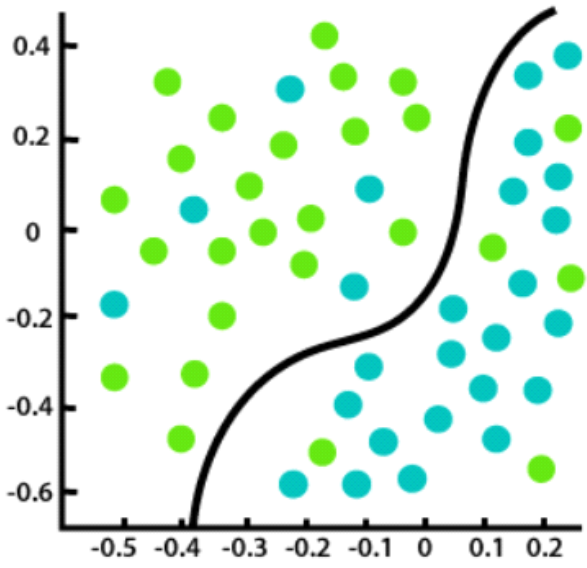
Week 5 Notes in GitHub

Data Mining and Machine Learning
Ch 18 Probabilistic Classification
Ch 22 Classification Assessment

Upcoming Deadlines

Classification

Saturday, February 5, 2022 1:13 AM

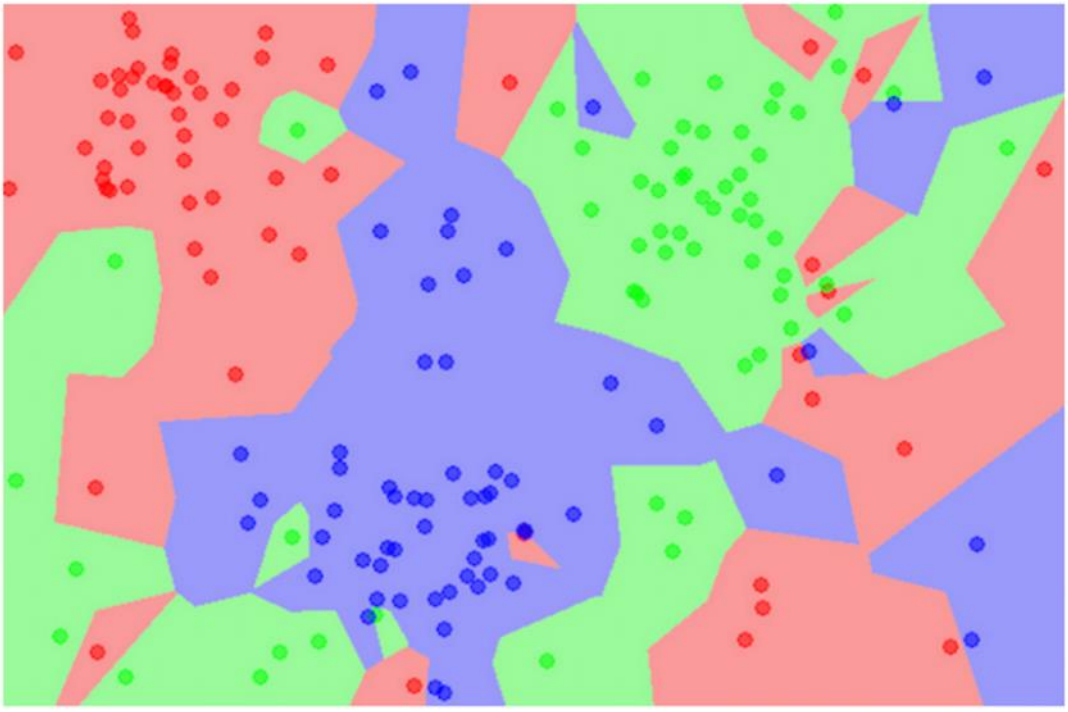


Classification

*Given a (high-dimensional) input point, predict **categorical** outputs*

Mathematically, a classifier is a function $M: \mathbb{R}^d \to \{c_1, \ldots, c_k\}$ mapping input datapoints to categorical output (a class)

$$\hat{y} = M(x)$$



The colored points are labeled examples and the $\mathbb{R}^2$ space is colored by the class to be assigned to points in different regions. (image from Wikipedia)

A classifier can be constructed in many different ways that we will cover in the next several weeks.

In supervised classification, we use a data set of known datapoints x_i and target classes y_i to train a classifier.

Math Required	**Classification Methods**	**Practical ML Ideas**
Probability and statistics Maximum likelihood Gaussian distribution	Probabilistic classification (Bayes classifier, logistic regression) Linear/quadratic classifiers (LDA/QDA, SVMs) Kernel methods (k-nearest neighbors) Support vector machines Decision trees	Cross-validation Interpretable ML

Classifiers

Wednesday, February 9, 2022 3:38 PM

A classifier $M: \mathbb{R}^d \to \{c_1, c_2, \ldots, c_k\}$

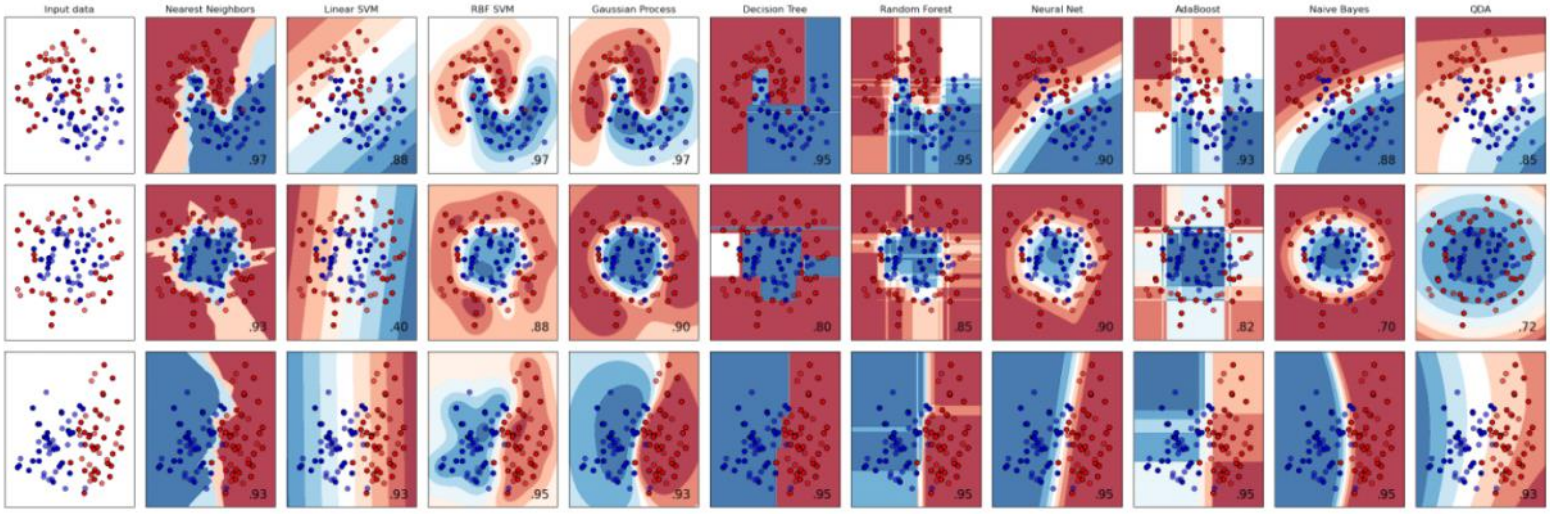
datapoints classes

$\hat{y}_i = M(x_i)$ - predicted class

Usually, we have a dataset of n points $x_1, \ldots, x_n$ with the corresponding class labels $y_1, y_2, \ldots, y_n$

goal: Construct M so that $\hat{y}_i = M(x_i) = y_i$ for all i

Different Classifiers Make Decisions Very Differently…



The Bayes classifier tries to estimate posterior probabilities that a point x belongs to each class.

$$\mathbb{P}(c_i|x) \text{ for } i=1,2,\ldots,k$$

and predicts x's class is the highest one

$$\hat{y} = \arg\max_{c_i} \mathbb{P}(c_i|x)$$

As its name suggests, it uses Bayes' Theorem to estimate these posterior probabilities

$$\hat{y} = \arg\max_{c_i} \mathbb{P}(c_i|x) = \arg\max_{c_i} \left\{ \frac{\mathbb{P}(x|c_i)\mathbb{P}(c_i)}{\mathbb{P}(x)} \right\} = \arg\max_{c_i} \left\{ \mathbb{P}(x|c_i)\mathbb{P}(c_i) \right\}$$

$\mathbb{P}(x)$ term is independent of c_i, so it does not matter...

Problems we don't know the prior probabilities $\mathbb{P}(c_i)$ or $\mathbb{P}(x|c_i)$.

Assumption 1: $\mathbb{P}(c_i)$ correspond to the proportion of datapoints dataset coming from class c_i

if there are n_i points in c_i, $\hat{\mathbb{P}}(c_i) = \frac{n_i}{n}$

... not required any other rule for computing prior

if there are ... 1

Note: this assumption is not required, any other rule for computing prior probabilities could work. Common to use prior knowledge or a previous state of an iterative process.

1

Assumption 2: points in class c_i have a multivariate normal distribution with parameters x_i, Σ_i,

$$f_i(x) = f_i(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma_i|}} \exp\left(-\frac{(x_i-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)}{2}\right)$$

$P(x|c_i) \approx \Sigma f_i(x)$ for a small ε

$P(x|c_i) \propto f_i(x)$

parameters μ_i, Σ_i must be estimated from the dataset (sample), so this is called parametric estimation

For each class, $\hat{\mu}_i = \frac{1}{n}\sum_{\text{class } i} x_j$, $\hat{\Sigma}_i = \frac{1}{n_i}\bar{D}_i^T\bar{D}_i$

($\hat{\mu}_i$: class sample mean vector; $\bar{D}_i$: centered data from class i)

Note: non-normal or even non-parametric estimation can be done, and we will study other ideas later.

With these assumptions, the Bayes classifier becomes

$$\hat{y} = \arg\max_{c_i}\{P(x|c_i)P(c_i)\} = \arg\max_{c_i}\{f_i(x|\mu_i, \Sigma_i)\cdot n_i\}$$

Algorithm: Bayes' Classifier

Algorithm: Bayes Classifier

train(D, Y):

for class $i = 1, \ldots, k$:

- $D_i \leftarrow \{x_j^T \mid y_j = c_i\}$ — class-subsets
- $n_i \leftarrow |D_i|$ — class sizes
- ~~$\hat{P}(c_i) \leftarrow \frac{n_i}{n}$~~ — ~~prior probabilities~~ (don't really need it with assumption 1)
- $\hat{m}_i \leftarrow \frac{1}{n_i} \sum_{D_i} x_j$ — class sample means
- $\bar{D}_i \leftarrow D_i - 1 \hat{m}_i^T$ — centered class subsets
- $\hat{\Sigma}_i \leftarrow \frac{1}{n_i} \bar{D}_i^T \bar{D}_i$ — class sample covariance matrix

return ~~$\hat{P}(c_i)$~~ n_i, $\hat{m}_i$, $\hat{\Sigma}_i$ for $i = 1, \ldots, k$

predict(x):

$\hat{y} \leftarrow \underset{c_i}{\operatorname{argmax}} \{ f(x, \hat{m}_i, \hat{\Sigma}_i) n_i \}$ — Could use $\hat{P}(c_i)$ — find class with max posterior probability

return $\hat{y}$

Bayes requires us to estimate

$$m_1, m_2, \ldots, m_k \in \mathbb{R}^d$$

$$\Sigma_1, \Sigma_2, \ldots, \Sigma_k \in \mathbb{R}^{d \times d}$$

$\Sigma_1, \Sigma_2, \ldots, \Sigma_k \in \mathbb{R}^{d \times d}$

$O(d^2)$

The Naive Bayes classifier assumes data columns are independent

$$\mathbb{P}(X_1 \in A_1, X_2 \in A_2, \ldots, X_n \in A_n) = \mathbb{P}(X_1 \in A_1) \cdot \mathbb{P}(X_2 \in A_2) \cdots \mathbb{P}(X_k \in A_k)$$

$$\Rightarrow Cov(X_j, X_i) = 0 \Rightarrow \Sigma = \begin{pmatrix} \sigma_1^2 & & \\ & \sigma_2^2 & \\ & & \ddots \\ & & & \sigma_d^2 \end{pmatrix}$$

Implementing the Bayes Classifier

With the ideas of how the Bayes classifier works, we can now implement it, but let's first import some libraries.

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sn

from scipy.stats import multivariate_normal
from sklearn import datasets
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import scipy
from tensorflow.keras.datasets import mnist

# increase the width of boxes in the notebook file (this is only cosmetic)
np.set_printoptions(linewidth=180)
```

```
class BayesClassifier:
    def fit(self, X, Y):
        # find the unique labels
        uniqueY = np.unique(Y)

        # find the dimensions
        n = X.shape[0]
        self.d = X.shape[1]
        self.k = uniqueY.shape[0]

        # initialize the outputs
        self.prior = np.zeros([self.k, 1])
        self.mu = np.zeros([self.k, self.d])
        self.Sigma = np.zeros([self.k, self.d, self.d])

        # compute class prior probabilities, sample means, and sample covariances
        for i, y in enumerate(uniqueY):
            print('Training for class', y)
            # split the X into its classes
            Xi = X[Y == y]

            # compute the size of each class
            ni = Xi.shape[0]

            # compute the priors
            self.prior[i] = ni / n

            # compute the sample mean
            self.mu[i] = np.mean(Xi, axis = 0)

            # compute the centered data
            XiBar = Xi - self.mu[i]

            # compute the sample covariance
            self.Sigma[i] = (1/ni) * XiBar.T @ XiBar
```

```
def predict(self, X):
    n = X.shape[0]

    posteriorPre = np.zeros([n, self.k])

    # compute the pdf term of the posterior probabilities
    for i in range(n):
        for j in range(self.k):
            posteriorPre[i][j] = scipy.stats.multivariate_normal.pdf(X[i], self.mu[j], self.Sigma[j], allow_singular = True)

    # compute a vector proportional to the posterior probabilities
    posterior = posteriorPre * self.prior.T

    # find the label for each datapoint by choosing the most probable class
    predictions = np.argmax(posterior, axis = 1)

    return predictions
```

If there are many classes and/or the dimension is high, the Bayes classifier must estimate many parameters:

$\hat{\mu}_1, \ldots, \hat{\mu}_K$ } K d-dimensional class means

$\hat{\Sigma}_1, \ldots, \hat{\Sigma}_K$ } K $d \times d$ covariance matrices

For large enough datasets this may be okay to do with low error, but then it is still expensive to compute $O(d^2)$ covariances and $O(d)$ means.

The naive Bayes classifier makes the simplifying assumption that the features (data columns) are independent.

$$\hookrightarrow \mathbb{P}(x|C_i) = \mathbb{P}(x_1, \ldots, x_d | C_i) = \mathbb{P}(x_1|C_i) \cdots \mathbb{P}(x_d|C_i) = \prod_{j=1}^{d} \mathbb{P}(x_j|C_i)$$

$$Cov(X_i, X_j) = E[X_i X_j] - E[X_i]E[X_j] = 0$$

$$\Rightarrow \Sigma_i = \begin{pmatrix} \sigma_{i1}^2 & & 0 \\ & \sigma_{i2}^2 & \\ 0 & & \ddots \\ & & & \sigma_{id}^2 \end{pmatrix}$$

$$\Rightarrow \Sigma_i^{-1} = \begin{pmatrix} \frac{1}{\sigma_{i1}^2} & & 0 \\ & \frac{1}{\sigma_{i2}^2} & \\ 0 & & \ddots \\ & & & \frac{1}{\sigma_{id}^2} \end{pmatrix}$$

d $(x - \mu_i)^2$

$$\left(\quad \frac{1}{\sigma_{ij}^2} \right)$$

$$\Rightarrow (x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i) = \sum_{j=1}^{d} \frac{(x_j-\mu_{ij})^2}{\sigma_{ij}^2}$$

Then, $$f_i(x|\mu_i,\Sigma_i) = \frac{1}{(2\pi)^{d/2}\sqrt{\prod_{j=1}^{d}\sigma_{ij}^2}} \exp\left(-\sum_{j=1}^{d}\frac{(x_j-\mu_{ij})^2}{2\sigma_{ij}^2}\right)$$

$$= \prod_{j=1}^{d} \frac{1}{\sqrt{2\pi}\,\sigma_{ij}} \exp\left(-\frac{(x_j-\mu_{ij})^2}{2\sigma_{ij}^2}\right)$$

$$= \prod_{j=1}^{d} \mathbb{P}(x_j|c_i)$$

With naive Bayes, we only have to estimate μ_i's, σ_{ij}^2's rather than μ_i's and full covariance matrices Σ_i.

$\Rightarrow$ naive Bayes runs in $O(nd)$ time, super fast.

The risk is that the independence assumption may be false, causing errors.

Algorithm: naive Bayes' Classifier

train(D, Y):

for class $i = 1, \ldots, k$:

  $D_i \leftarrow \{x_j^T \mid y_j = c_i\}$ class-subsets

TU

$D_i \leftarrow \{x_j^T \mid y_j = c_i\}$ — class-subsets

$n_i \leftarrow |D_i|$ — class sizes

~~$\hat{P}(c_i) \leftarrow \frac{n_i}{n}$~~ — ~~prior probabilities~~ (don't really need it with assumption 1)

$\hat{\mu}_i \leftarrow \frac{1}{n_i} \sum_{D_i} x_j$ — class sample means

$\bar{D}_i \leftarrow D_i - \mathbf{1} \cdot \hat{\mu}_i^T$ — centered class subsets

~~$\hat{\Sigma}_i \leftarrow \frac{1}{n_i} \bar{D}_i^T \bar{D}_i$~~ $\hat{\sigma}_{ij}^2 = \frac{1}{n_i} (\bar{x}_j^i)^T \bar{x}_j^i$ — class sample ~~covariance matrix~~ variances

return ~~$\hat{P}(c_i)$~~ n_i, $\hat{\mu}_i$, ~~$\hat{\Sigma}_i$~~ $\hat{\sigma}_{ij}^2$ for $i = 1, \ldots, k$

predict(x):

Could use $\hat{P}(c_i)$

$\hat{y} \leftarrow \underset{c_i}{\operatorname{argmax}} \left\{ \prod_{j=1}^{d} f(x_j \mid \hat{\mu}_{ij}, \hat{\sigma}_{ij}^2) \cdot n_i \right\}$ — find class with max posterior probability

simpler with naive Bayes

return $\hat{y}$

Example: Irises Dataset

Iris Versicolor

Iris Setosa

Iris Virginica

	Sepal length X_1	Sepal width X_2	Petal length X_3	Petal width X_4	Class X_5
$\mathbf{x}_1$	5.9	3.0	4.2	1.5	Iris-versicolor
$\mathbf{x}_2$	6.9	3.1	4.9	1.5	Iris-versicolor
$\mathbf{x}_3$	6.6	2.9	4.6	1.3	Iris-versicolor
$\mathbf{x}_4$	4.6	3.2	1.4	0.2	Iris-setosa
$\mathbf{x}_5$	6.0	2.2	4.0	1.0	Iris-versicolor
$\mathbf{x}_6$	4.7	3.2	1.3	0.2	Iris-setosa
$\mathbf{x}_7$	6.5	3.0	5.8	2.2	Iris-virginica
$\mathbf{x}_8$	5.8	2.7	5.1	1.9	Iris-virginica
⋮	⋮	⋮	⋮	⋮	⋮
$\mathbf{x}_{149}$	7.7	3.8	6.7	2.2	Iris-virginica
$\mathbf{x}_{150}$	5.1	3.4	1.5	0.2	Iris-setosa

z

Projecting the data onto two dimensions with PCA…

Example 22.1. Figure 22.1 shows the 2-dimensional Iris dataset, with the two attributes being `sepal length` and `sepal width`. It has 150 points, and has three equal-sized classes: `Iris-setosa` (c_1; circles), `Iris-versicolor` (c_2; squares) and

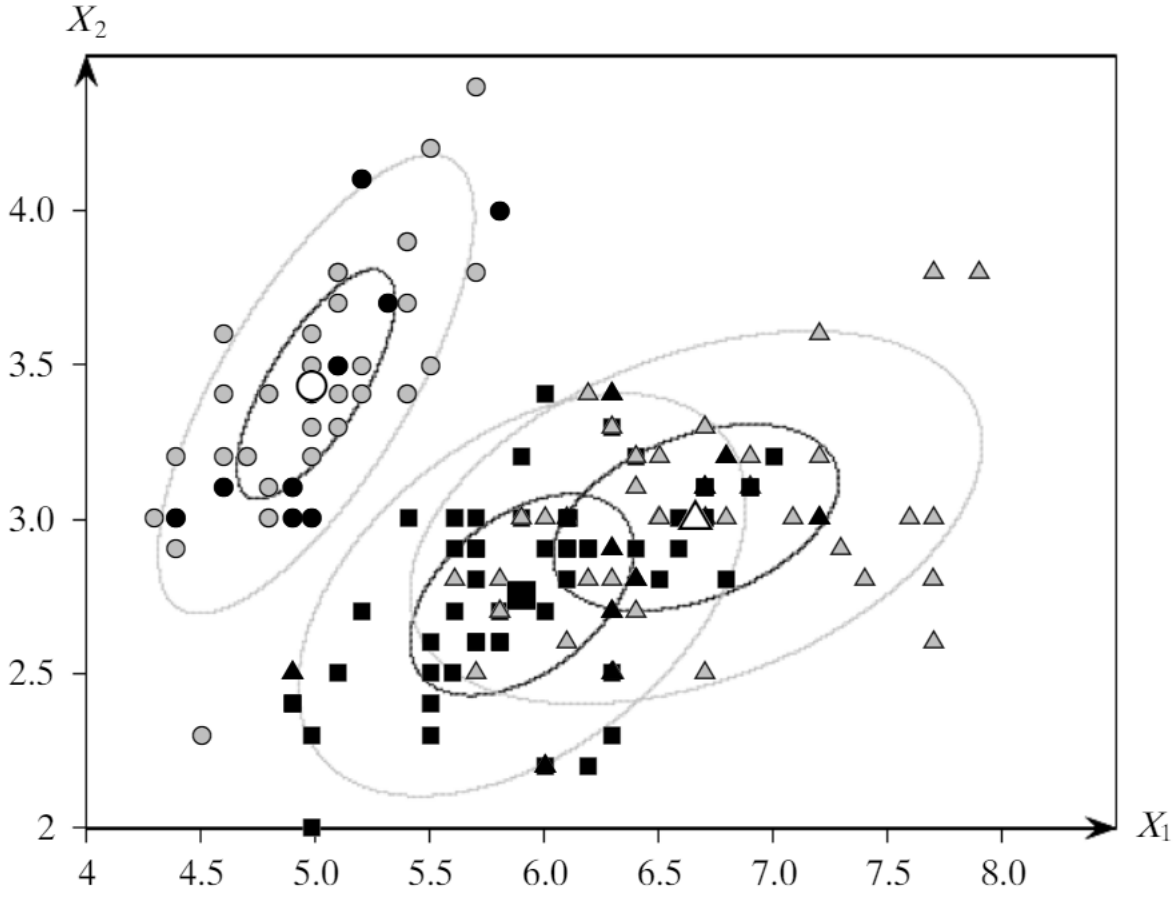


Figure 22.1. Iris dataset: three classes.

`Iris-virginica` (c_3; triangles). The dataset is partitioned into training and testing sets, in the ratio 80:20. Thus, the training set has 120 points (shown in light gray), and the testing set $\mathbf{D}$ has $n = 30$ points (shown in black). One can see that whereas c_1 is well separated from the other classes, c_2 and c_3 are not easy to separate. In fact, some points are labeled as both c_2 and c_3 (e.g., the point $(6, 2.2)^T$ appears twice, labeled as c_2 and c_3).

We classify the test points using the full Bayes classifier (see Chapter 18). Each class is modeled using a single normal distribution, whose mean (in white) and density contours (corresponding to one and two standard deviations) are also plotted in Figure 22.1. The classifier misclassifies 8 out of the 30 test cases. Thus, we have

$$Error\ Rate = 8/30 = 0.267$$

$$Accuracy = 22/30 = 0.733$$

Example 22.2. Consider the 2-dimensional Iris dataset shown in Figure 22.1. In Example 22.1 we saw that the error rate was 26.7%. However, the error rate measure does not give much information about the classes or instances that are more difficult to classify. From the class-specific normal distribution in the figure, it is clear that the Bayes classifier should perform well for c_1, but it is likely to have problems discriminating some test cases that lie close to the decision boundary between c_2 and c_3. This information is better captured by the confusion matrix obtained on the testing set, as shown in Table 22.1. We can observe that all 10 points in c_1 are classified correctly. However, only 7 out of the 10 for c_2 and 5 out of the 10 for c_3 are classified correctly.

From the confusion matrix we can compute the class-specific precision (or accuracy) values:

$$prec_1 = \frac{n_{11}}{m_1} = 10/10 = 1.0$$

$$prec_2 = \frac{n_{22}}{m_2} = 7/12 = 0.583$$

$$prec_3 = \frac{n_{33}}{m_3} = 5/8 = 0.625$$

The overall accuracy tallies with that reported in Example 22.1:

$$Accuracy = \frac{(n_{11} + n_{22} + n_{33})}{n} = \frac{(10 + 7 + 5)}{30} = 22/30 = 0.733$$

The overall accuracy tallies with that reported in Example 22.1:

$$Accuracy = \frac{(n_{11} + n_{22} + n_{33})}{n} = \frac{(10 + 7 + 5)}{30} = 22/30 = 0.733$$

The class-specific recall (or coverage) values are given as

$$recall_1 = \frac{n_{11}}{n_1} = 10/10 = 1.0$$

$$recall_2 = \frac{n_{22}}{n_2} = 7/10 = 0.7$$

$$recall_3 = \frac{n_{33}}{n_3} = 5/10 = 0.5$$

From these we can compute the class-specific F-measure values:

$$F_1 = \frac{2 \cdot n_{11}}{(n_1 + m_1)} = 20/20 = 1.0$$

$$F_2 = \frac{2 \cdot n_{22}}{(n_2 + m_2)} = 14/22 = 0.636$$

$$F_3 = \frac{2 \cdot n_{33}}{(n_3 + m_3)} = 10/18 = 0.556$$

Thus, the overall F-measure for the classifier is

$$F = \frac{1}{3}(1.0 + 0.636 + 0.556) = \frac{2.192}{3} = 0.731$$

Table 22.1. Contingency table for Iris dataset: testing set

	True			
Predicted	Iris-setosa (c_1)	Iris-versicolor (c_2)	Iris-virginica(c_3)	
Iris-setosa (c_1)	10	0	0	$m_1 = 10$
Iris-versicolor (c_2)	0	7	5	$m_2 = 12$
Iris-virginica (c_3)	0	3	5	$m_3 = 8$
	$n_1 = 10$	$n_2 = 10$	$n_3 = 10$	$n = 30$

Example 22.3. Consider the Iris dataset projected onto its first two principal components, as shown in Figure 22.2. The task is to separate `Iris-versicolor` (class c_1; in circles) from the other two Irises (class c_2; in triangles). The points from class c_1 lie in-between the points from class c_2, making this is a hard problem for (linear) classification. The dataset has been randomly split into 80% training (in gray) and 20% testing points (in black). Thus, the training set has 120 points and the testing set has $n = 30$ points.

Applying the naive Bayes classifier (with one normal per class) on the training set yields the following estimates for the mean, covariance matrix and prior probability for each class:

$$\hat{P}(c_1) = 40/120 = 0.33 \qquad \hat{P}(c_2) = 80/120 = 0.67$$

$$\hat{\mu}_1 = \begin{pmatrix} -0.641 & -0.204 \end{pmatrix}^T \qquad \hat{\mu}_2 = \begin{pmatrix} 0.27 & 0.14 \end{pmatrix}^T$$

$$\widehat{\boldsymbol{\Sigma}}_1 = \begin{pmatrix} 0.29 & 0 \\ 0 & 0.18 \end{pmatrix} \qquad \widehat{\boldsymbol{\Sigma}}_2 = \begin{pmatrix} 6.14 & 0 \\ 0 & 0.206 \end{pmatrix}$$

The mean (in white) and the contour plot of the normal distribution for each class are also shown in the figure; the contours are shown for one and two standard deviations along each axis.

For each of the 30 testing points, we classify them using the above parameter estimates (see Chapter 18). The naive Bayes classifier misclassified 10 out of the 30 test instances, resulting in an error rate and accuracy of

$$Error\ Rate = 10/30 = 0.33$$

$$Accuracy = 20/30 = 0.67$$

Table 22.3. Iris PC dataset: contingency table for binary classification

	True		
Predicted	Positive (c_1)	Negative (c_2)	
Positive (c_1)	$TP = 7$	$FP = 7$	$m_1 = 14$
Negative (c_2)	$FN = 3$	$TN = 13$	$m_2 = 16$
	$n_1 = 10$	$n_2 = 20$	$n = 30$

The confusion matrix for this binary classification problem is shown in Table 22.3. From this table, we can compute the various performance measures:

$$prec_P = \frac{TP}{TP + FP} = \frac{7}{14} = 0.5$$

$$prec_N = \frac{TN}{TN + FN} = \frac{13}{16} = 0.8125$$

$$recall_P = sensitivity = TPR = \frac{TP}{TP + FN} = \frac{7}{10} = 0.7$$

$$recall_N = specificity = TNR = \frac{TN}{TN + FP} = \frac{13}{20} = 0.65$$

$$FNR = 1 - sensitivity = 1 - 0.7 = 0.3$$

$$FPR = 1 - specificity = 1 - 0.65 = 0.35$$

We can observe that the precision for the positive class is rather low. The true positive rate is also low, and the false positive rate is relatively high. Thus, the naive Bayes classifier is not particularly effective on this testing dataset.

Example 22.4. Consider the binary classification problem from Example 22.3 for the Iris principal components dataset. The test dataset $\mathbf{D}$ has $n = 30$ points, with $n_1 = 10$ points in the positive class and $n_2 = 20$ points in the negative class.

We use the naive Bayes classifier to compute the probability that each test point belongs to the positive class (c_1; `iris-versicolor`). The score of the classifier for test point $\mathbf{x}_i$ is therefore $S(\mathbf{x}_i) = P(c_1|\mathbf{x}_i)$. The sorted scores (in decreasing order) along with the true class labels are shown in Table 22.5.

The ROC curve for the test dataset is shown in Figure 22.3. Consider the positive score threshold $\rho = 0.71$. If we classify all points with a score above this value as positive, then we have the following counts for the true and false positives:

$$TP = 3 \qquad\qquad FP = 2$$

The false positive rate is therefore $\frac{FP}{n_2} = 2/20 = 0.1$, and the true positive rate is $\frac{TP}{n_1} = 3/10 = 0.3$. This corresponds to the point $(0.1, 0.3)$ in the ROC curve. Other points on the ROC curve are obtained in a similar manner as shown in Figure 22.3. The total area under the curve is 0.775.

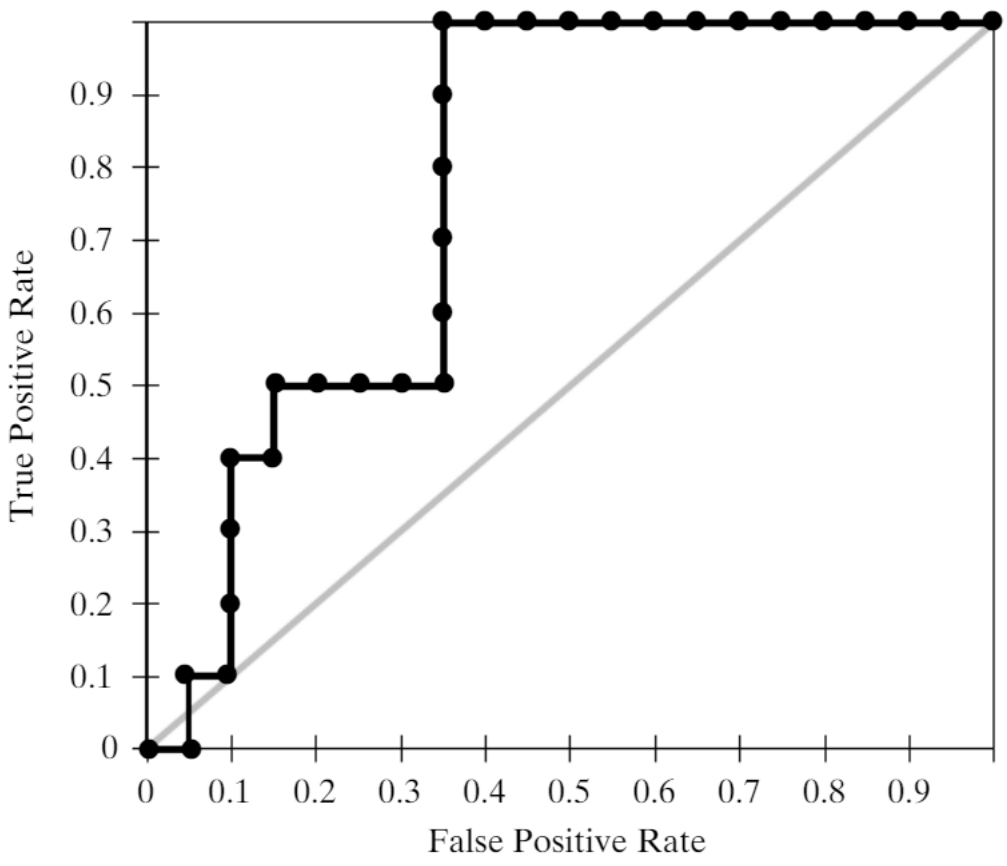


Figure 22.3. ROC plot for Iris principal components dataset. The ROC curves for the naive Bayes (black) and random (gray) classifiers are shown.

ρ	FP	TP	(FPR, TPR)	AUC
∞	0	0	$(0, 0)$	0
0.9	0	1	$(0, 0.333)$	0
0.8	1	3	$(0.5, 1)$	0.333
0.1	2	3	$(1, 1)$	0.833

Figure 22.4 shows the ROC plot, with the shaded region representing the AUC. We can observe that a trapezoid is obtained whenever there is at least one positive and one negative point with the same score. The total AUC is 0.833, obtained as the sum of the trapezoidal region on the left (0.333) and the rectangular region on the right (0.5).

Example 22.6. In addition to the ROC curve for the naive Bayes classifier, Figure 22.3 also shows the ROC plot for the random classifier (the diagonal line in gray). We can see that the ROC curve for the naive Bayes classifier is much better than random. Its AUC value is 0.775, which is much better than the 0.5 AUC for a random classifier. However, at the very beginning, naive Bayes performs worse than the random classifier because the highest scored point is from the negative class. As such, the ROC curve should be considered as a discrete approximation of a smooth curve that would be obtained for a very large (infinite) testing dataset.

MNIST Dataset

The Modified National Institute of Standards and Technology (MNIST) database contains 70,000 grayscale labeled images (each 28-by-28 pixels) of handwritten digits from US Census Bureau employees and high school students. Classifying the MNIST database is a common benchmark for machine learning techniques.