

Lab1

MAP and Gaussian Datasets



Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Discriminants for Gaussian classes

- MAP decision rule $\omega_{MAP} = \arg \max_{\omega_i} \Pr(\omega_i | \mathbf{x})$
- Density function for class i : $f_{\mathbf{x}}(\mathbf{x} | \omega_i) \sim N(\boldsymbol{\mu}_i, \mathbf{C}_i)$
- A priori probability: $\Pr(\omega_i)$
- Discriminant function for MAP
$$h_i(\mathbf{x}) = \ln f_{\mathbf{x}}(\mathbf{x} | \omega_i) + \ln \Pr(\omega_i) =$$
$$= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{C}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{d}{2} \ln(2\pi) - \frac{1}{2} \ln |\mathbf{C}_i| + \ln \Pr(\omega_i)$$
- Three cases for the covariance matrix:
$$C_i = \sigma^2 I$$
$$C_i = C$$
$$C_i = \text{arbitrary}$$

Synthetic Gaussian datasets

- **Case 1:** $f(x | w_c) = N(\mu_c, \sigma^2 I)$

example with c=2 classes, d=3 features (Mlearn_lab1_1.ipynb)

- **Case 2:** $f(x | w_c) = N(\mu_c, C)$

example with c=4 classes, d=2 features (Mlearn_lab1_2.ipynb)

in Lab1 code: Variables
M_means and M_covar


Apply MAP for cases 1 or 2:

- discriminant is a linear function
- decision boundaries are hyperplanes (lines for d=2, planes for d=3)
- decision boundary:

$$w^t x + w_0 = 0$$

$$w_1 x_1 + w_2 x_2 + \dots + w_d x_d + w_0 = 0$$

Synthetic Gaussian datasets

- **Case 3:** $f(x | w_c) = N(\mu_c, C_c)$  in Lab1 code: Variables M_means and M_covar

example with c=4 classes, d=2 features (Mlearn_lab1_2.ipynb)

Apply MAP for case 3:

- discriminant is a quadratic function
- decision boundaries are hyper-quadratic (lines, spheres, ellipsoids, paraboloids, hyperboloids...)
- decision boundary: $w^t Ax + w^t x + w_0 = 0$

example d=2 $x_1^2 A_{11} + x_2^2 A_{22} + x_1 A_{12} x_2 + x_2 A_{21} x_1 + w_1 x_1 + w_2 x_2 + w_0 = 0$

Synthetic Gaussian datasets

In the Notebooks:

- **Case1:** $c=2$ classes, $d=3$ features (Mlearn_lab1_1.ipynb)

$$f(\mathbf{x}|\omega_c) = N(\mathbf{m}_c, \mathbf{C}); \quad c = 1, 2$$
$$\mathbf{C} = \frac{1}{d} \sigma^2 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}; \quad d = 3$$
$$SNR = 10 \log_{10} \left(\frac{\text{average energy}}{\sigma^2} \right) = 10 \log_{10} \left(\frac{\mathbf{m}_c^T \mathbf{m}_c}{\sigma^2} \right)$$

Run the code several times, using different SNR values (SNR = 3, 0, -3, -10)

- **Case2:** $c=4$ classes, $d=2$ features (Mlearn_lab1_2.ipynb)

$$f(\mathbf{x}|\omega_c) = N(\mathbf{m}_c, \mathbf{C}); \quad c = 1, \dots, 4$$
$$\mathbf{C} = \frac{1}{d} \sigma^2 \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}; \quad d = 2$$

- **Case2:** $c=4$ classes, $d=2$ features (Mlearn_lab1_2.ipynb)

$$f(x | w_c) = N(\mu_c, C_c)$$

C_c Each class c a different cov matrix C_c (different ρ)

Scikit-learn

- <https://scikit-learn.org/stable/> open source ML library in Python
- Data representation:
 - feature matrix [n_samples, n_features] (NumPy array o Pandas DataFrame)
 - target array [n_samples] (NumPy array o Pandas Series)
- Scikit-learn Estimator API provides a consistent interface for a wide range of ML applications
- Typically, the steps for using the Scikit-Learn estimator API are the following:
 1. Choose a class of model by importing the appropriate estimator class from Scikit-Learn
 2. Choose model hyperparameters by instantiating this class with desired values.
 3. Arrange data into a features matrix and target vector.
 4. Fit the model to the data by calling the `fit()` method of the model instance
 5. Apply the Model to new data:
 - For supervised learning, often we predict labels for unknown data using the `predict()` method.
 - For unsupervised learning, we often transform or infer properties of the data using the `transform()` or `predict()` method.

Linear and quadratic discriminant analysis in scikit-learn

Obtaining linear and quadratic classifiers using scikit-learn:

- Classifier design using the training dataset: features in matrix **X_train**, labels in vector **y_train**

```
# Linear Discriminant Analysis
lda = LinearDiscriminantAnalysis(solver="svd", store_covariance=True)
ldamodel = lda.fit(X_train, y_train)
```

```
# Quadratic Discriminant Analysis
qda = QuadraticDiscriminantAnalysis(store_covariance=True)
qdamodel = qda.fit(X_train, y_train)
```

- Predicted labels for data in matrix X_test, labels in y_test

```
y_tpred_lda = ldamodel.predict(X_test)
y_tpred_qda = qdamodel.predict(X_test)
```

Metrics

- **Confusion matrix:** provides a by-class comparison between the results of the predicted and the actual classes.
 - Correct classifications are in the diagonal

		Predicted class		
		class1	class2	class3
Actual class	class1	x(1,1)	x(1,2)	x(1,3)
	class2	x(2,1)	x(2,2)	x(2,3)
	class3	x(3,1)	x(3,2)	x(3,3)

		Predicted class		
		class1	class2	class3
Actual class	class1	5	3	0
	class2	2	3	1
	class3	0	2	11

- **Accuracy:** measures the proportion of correct classifications (no distinction between classes)

$$A = \frac{\sum_{i=1}^n x(i, i)}{\sum_{i=1}^n \sum_{j=1}^n x(i, j)}$$

- **Error rate:** 1 - accuracy

$$E = 1 - A$$

Metrics for binary problems

- **Binary classification problem:** 2 classes. The class of interest is commonly called the **positive class**, and the other **negative class**.

		Predicted class	
		positive	negative
Actual class	positive	True positive TP	False negative FN
	negative	False positive FP	True negative TN

- **Accuracy:** $A = \frac{TP + TN}{all} = \frac{TP + TN}{TP + FP + TN + FN}$

Error rate: $E = 1 - A$

- **Recall (sensitivity, true positive rate):**

$$R = TPR = \frac{TP}{actual\ positives} = \frac{TP}{TP + FN}$$

- **Precision**

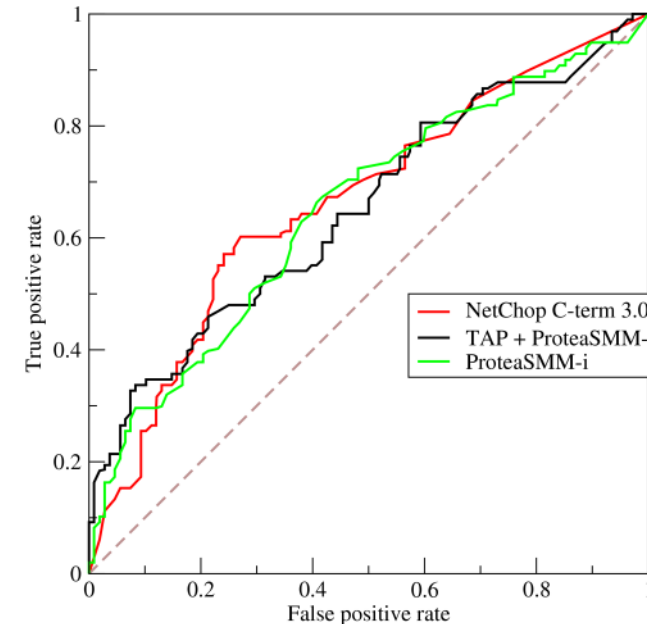
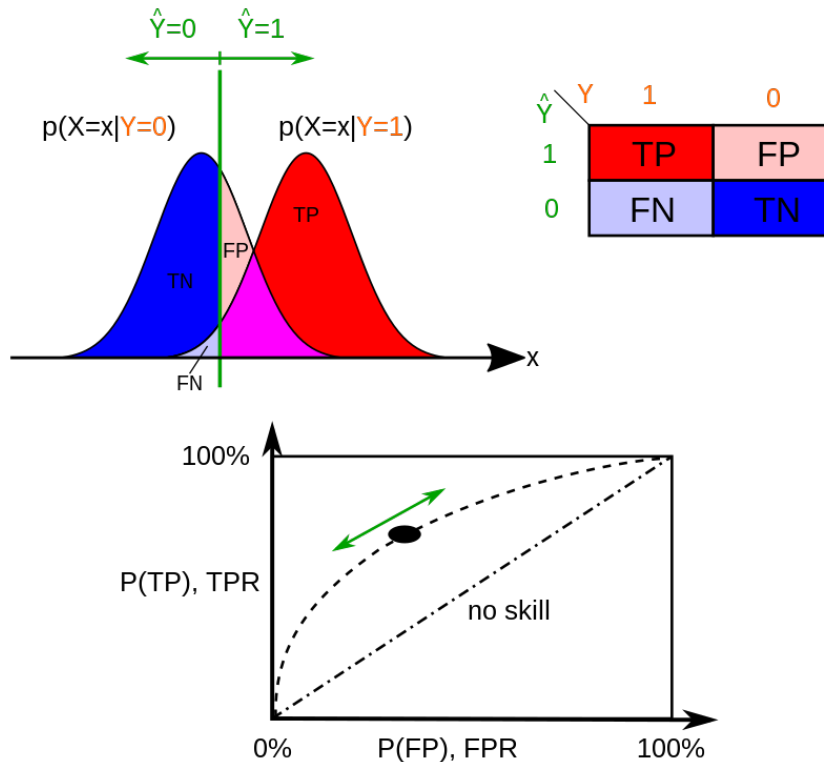
$$P = \frac{TP}{predicted\ positive} = \frac{TP}{TP + FP}$$

- **False positive rate:**

$$FPR = \frac{FP}{actual\ negative} = \frac{FP}{TN + FP}$$

Metrics for binary problems

- Receiver Operating Characteristics or ROC curve is a plot of the **true positive rate** (TPR) against the **false positive rate** (FPR), obtained varying the classifier threshold.
- One classifier produces a single point in ROC space; plotting the point for each possible threshold results in a curve



Figures: Wikipedia ROC

Metrics

- **Mahalanobis distance:** squared Mahalanobis distance between an observation x and a sample distribution with mean μ_2 and covariance matrix C_2 :

$$D_M(x, \omega_2) = (x - \mu_2)^T C_2^{-1} (x - \mu_2)$$

squared Mahalanobis distance between two classes:

$$D_M(\omega_1, \omega_2) = \sum_{x \in \omega_1} (x - \mu_2)^T C_2^{-1} (x - \mu_2)$$

where the mean and covariance matrix are estimated using the training dataset. Note that this is not a symmetric measure

Metrics in scikit-learn

Metrics module: `from sklearn.metrics import`

- **Error and confusion matrix:**

```
linear_error = 1. - accuracy_score(y_test, y_test_lda)
confusion_matrix(y_test, y_test_lda)
```

Note: `ConfusionMatrixDisplay` can be used to visually represent a confusion matrix

- **Receiver Operating Characteristic (ROC)**

```
ldaroc = RocCurveDisplay.from_estimator(ldamodel, X_test, y_test, ax=ax[0])
```

- **Mahalanobis distance:**

- See `def mahalanobis(x=None, data=None, cov=None)`