Classification

SLM003 06/08/2018

References:

`ISL04` James G., Witten D., Hastie T., Tibshirani R. (2013) **Classification**. In: *An Introduction to Statistical Learning*. Springer Texts in Statistics, vol 103. Springer, New York, NY. doi: https://doi.org/10.1007/978-1-4614-7138-7_4 (https://doi.org/10.1007/978-1-4614-7138-7_4)

`ESL04` Hastie T., Tibshirani R., Friedman J. (2009) **Linear Methods for Classification**. In: *The Elements of Statistical Learning* (2nd ed.). Springer Series in Statistics. Springer, New York, NY. doi: https://doi.org/10.1007/978-0-387-84858-7_4 (https://doi.org/10.1007/978-0-387-84858-7_4)

Outline

1. Logistic regression
2. Discriminant analysis
    A. Linear discriminant analysis (LDA)
    B. Quadratic discriminant analysis (QDA)

Objectives

- Understand the principles behind the methods
- Develop intuition of the mathematical formulation

What is ”classification”?

- **Supervised learning**: use inputs to predict output
- Classification predicts ***qualitative*** (a.k.a. *categorical*, *discrete*) outputs
- Input: ***predictors*** (a.k.a. *features*, *independent variables*, X) -- quantitative and/or qualitative
- Output: ***response*** (a.k.a. *target*, *dependent variable*, y)
  - which may be refered to as different *response levels*, *targets*, ***classes***, *categories*

Logistic regression

Goal: Describe predictor-response relationship in the *training data* using the **logistic model**. Make prediction using this model.

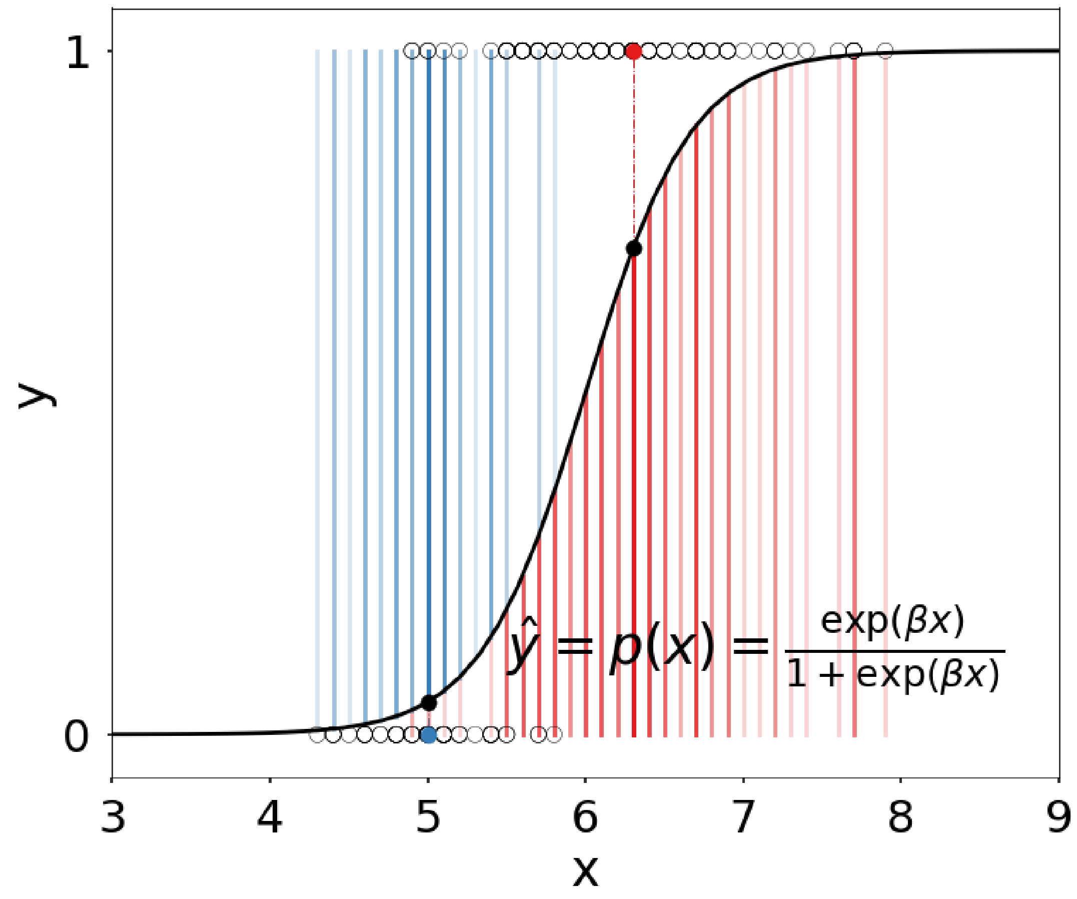
The *logistic function* (a.k.a. *sigmoid curve*) is defined as:

$$p(X) = \frac{\exp(\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p)}{1 + \exp(\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p)} \tag{4.6}$$

$p(X)$: predicted response | X_i: predictors | β_i: parameters of the model

How to fit the model, i.e. how to determine the appropriate β_i?

Fitting a logistic model using "maximum likelihood" ($k = 2$)



For a single training data point:

if $y_i = 1$, model plausibility at $x_i = p(x_i)$

if $y_i = 0$, model plausibility at $x_i = 1 - p(x_i)$

n.b. the closer $p(x_i)$ is to 0, i.e. y_i, the better

For all training data,
Model plausibility, i.e. "Likelikhood":

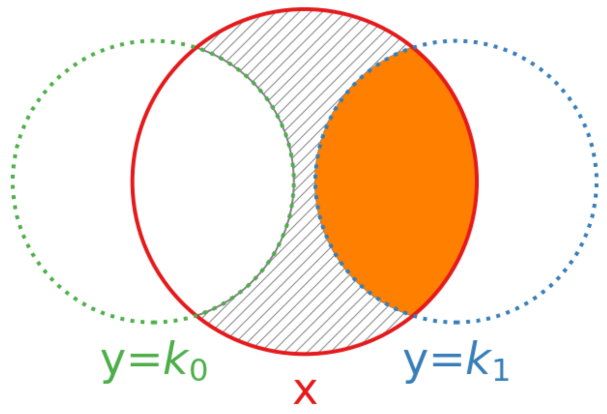
$$\mathrm{l} = \prod_{i:y_i=1} p(x_i) \prod_{i\prime:y_{i\prime}=0} (1 - p(x_{i\prime}))$$

Model is best-fitted when β maximises l.

Discriminant analysis

Goal: Assign data to the most probable class based on **distribution statistics** derived from *training data* and/or prior knowledge

Bayes' Theorem



$$P(x)\cdot P(y=k_1|x) = P(y=k_1)\cdot P(x|y=k_1)$$

$$P(y=k_1|x) = \frac{P(y=k_1)\cdot P(x|y=k_1)}{P(y=k_0)\cdot P(x|y=k_0) + P(y=k_1)\cdot P(x|y=k_1)}$$

$$\underbrace{P(y=k_i|x)}_{\text{posterior probability}} = \frac{\overbrace{P(y=k_i)}\cdot\overbrace{P(x|y=k_i)}}{\sum_{l=0}^{K}\underbrace{P(y=l)}_{\text{class prior}}\cdot\underbrace{P(x|y=k_l)}_{\text{class density } f(x)}}$$

Classification using discriminant analysis

For classification, we do not need to know the posterior $P(y=k_i|x)$, we need only to know which class k_i has the highest posterior, i.e. $\underset{k_i}{\operatorname{argmax}} P(y=k_i|x)$

Based on assumptions about the density function $P(x|y=k_l)$, we can define a *discriminant function* $\delta(x)$, such that:

$$\underset{k}{\operatorname{argmax}}\, \delta_k(x) = \underset{k}{\operatorname{argmax}}\, P(y=k|x)$$

Assume *Gaussian* (a.k.a. *normal*) density function,

- With **common** *predictor covariance* Σ shared by all classes, we can derive a *linear* discriminant (LDA):
$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} {\mu_k}^T \Sigma^{-1} \mu_k + \log \pi_k \tag{4.19}$$
- With **class-specific** *predictor covariance* Σ_k, we get a *quadratic* discriminant (QDA):
$$\delta_k(x) = -\frac{1}{2} x^T \Sigma_k^{-1} x + x^T \Sigma_k^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k - \frac{1}{2} \log |\Sigma_k| \tag{4.23}$$

Errors are not born equal: Thresholding binary classification

As discussed so far, we threshold the predicted probability (both for logistic regression and for discriminant analysis) at 0.5, without distinguishing between different types of errors.

	Pos	Neg
Predict pos	True pos	False pos (Type I error)
Predict neg	False neg (Type II error)	True neg

$$\text{Sensitivity, Recall} = \frac{\text{True pos}}{\text{Pos}}$$

$$\text{Specificity} = 1 - \frac{\text{False pos}}{\text{Neg}} = \frac{\text{True neg}}{\text{Neg}}$$

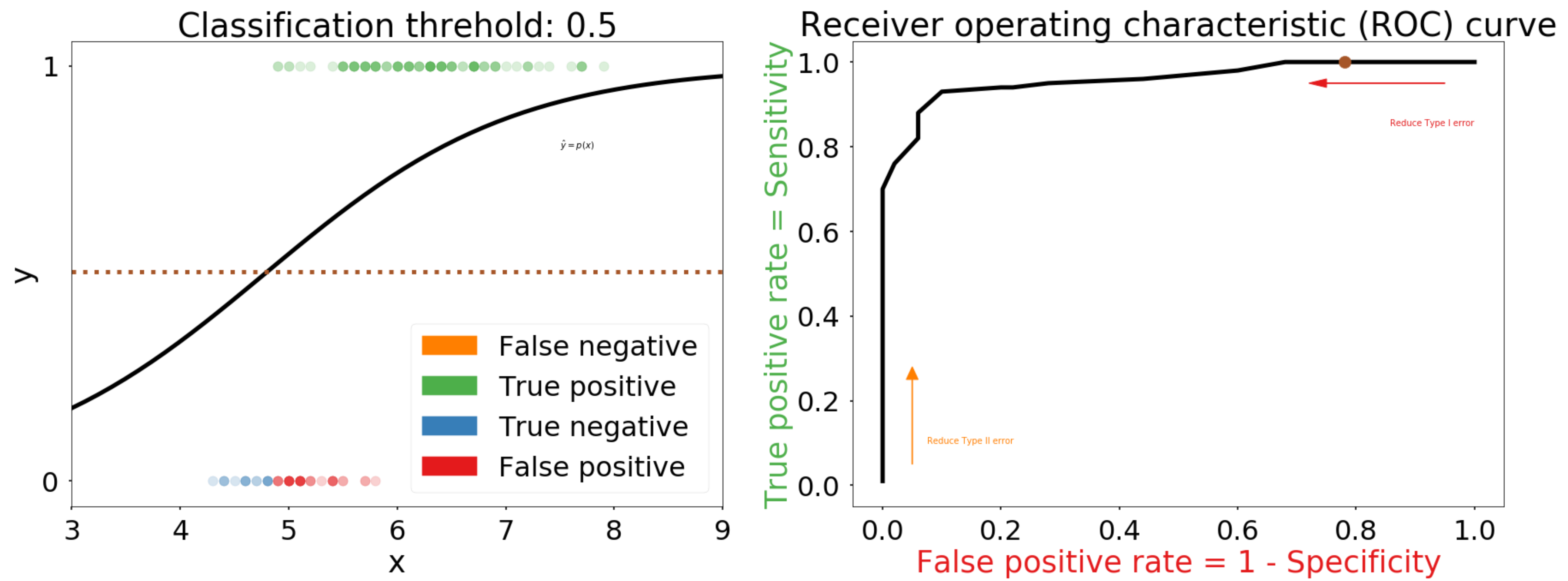
$$\text{Precision} = \frac{\text{True pos}}{\text{Predict pos}}$$

$$\text{Accuracy} = \frac{\text{True}}{\text{Total}}$$

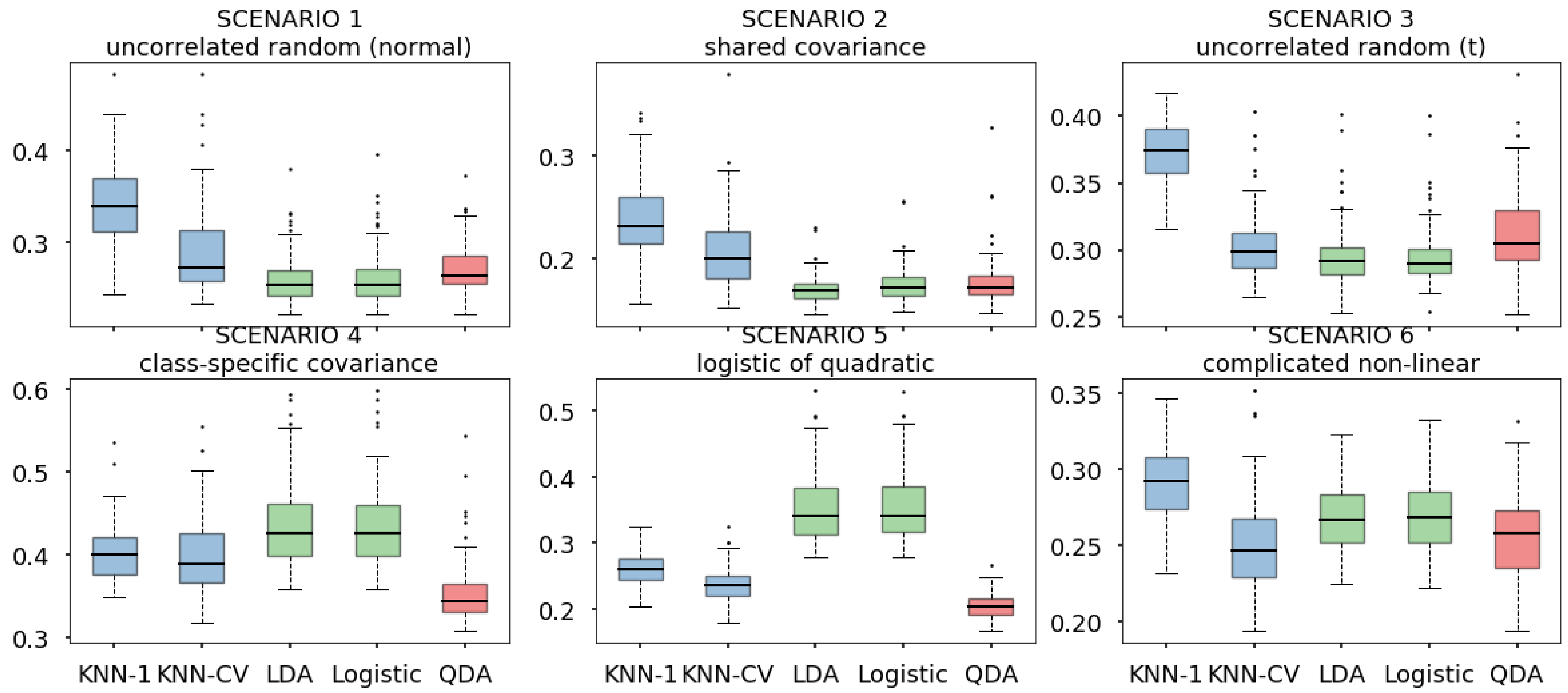
Further reading: Wikipedia:Confusion_matrix (https://en.wikipedia.org/wiki/Confusion_matrix)

Choosing classification threshold based on the ROC curve

```
In [5]: interactive(fig_threshold, page=(0,len(thresh_list)-1))
```



Comparison between different classifiers (reproducing Fig 4.10, 4.11)



Summary

- Linear decision boundary:
  - Logistic Regression: model each binary decision using a logistic form of *linear* regression, predict the binary response probability
  - LDA: model predictor distributions of each class as *Gaussian*, then based on Bayes' Theorem, compare to see which response is more probable
    - More stable than logistic regression when classes are well-separated
    - If Guassian assumption is valid, more stable than logistic regression when sample size is small
- Non-linear decision boundary:
  - QDA: same as LDA, but allow each class to have *different predictor covariances*
    - More suitable for fitting non-linear decision boundary, but risk overfitting if true boundary is linear

Scikit-Learn basics

Load example dataset

In [11]:
```
from sklearn import datasets

X, y = datasets.load_iris(return_X_y=True)

X.shape, y.shape
```

Out[11]:
```
((150, 4), (150,))
```

sklearn estimator classes

- Regressor: `neighbors.KNeighborsRegressor`, `linear_model.LinearRegression`
- Classifier: `neighbors.KNeighborsClassifier`, `linear_model.LogisticRegression`
- Clusterer: `cluster.KMeans`, `cluster.AgglomerativeClustering`
- Transformer: `decomposition.PCA`, `manifold.TSNE`
- etc.

General usage

In [12]:
```
# 0. Import estimator constructor class
from sklearn.neighbors import KNeighborsClassifier

# 1. Initialise an estimator instance
estimator = KNeighborsClassifier()

# 3. Initalise an estimator instance with specific parameters
# Note that estimator instance has been overwritten
estimator = KNeighborsClassifier(n_neighbors=2)  # Jupyter: Shift + Tab inside brackets for docstrings; supports tab-c
ompletion

# 4. Modify parameters of existing estimator instance
estimator.set_params(n_neighbors=4)

# 5. Jupyter: tab-completion is your friend
# estimator.

# 2. Fitting (Jupyter: toggle comment with Ctrl + /)
estimator.fit(X, y)  # Also display estimator parmeters by default
```

Out[12]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='uniform')
```

Classifier-specific usage

In [13]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

In [14]:
```
from sklearn.model_selection import train_test_split

# 1. Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=30)

# 2. Fit with training data
logreg.fit(X_train, y_train)

# 3. Classify test data
logreg.predict(X_test)

# 4. Predict probability p_k(x=x_i), useful for thresholding/weighting
logreg.predict_proba(X_test).shape
np.argmax(logreg.predict_proba(X_test), axis=1) == logreg.predict(X_test)
np.argmax(logreg.predict_proba(X_test)*[1,2,1], axis=1) == logreg.predict(X_test)

# 5. Test accuracy
logreg.predict(X_test) == y_test
logreg.score(X_test, y_test)
```

Out[14]:
```
0.9666666666666667
```

Classifier attributes (model paramenters)

In [15]:
```
# Store attributes are indicated by a trailing underscore
print(logreg.classes_, '\n\n', logreg.coef_, '\n\n', logreg.intercept_)

# Attributes are updated upon re-training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=30)
logreg.fit(X_train, y_train)
print('\n\n', logreg.coef_, '\n\n', logreg.intercept_)
```

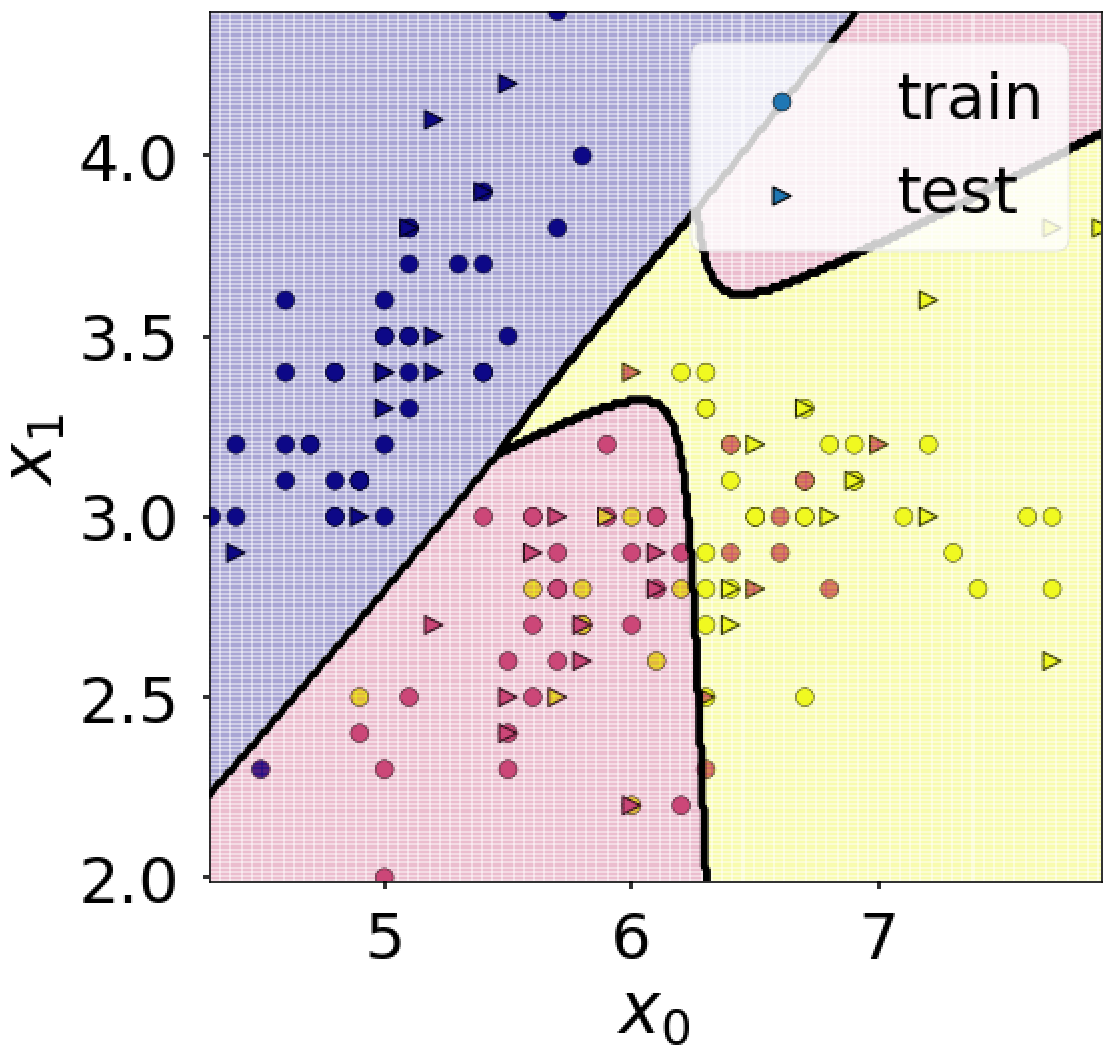
```
[0 1 2]

 [[ 0.42246987  1.35051861 -2.14480559 -0.95452713]
 [ 0.33231761 -1.37640461  0.43295424 -0.99460386]
 [-1.56735495 -1.41692793  2.2974056   2.26647004]]

 [ 0.25969649  0.85941538 -1.01267347]


 [[ 0.39709677  1.3914835  -2.17758643 -0.97797652]
 [ 0.58191457 -1.76581288  0.5512131  -1.36892025]
 [-1.52492997 -1.52941477  2.30289617  2.30936333]]

 [ 0.24666901  0.80446332 -1.114254  ]
```

Plotting decision boundary of a trained classifier

In [16]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

qda = QuadraticDiscriminantAnalysis(store_covariance=True)
X_train, X_test, y_train, y_test = train_test_split(X[:,:2], y, test_size=0.25)  # Limit to p=2
qda.fit(X_train, y_train)
```

Out[16]:
```
QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
                              store_covariance=True, store_covariances=None, tol=0.0001)
```

In [17]:
```
%%capture stdout
import matplotlib.pyplot as plt
from util import make_mesh
fig, ax = plt.subplots(figsize=(8,8))

# 1. Plot train and test data with different markers
ax.scatter(X_train[:,0], X_train[:,1], c=y_train, \
           s=80, cmap='plasma', edgecolors='k', label='train')
ax.scatter(X_test[:,0], X_test[:,1], c=y_test, \
           s=80, cmap='plasma', edgecolors='k', marker='>', \
           lw=1, label='test')
ax.legend()
ax.set_xlabel('$x_0$')
ax.set_ylabel('$x_1$')

# 2. Make a grid of points
x_mesh, (X0, X1) = make_mesh(X, step_size=0.01)
print(x_mesh.shape, x_mesh[0, 0, :])

# 3. Evaluate response class at each grid point
grid_eval = qda.predict(x_mesh.reshape(-1,2)).reshape(\
                    x_mesh.shape[:2])
ax.pcolormesh(X0, X1, grid_eval, cmap='plasma', alpha=0.1)
ax.set_xlim(X0.min(), X0.max())
ax.set_ylim(X1.min(), X1.max())

# 4. Plot decision boundary
ax.contour(X0, X1, grid_eval, vmin=0, vmax=0.5, cmap="Greys")

plt.show()
```


train
test
4.0
3.5
3.0
2.5
2.0
5
6
7
x_0
x_1