Classification

Elsa Ferreira Gomes

2024/2025

Classification

- A **Linear Regression** model assumes that Y is a **quantitative** variable

- In **Classification** Y is a **qualitative** (or categorical) variable

Example of Classifiers

- Logistic Regression
- Discriminant Analysis

Classification

- **Qualitative variables** take values in an unordered set C, such as: *eye color* ∈ {brown, blue, green}

- Given a feature vector X and a **qualitative** response Y taking values in the set C, the classification task is:

  - build a function $C(X)$ that takes as input the feature vector X and predicts its value for Y.
- Often we are more interested in estimating the probabilities that X belongs to each category in C.

  - Example: it is more valuable to have an estimate of the probability that an insurance claim is fraudulent, than a classification fraudulent or not.

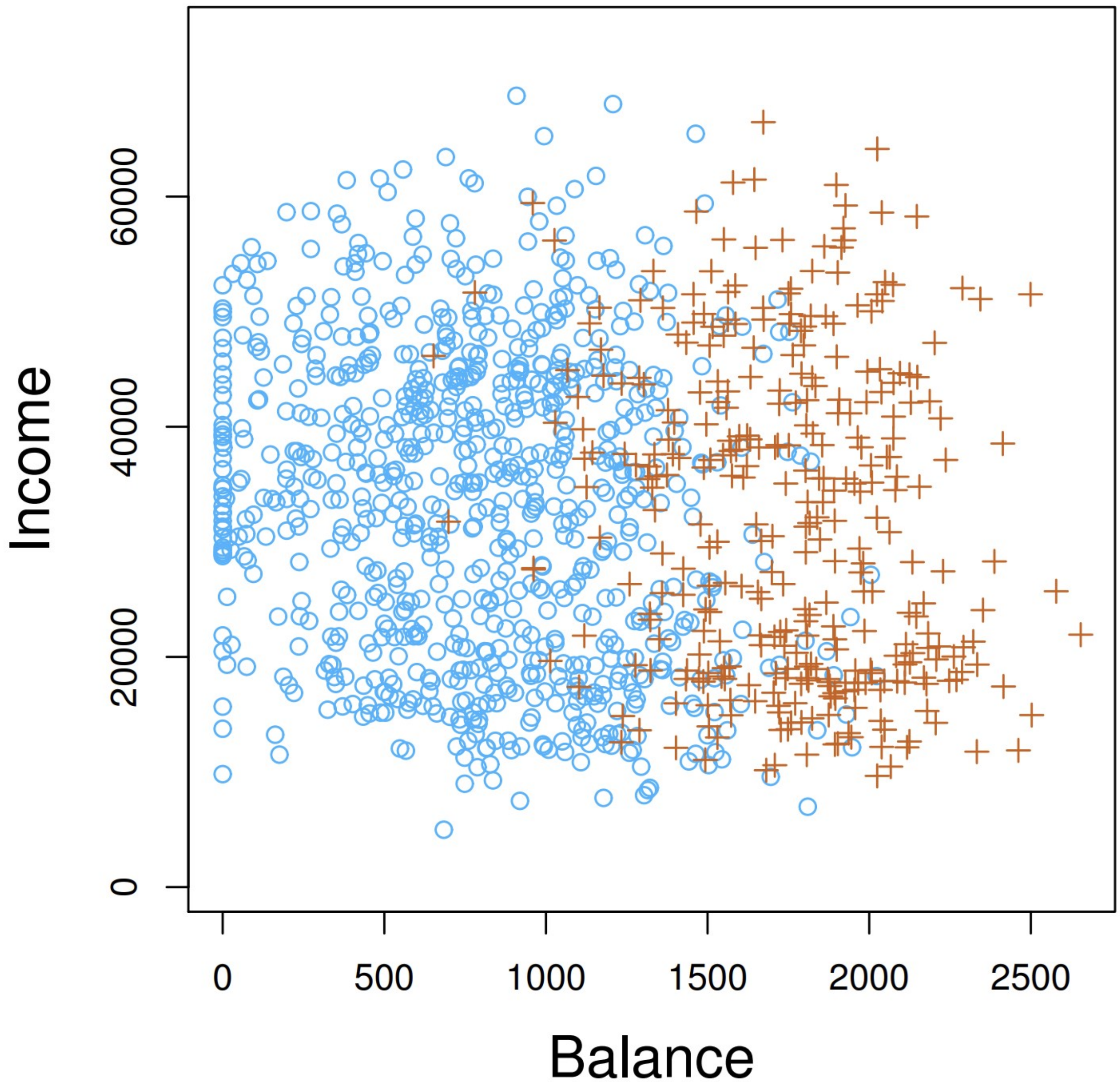
Can we use Linear Regression?

- Consider the simulated Default dataset with 10000 observations on:

  - $default$: indicating whether the customer defaulted on their debt (No and Yes)
  - $student$: indicating whether the customer is a student (No and Yes)
  - $balance$: The average balance that the customer has remaining on their credit card after making their monthly payment income
  - $income$: The income of customer
- We are interested in predicting whether an individual will default on his or her credit card payment, on the basis of annual income and monthly credit card balance.

- Suppose for the $default$ classification task that we code

$$Y = \begin{cases} 0 & if \quad No \\ 1 & if \quad Yes \end{cases}$$



Can we use Linear Regression?

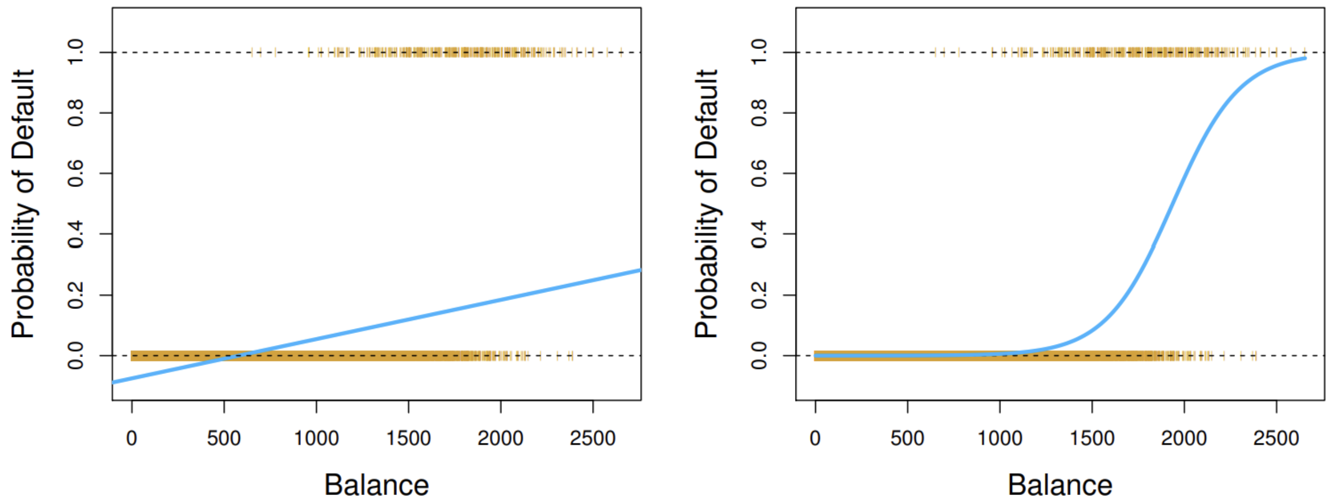
How can we simply perform a linear regression of Y on X and classify as **Yes** if $\hat{Y} > 0.5$?

- In a binary case, linear regression does a good job as a classifier, and is equivalent to *linear discriminant analysis* (LDA).
- Since in the population E(Y |X = x) = Pr(Y = 1|X = x), we might think that we can use regression.
- However, linear regression might produce probabilities less than zero or bigger than one.

  -**Logistic regression** (also called Logit Regression) ensures that our estimate for p(X) lies between 0 and 1 (is more appropriate).

See the orange marks indicating the response Y (0 or 1).

- Linear Regression does not estimate $Pr(Y = 1|X)$ well
  - Predicted Y can exceed 0 and 1 range
- Logistic Regression
  - Predicted Y lies within 0 and 1 range



In [1]:

```
# %load ../standard_import.txt
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn.linear_model as skl_lm
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, classification_report, precision_score
from sklearn import preprocessing
from sklearn import neighbors

import statsmodels.api as sm
#import statsmodels.formula.api as smf

#%matplotlib inline
#plt.style.use('seaborn-white')
import seaborn as sns
```

Logistic Regression

- Given a set of points X with classes Y
  - Assume binary classification
- We want a function $f(x)$ that finds y that maximizes $\Pr(Y = y|X = x)$

Defining f

- Models an output $0 \leq p(x) \leq 1$
  - $f(x) == class_1$ if $p(x) > 0.5$
  - $f(x) == class_2$ if $p(x) \leq 0.5$
- To **directly approximate** the Bayesian Classifier

- $p(x) \approxeq \Pr(Y = class_1 | X = x)$

Logistic Regression

- Consider
  - $p(x) \approxeq \Pr(Y = 1 | X = x)$

$$p(x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

- We can see, for any value of β_0, β_1 or X, $p(X)$ is in $[0, 1]$

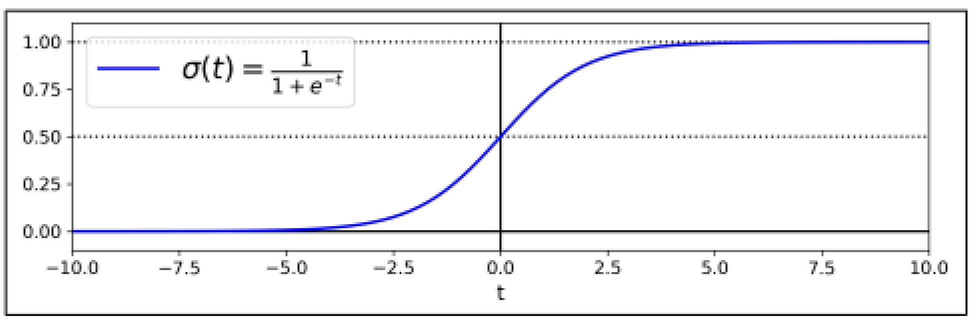
- We can also have a monotone transformationof $p(X)$ (*logit*):

$$\ln\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 x$$

- $p(x)$ is now the **logistic function** (or **sigmoid**)
- $\beta_0 + \beta_1 x$ is the linear model within linear logist equation

Remember Logistic Function?

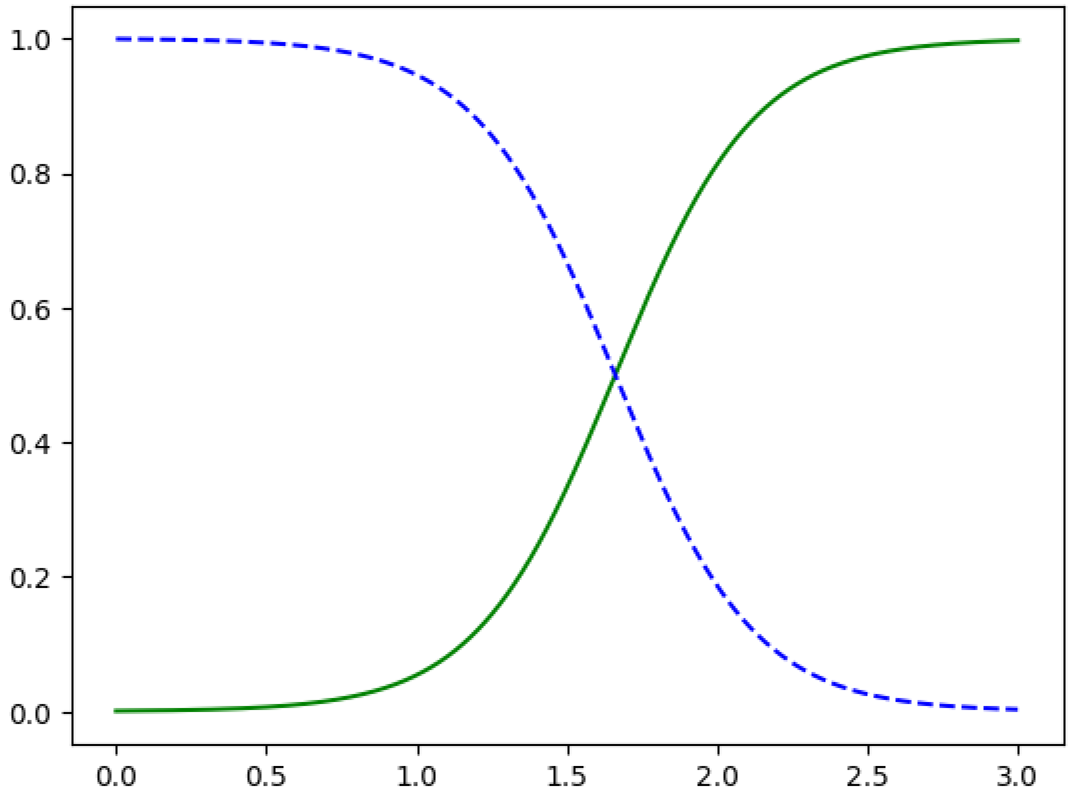
$\sigma(t) = \frac{1}{1+e^{-t}}$



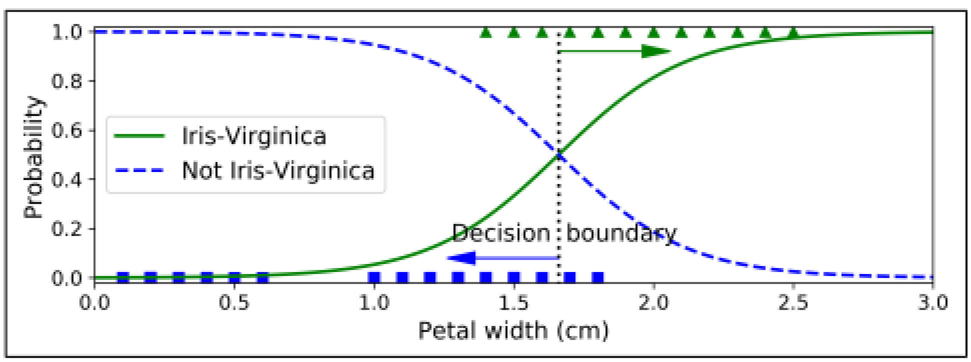
Notice that $\sigma(t) < 0.5$ when $t < 0$, and $\sigma(t) \geq 0.5$ when $t \geq 0$, so a Logistic Regression model predicts 1 if X is positive, and 0 if it is negative.

In [2]:
```
from sklearn import datasets
iris = datasets.load_iris()
list(iris.keys())
#['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename']
X = iris["data"][:, 3:] # petal width
y = (iris["target"] == 2).astype(np.int64) # 1 if Iris-Virginica, else 0

#Now lets train a Logistic Regression model:

from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X, y)

#Lets look at the model's estimated probabilities for flowers with petal widths varying
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-", label="Iris-Virginica")
plt.plot(X_new, y_proba[:, 0], "b--", label="Not Iris-Virginica")
# + more Matplotlib code to make the image look pretty
```

Out[2]: [<matplotlib.lines.Line2D at 0x182fa655510>]



Example of iris dataset: Estimated probabilities and decision boundary



In [3]:
```
#the decision boundary is about 1.6

log_reg.predict([[1.7], [1.5]])
```

Out[3]: array([1, 0], dtype=int64)

In [4]:
```
df = pd.read_csv('Default.csv')

# Note: factorize() returns two objects: a label array and an array with the unique valu
# We are only interested in the first object.
df['default2'] = df.default.factorize()[0]
df['student2'] = df.student.factorize()[0]
df.head(3)
```

Out[4]:

	default	student	balance	income	default2	student2
0	No	No	729.526495	44361.625074	0	0
1	No	Yes	817.180407	12106.134700	0	1
2	No	No	1073.549164	31767.138947	0	0

Logistic Regression

Fitting

- We can **learn** (or estimate) the parameters β
  - we use **Maximum Likelihood** (MLE) to estimate the parameters (of a model)

$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

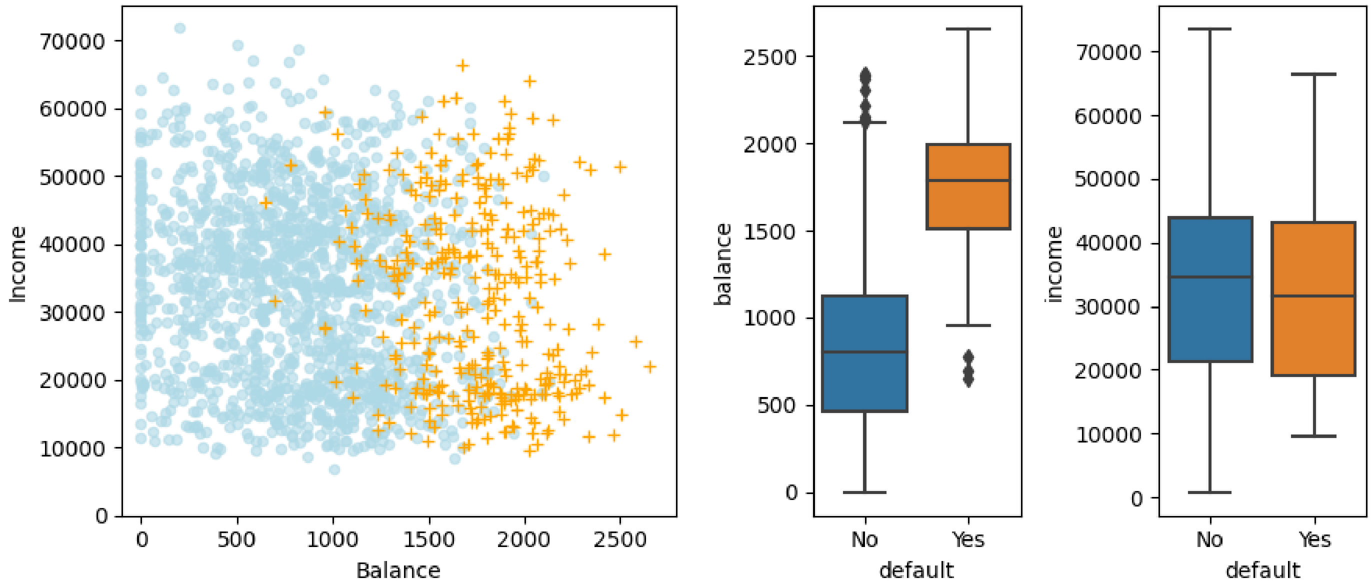
- The estimates β_0 and β_1 are chosen to maximize this likelihood function.
  - We replace $p(x)$ by the logistic expression
  - Derive the log Likelihood
  - Equal to zero
  - Obtain a closed form for the update equations
  - Use an iterative algorithm (solver) for the maximization

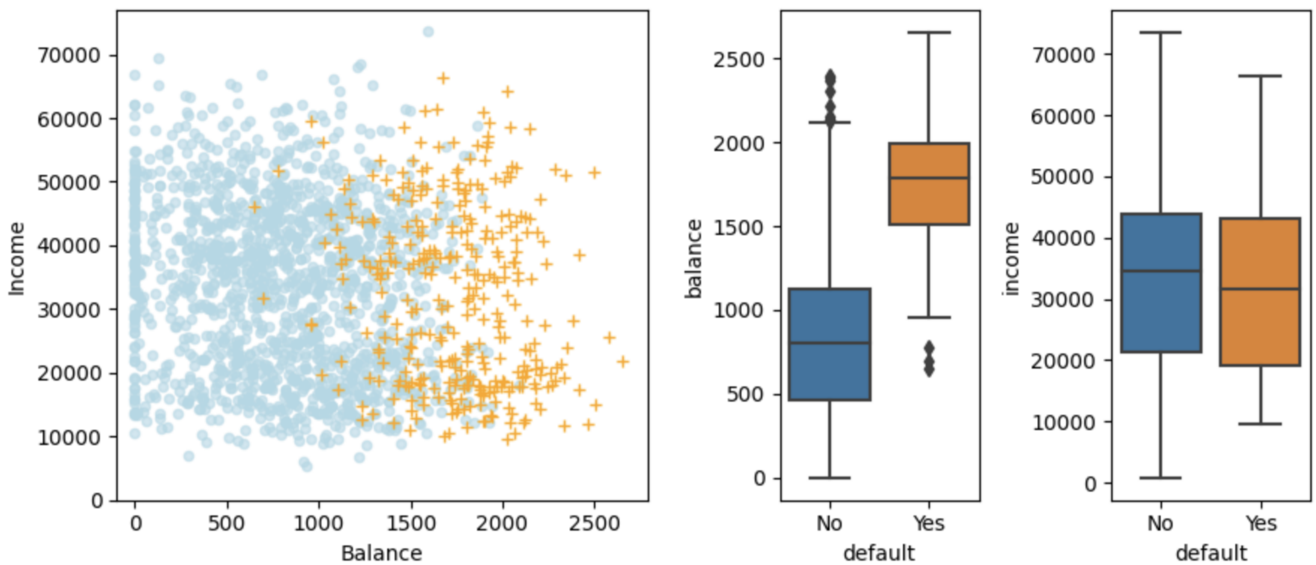
Example

Consider the Default dataset, where 'default' is one of the two categories: Yes, No.
Rather than modeling Y, Logist Regression models the probability that Y belongs to particular category.

In [5]:

```
fig = plt.figure(figsize=(9,4))
gs = mpl.gridspec.GridSpec(1, 4)
ax1 = plt.subplot(gs[0,:-2])
ax2 = plt.subplot(gs[0,-2])
ax3 = plt.subplot(gs[0,-1])

# Take a fraction of the samples where target value (default) is 'no'
df_no = df[df.default2 == 0].sample(frac=0.15)

# Take all samples  where target value is 'yes'
df_yes = df[df.default2 == 1]

df_ = pd.concat([df_no, df_yes])

ax1.scatter(df_[df_.default == 'No'].balance, df_[df_.default == 'No'].income, s=20, c='
ax1.scatter(df_[df_.default == 'Yes'].balance, df_[df_.default == 'Yes'].income, s=40, c

ax1.set_ylim(ymin=0)
ax1.set_ylabel('Income')
ax1.set_xlim(xmin=-100)
ax1.set_xlabel('Balance')

c_palette = {'No':'lightblue', 'Yes':'orange'}

sns.boxplot(x = 'default', y = 'balance', data = df, ax=ax2)
sns.boxplot(x = 'default', y = 'income', data = df, ax=ax3)
#https://www.geeksforgeeks.org/box-plot-visualization-with-pandas-and-seaborn/

gs.tight_layout(plt.gcf())
```





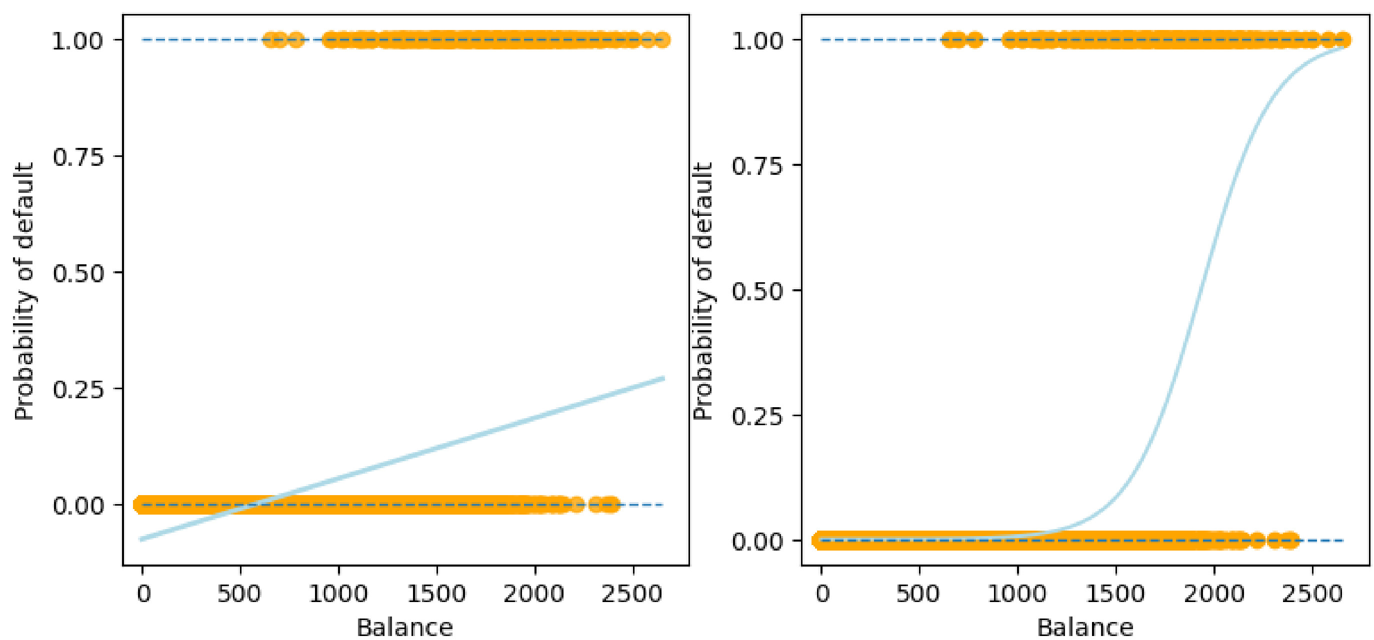
In the figure above we can see:

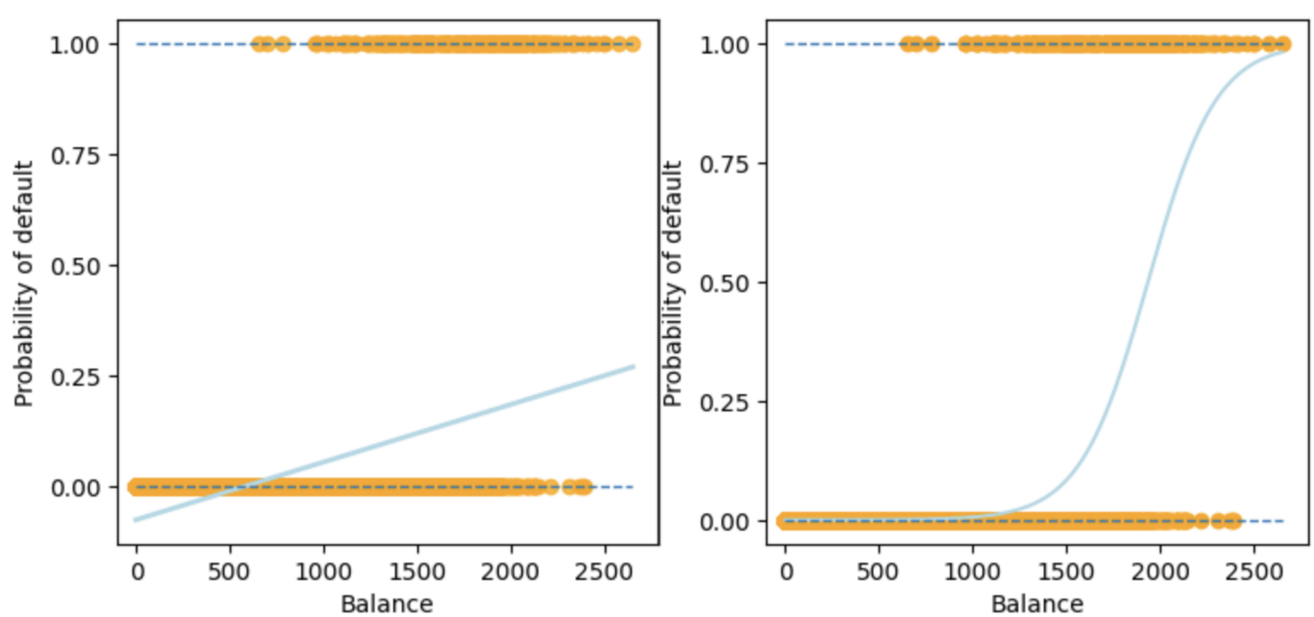
- at the left, the anual incomes and credit card balances of a number of individuals. The individuals who faulted on their payments are in orange and those who did not are in blue.
- at the center, the boxplots of balance as function of default status;
- at the right, the boxplots of income as function of default status.

In [6]:

```
### train and test set

X_train = df.balance.values.reshape(-1,1)
y = df.default2

# Create array of test data. Calculate the classification probability
# and predicted classification.

X_test = np.arange(df.balance.min(), df.balance.max()).reshape(-1,1)

clf = skl_lm.LogisticRegression()

clf.fit(X_train,y)

prob = clf.predict_proba(X_test)

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(9,4))
```

```python
# Left plot
sns.regplot(x ='balance', y = 'default2', data = df, order=1, ci=None, scatter_kws={'col

# Right plot
ax2.scatter(X_train, y, color='orange')
ax2.plot(X_test, prob[:,1], color='lightblue')

for ax in fig.axes:
    ax.hlines(1, xmin=ax.xaxis.get_data_interval()[0],
              xmax=ax.xaxis.get_data_interval()[1], linestyles='dashed', lw=1)
    ax.hlines(0, xmin=ax.xaxis.get_data_interval()[0],
              xmax=ax.xaxis.get_data_interval()[1], linestyles='dashed', lw=1)
    ax.set_ylabel('Probability of default')
    ax.set_xlabel('Balance')
    ax.set_yticks([0, 0.25, 0.5, 0.75, 1.])
    ax.set_xlim(xmin=-100)
```





In the figure we can see the result of the Classification using the Default dataset. The orange ticks indicate 0/1 values coded by default (No/Yes). We can see predicted probabilities of default using:

- Linear Regression (with some negative values...), on the left;

- Logistic Regression, where all probabilities betweene 0 and 1, on the right, .

In [7]:
```
###
###  Making Predictions
###

#Estimating the coefficients of the logist regression model that estimates the default u

y = df.default2

#Using scikit-learn

clf = skl_lm.LogisticRegression()

#Reshape data using array.reshape(-1, 1) if your data has a single feature or array.resh
X_train = df.balance.values.reshape(-1,1)

clf.fit(X_train,y)

print(clf)
print('classes: ',clf.classes_)
print('coefficients: ',clf.coef_)
print('intercept :', clf.intercept_)
```

```
LogisticRegression()
classes:  [0 1]
coefficients:  [[0.00549892]]
intercept : [-10.65132824]
```

Making predictions:

- Considering these parameters, what is our estimated probability of default for someone with a balance of $1000?

$$p(1000) = \frac{e^{-10.6513+0.0055x1000}}{1+e^{10.6513+0.0055x1000}} = \frac{1}{1+e^{-(10.6513+0.0055x1000)}} = 0.006$$

In [8]:
```
##### Using statsmodels

X_train = sm.add_constant(df.balance)
est = sm.Logit(y.ravel(), X_train).fit()
est.summary2().tables[1]
```

```
Optimization terminated successfully.
         Current function value: 0.079823
         Iterations 10
```

Out[8]:

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	-10.651331	0.361169	-29.491287	3.723665e-191	-11.359208	-9.943453
balance	0.005499	0.000220	24.952404	2.010855e-137	0.005067	0.005931

For the Default data, estimated coefcients of the logistic regression model that predicts the probability of default using balance. A one-unit increase in balance is associated with an increase in the log odds of default by 0.0055 units.

For the Default data, estimated coeficients of the logistic regression model that predicts the probability of default using student status. Student status is encoded as a dummy variable, with a value of 1 for a student and a value of 0 for a non-student, and represented by the variable student[Yes] in the table.

In [9]:
```python
X_train = sm.add_constant(df.student2)
y = df.default2

est = sm.Logit(y, X_train).fit()
est.summary2().tables[1]
```

```
Optimization terminated successfully.
         Current function value: 0.145434
         Iterations 7
```

Out[9]:

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	-3.504128	0.070713	-49.554094	0.000000	-3.642723	-3.365532
student2	0.404887	0.115019	3.520177	0.000431	0.179454	0.630320

Multiple Logistic Regression

Logistic Regression with several variables

$$\ln\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

$$p(x) = \frac{e^{\beta_0+\beta_1 x_1+\ldots+\beta_p x_p}}{1+e^{\beta_0+\beta_1 x_1+\ldots+\beta_p x_p}}$$

Example: Confounding

- Students tend to have higher balances than non-students, so their marginal default rate is higher than for non-students.
- For each level of balance, students default less than non-students.
- Multiple logistic regression can highlight this out

In [10]:
```python
X_train = sm.add_constant(df[['balance', 'income', 'student2']])
est = sm.Logit(y, X_train).fit()
est.summary2().tables[1]
```

```
Optimization terminated successfully.
         Current function value: 0.078577
         Iterations 10
```

Out[10]:

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	-10.869045	0.492273	-22.079320	4.995499e-108	-11.833882	-9.904209
balance	0.005737	0.000232	24.736506	4.331521e-135	0.005282	0.006191
income	0.000003	0.000008	0.369808	7.115254e-01	-0.000013	0.000019
student2	-0.646776	0.236257	-2.737595	6.189022e-03	-1.109831	-0.183721

In [11]:
```python
# balance and default vectors for students
X_train = df[df.student == 'Yes'].balance.values.reshape(df[df.student == 'Yes'].balance
y = df[df.student == 'Yes'].default2

# balance and default vectors for non-students
X_train2 = df[df.student == 'No'].balance.values.reshape(df[df.student == 'No'].balance.
y2 = df[df.student == 'No'].default2

# Vector with balance values for plotting
```

```
X_test = np.arange(df.balance.min(), df.balance.max()).reshape(-1,1)

clf = skl_lm.LogisticRegression(solver='newton-cg')
clf2 = skl_lm.LogisticRegression(solver='newton-cg')

clf.fit(X_train,y)
clf2.fit(X_train2,y2)

prob = clf.predict_proba(X_test)
prob2 = clf2.predict_proba(X_test)
```

In [12]:

```
df.groupby(['student','default']).size().unstack('default')
```

Out[12]:

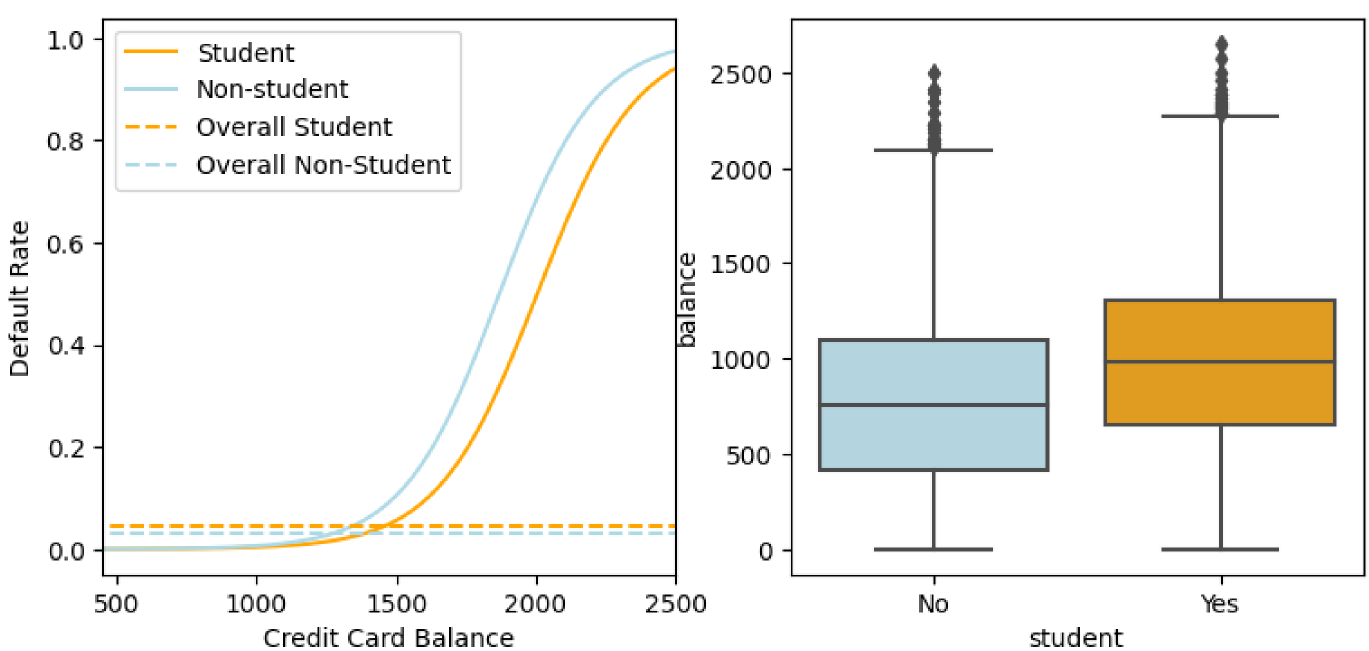
default	No	Yes
student		
No	6850	206
Yes	2817	127

In [13]:

```
# creating plot
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(9,4))

# Left plot
ax1.plot(X_test, pd.DataFrame(prob)[1], color='orange', label='Student')
ax1.plot(X_test, pd.DataFrame(prob2)[1], color='lightblue', label='Non-student')
ax1.hlines(127/2817, colors='orange', label='Overall Student',
           xmin=ax1.xaxis.get_data_interval()[0],
           xmax=ax1.xaxis.get_data_interval()[1], linestyles='dashed')
ax1.hlines(206/6850, colors='lightblue', label='Overall Non-Student',
           xmin=ax1.xaxis.get_data_interval()[0],
           xmax=ax1.xaxis.get_data_interval()[1], linestyles='dashed')
ax1.set_ylabel('Default Rate')
ax1.set_xlabel('Credit Card Balance')
ax1.set_yticks([0, 0.2, 0.4, 0.6, 0.8, 1.])
ax1.set_xlim(450,2500)
ax1.legend(loc=2)

# Right plot
sns.boxplot(x = 'student', y = 'balance', data = df, ax=ax2, palette=c_palette)
```

Out[13]:

```
<Axes: xlabel='student', ylabel='balance'>
```



Why Discriminant Analysis?

- When there is substantial separation between the two classes, the parameter estimates for the Logistic Regression model are surprisingly unstable.
  - Linear Discriminant Analysis does not have this problem.
- If the distribution of the predictors X is approximately normal in each of the classes (and the sample size is small), Linear Discriminant Analysis is more stable than Logistic Regression.
- Linear Discriminant Analysis is popular in the case of more than two response classes.

Discriminant Analysis

- Linear Discriminant Analysis (LDA) is
  - a machine learning algorithm;
  - used to find the Linear Discriminant function that best classifies or discriminates or separates two classes of data
  - The main purpose of LDA is
    - to find the line (or plane) that best separates data points belonging to different classes
  - The key idea behind LDA is
    - that the decision boundary should be chosen such that it (Fisher criterion)
      - maximizes the distance between the means of the two classes
      - minimizing the variance within each classes data or within-class scatter.

Discriminant Analysis

- Given a set of points X with classes Y
  - Assume binary classification
- We want a function $f(x)$ that finds y that maximizes $\Pr(Y = y|X = x)$

The general idea

- Suppose we know the exact distribution of the points in X for each class
  - $\Pr(X = x|Y = class_1)$ and $\Pr(X = x|Y = class_2)$
- Suppose we have 1 predictor
- Given a point x we can use the *Bayes theorem*

$$\Pr(Y = k|X = x) = \frac{\Pr(X = x|Y = k)\Pr(Y = k)}{\Pr(X = x)}$$

or

$$\Pr(Y=k|X=x)=\frac{\pi_k f_k(x)}{\sum \pi_l f_l(x)}$$

Here the approach is to model the distribution of X in each of the classes separately, and then use Bayes theorem to flip things around and obtain Pr(Y|X). When we use normal (Gaussian) distributions for each class, this leads to linear or quadratic discriminant analysis. However, this approach is quite general, and other distributions can be used as well. We will focus on normal distributions.

Discriminant Analysis

- Given a set of points X with classes Y we don't know the true probabilities
- We estimate them from the data
  - $\Pr(Y=k)$ from the proportion of the classes
  - We **assume** that $\Pr(X=x|Y=k)$ is **Gaussian** and estimate **parameters**:
    - μ_k from the points in each class
    - σ^2 by averaging the σ_k^2 for each class
  - We do not need to know $\Pr(X=x)$
  - We are actually using **MLE** to estimate μ_k and σ^2
- If we have more than one attribute in X
  - We estimate **covariance** instead of variance

Discriminant Analysis

Linear Discriminant Analysis

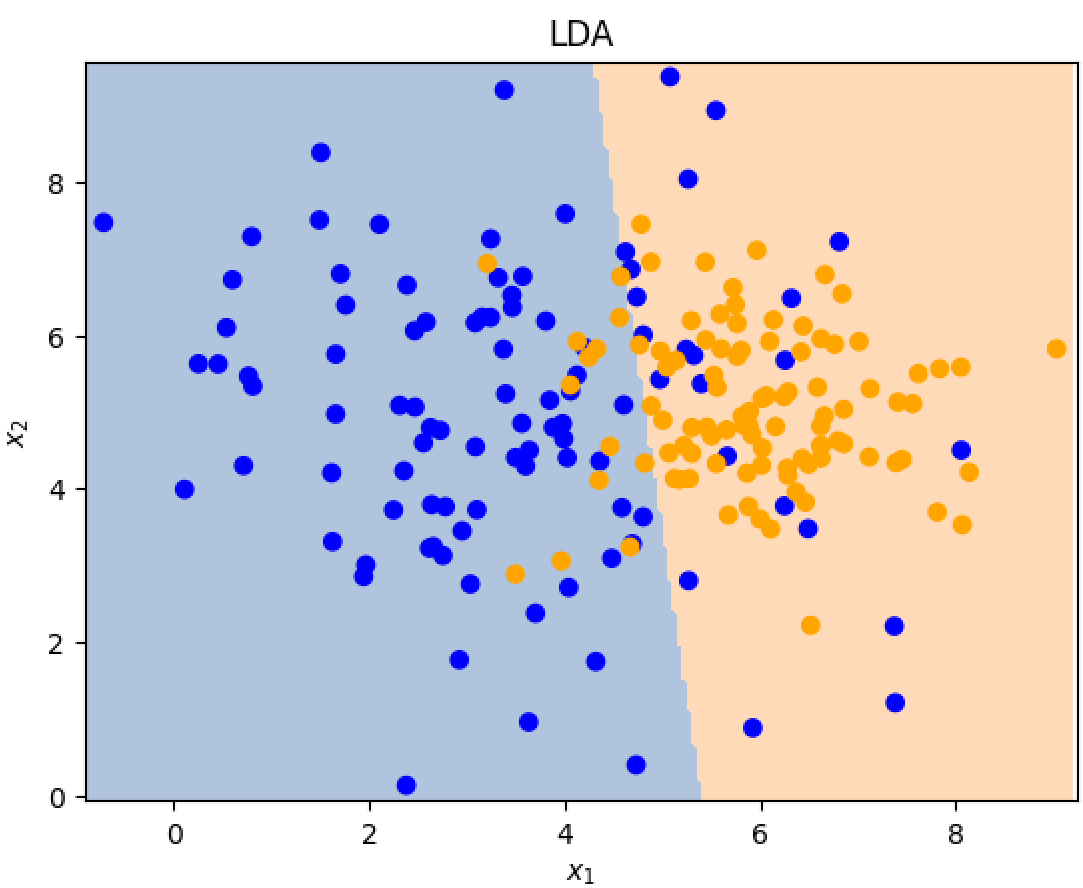
- We assume that all the densities $\Pr(X=x|Y=k)$ have the same **covariance**
  - This results in linear boundaries

Quadratic Discriminant Analysis

- We relax the assumption of same covariance for each class
  - This results in quadratic boundaries

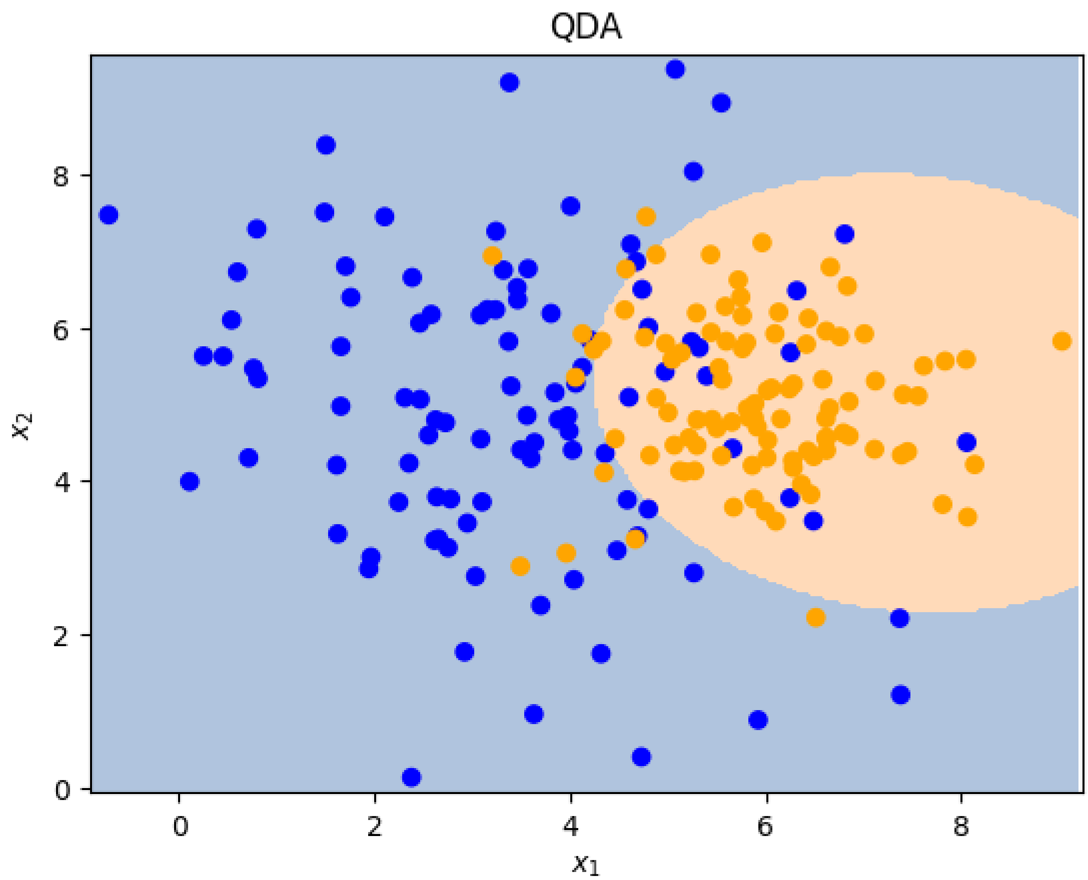
In [14]:
```
from sklearn.datasets import make_blobs
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
import matplotlib.pyplot as plt
from matplotlib import colors

def plot_classifier_boundary(model,X,h = .05):
    # this function can be used with any sklearn classifier
    # ready for two classes but can be easily extended
    cmap_light = colors.ListedColormap(['lightsteelblue', 'peachpuff'])
    x_min, x_max = X[:, 0].min()-.2, X[:, 0].max()+.2
    y_min, y_max = X[:, 1].min()-.2, X[:, 1].max()+.2
    # generate a grid with step h
```

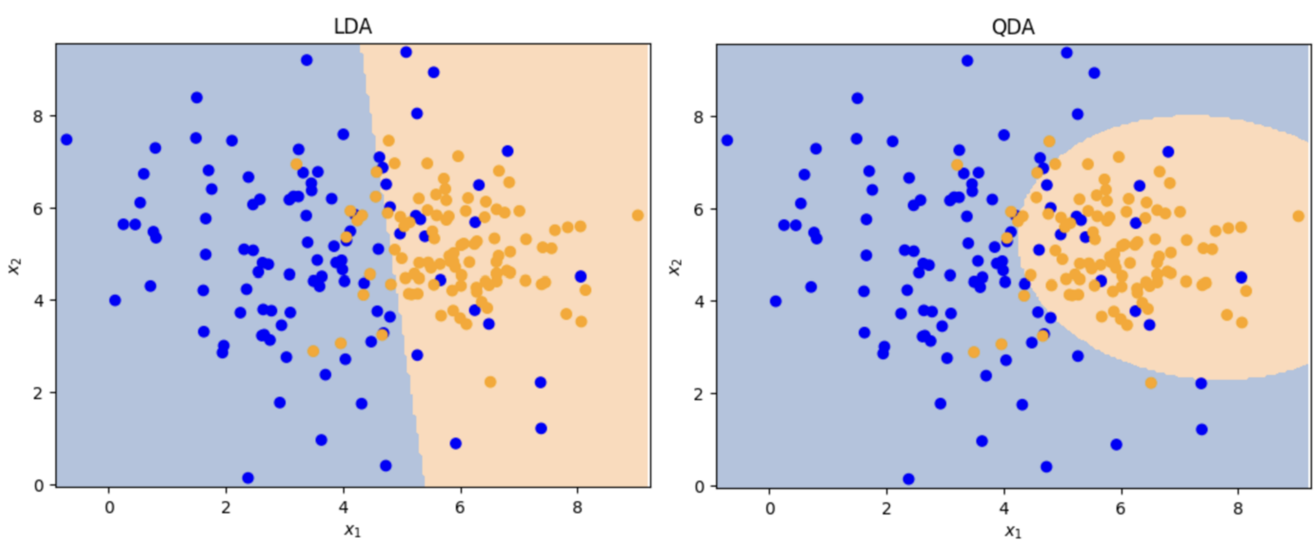
```
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    # the method ravel flattens xx and yy
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=cmap_light)
    plt.xlim((x_min,x_max))
    plt.ylim((y_min,y_max))

n_points=100
std1=2
std2=1


X,y = make_blobs(n_samples=[n_points,n_points], centers=[(3,5),(6,5)],
                 n_features=2, cluster_std=[std1,std2],
                 random_state=1, shuffle=False)

cmap = colors.ListedColormap(['blue','orange'])

logr=skl_lm.LogisticRegression().fit(X,y)

plot_classifier_boundary(logr,X)

plt.scatter(X[:,0],X[:,1],color=cmap(y))
plt.title('LDA')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$');
```



In [15]:
```
discr=QDA().fit(X,y)

plot_classifier_boundary(discr,X)

plt.scatter(X[:,0],X[:,1],color=cmap(y))
plt.title('QDA')
```

```
plt.xlabel('$x_1$')
plt.ylabel('$x_2$');
```





Confusion matrix

A confusion matrix compares the LDA predictions to the true default statuses for the 10000 training observations in the Default dataset. Elements of the diagonal of the matrix represent individuals whose default statuses were correctly predicted, while of-diagonal elements represent individuals that were misclassifed. LDA made incorrect predictions for 23 individuals who did not default and for 252 individuals who did default.

In [16]:
```
X = df[['balance', 'income', 'student2']].to_numpy()
y = df.default2.to_numpy()
```

```python
#lda = LinearDiscriminantAnalysis(solver='svd')
lda = LinearDiscriminantAnalysis()
y_pred = lda.fit(X, y).predict(X)

df_ = pd.DataFrame({'True default status': y,
                    'Predicted default status': y_pred})
df_.replace(to_replace={0:'No', 1:'Yes'}, inplace=True)

df_.groupby(['Predicted default status','True default status']).size().unstack('True def
```

Out[16]:

True default status	No	Yes
Predicted default status		
No	9645	254
Yes	22	79

In [17]:

```python
print(classification_report(y, y_pred, target_names=['No', 'Yes']))
```

```
              precision    recall  f1-score   support

          No       0.97      1.00      0.99      9667
         Yes       0.78      0.24      0.36       333

    accuracy                           0.97     10000
   macro avg       0.88      0.62      0.67     10000
weighted avg       0.97      0.97      0.97     10000
```

Instead of using the probability of 50% as decision boundary, we say that a probability of default of 20% is to be classified as 'Yes'.

In [18]:

```python
decision_prob = 0.2
y_prob = lda.fit(X, y).predict_proba(X)

df_ = pd.DataFrame({'True default status': y,
                    'Predicted default status': y_prob[:,1] > decision_prob})
df_.replace(to_replace={0:'No', 1:'Yes', 'True':'Yes', 'False':'No'}, inplace=True)

df_.groupby(['Predicted default status','True default status']).size().unstack('True def
```

Out[18]:

True default status	No	Yes
Predicted default status		
False	9435	140
True	232	193

Summary

- Logistic regression is very popular for classification, especially when K = 2.
- LDA is useful when n is small, or the classes are well separated, and Gaussian assumptions are reasonable. Also when K > 2.