Applied Statistical Methods

Classification - Part I

Xuemao Zhang
East Stroudsburg University

April 10, 2023

Outline

- Introduction
- K-Nearest Neighbors
- Logistic Regression

Introduction

- Classification involves predicting a **categorical/qualitative response**:
  - Cancer versus Normal
  - Tumor Type 1 versus Tumor Type 2 versus Tumor Type 3
- Classification problems tend to occur even more frequently than regression problems in biomedical applications.
- Categorical/qualitative variables take values in an unordered set: e.g.
  - eye color $\in$ {brown, blue, green}
  - email $\in$ {spam; not spam}.
- We want to build a function that $C(X)$ takes as input the feature vector $X = (X_1, \ldots, X_p)$ and predicts the value for Y, i.e. $C(X)$ is in which category.
- Often we are more interested in estimating the probability that X belongs to a given category.
  - For example: we might want to know the probability that someone will develop diabetes, rather than to predict whether or not they will develop diabetes.

Introduction

- We used training data to conduct classifications.
- What we really care about is how well the method works on new data. We call this new data "**Test Data**".
- Let n be the total number observations. For example, the data $(x1, y1), \ldots, (x_n, y_n)$. And let $\hat{y}$ be our estimate. Then the training **error rate**, the proportion of mistakes that are made

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i).$$

- We are most interested in the error rates that result from applying our classifier to test observations that were not used in training. The **test error rate** associated with a set of test observations of the form test error (x_0, y_0) is given by

$$Ave(I(y_0 \neq \hat{y}_0)).$$

K-Nearest Neighbors

- Can we take a totally non-parametric (model-free) approach to classification?
- K-nearest neighbors (KNN):
  - 1. For any given X_0, identify the k observations whose X values are *closest to the observation X_0 at which we want to make a prediction.*
  - 2. Classify the observation of interest X_0 to the most frequent class label of those K nearest neighbors: If the majority of the Y's are orange we predict orange otherwise guess blue.
- The smaller that k is the more flexible the method will be.

K-Nearest Neighbors

- Example: K-nearest neighbors in two dimensions

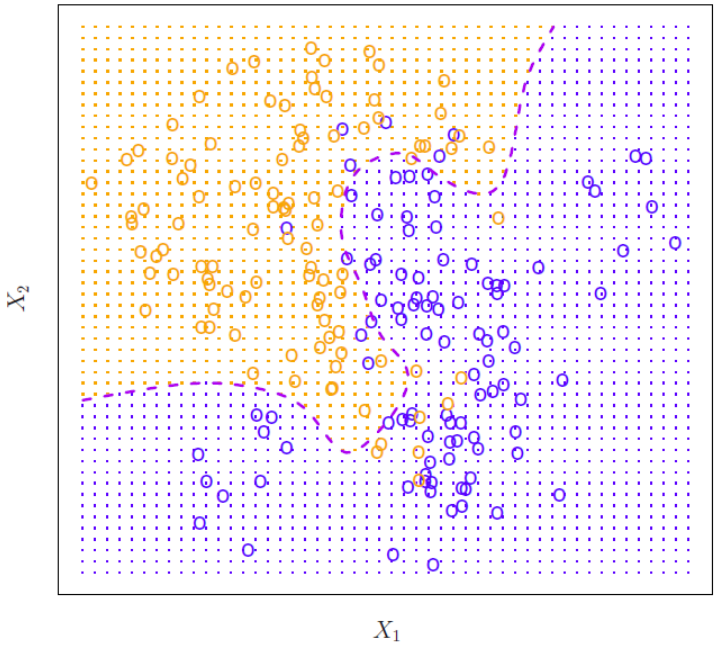
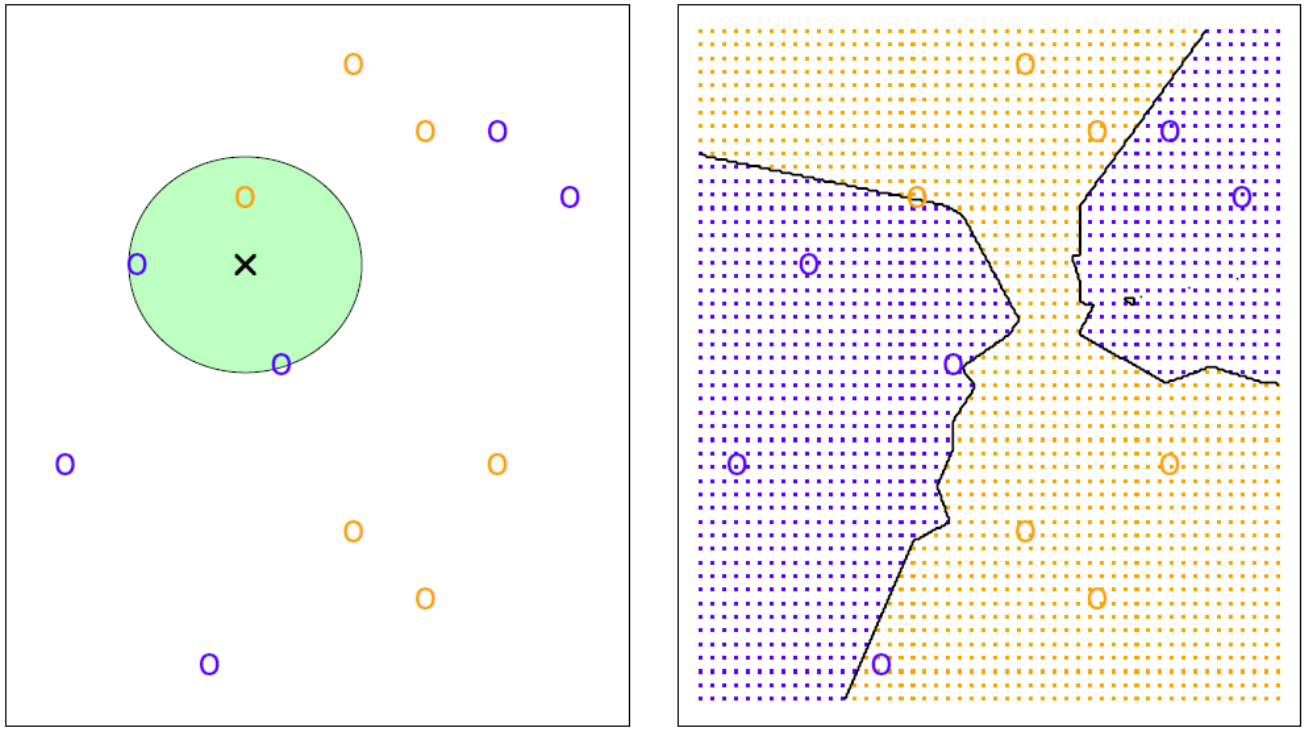


Figure 1: A simulated data set consisting of 100 observations in each of two groups, indicated in blue and in orange. The purple dashed line represents the Bayes decision boundary. The orange background grid indicates the region in which a test observation will be assigned to the orange class, and the blue background grid indicates the region in which a test observation will be assigned to the blue class.

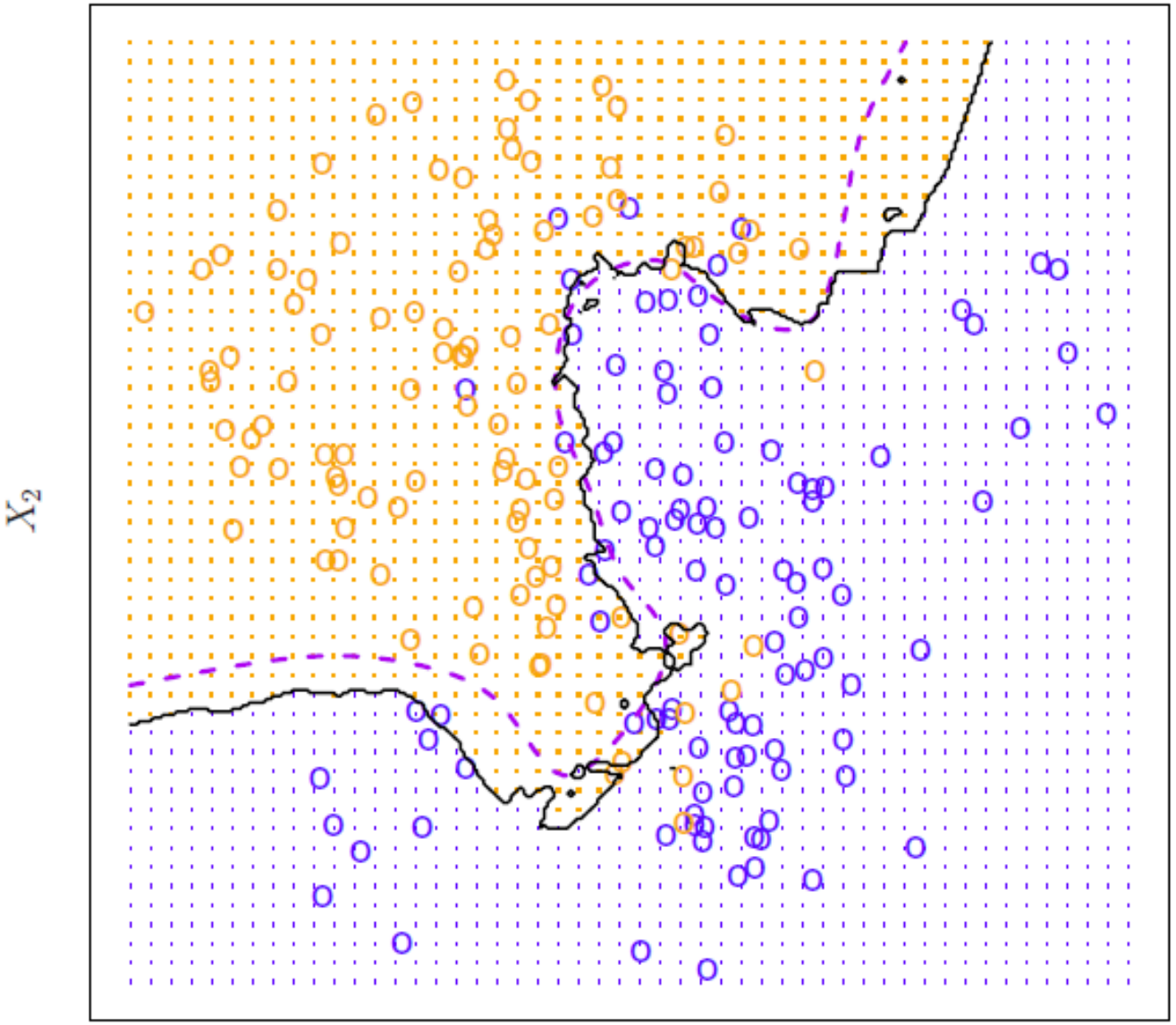
K-Nearest Neighbors

- Example: KNN Example with $k = 3$



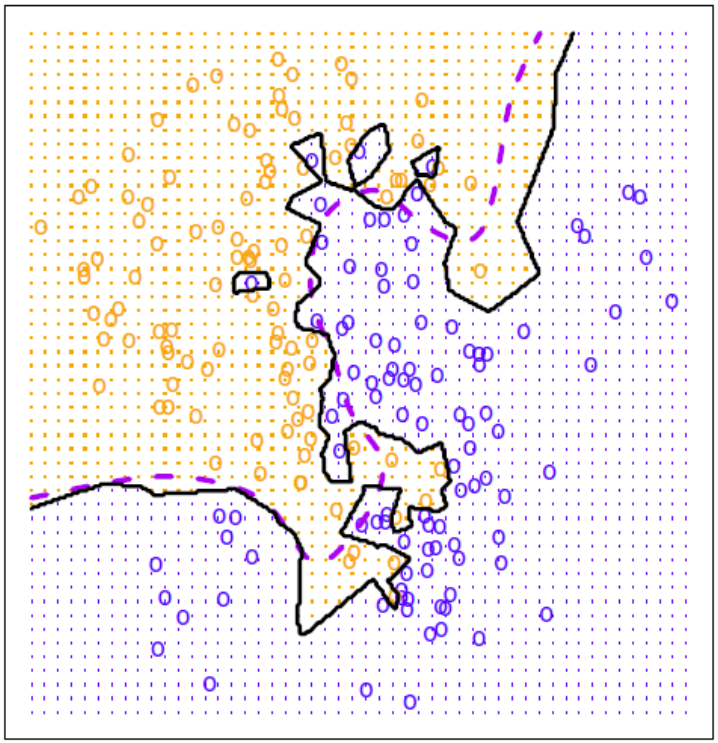
K-Nearest Neighbors

- Example: KNN Example with $k = 10$

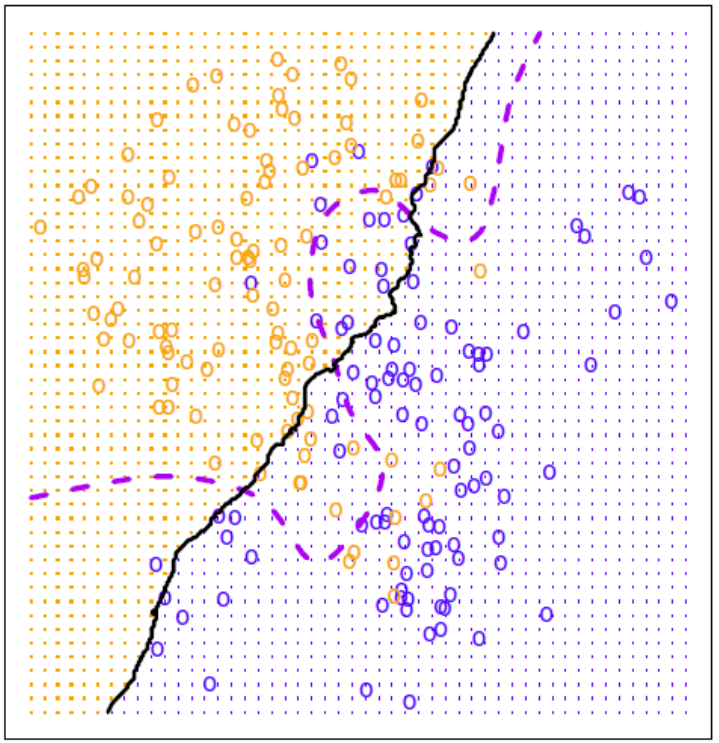


K-Nearest Neighbors

KNN: K=1



KNN: K=100



K-Nearest Neighbors

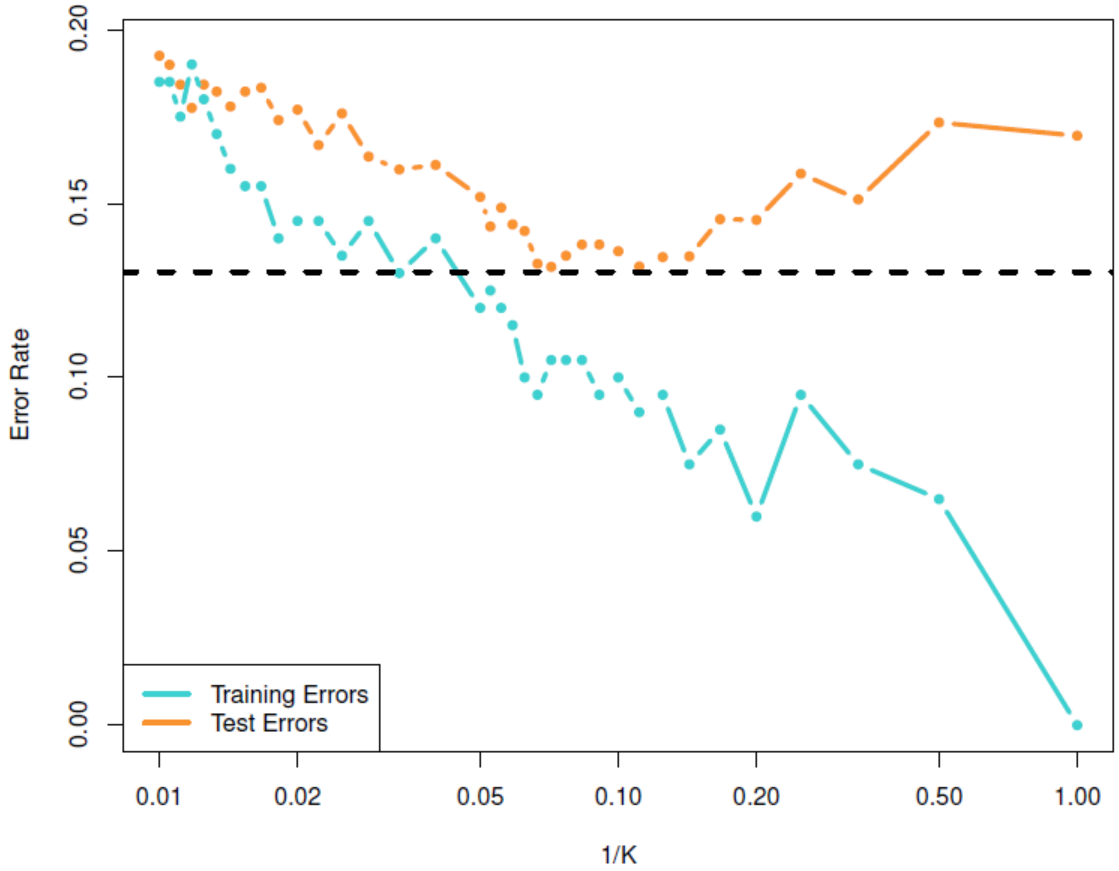
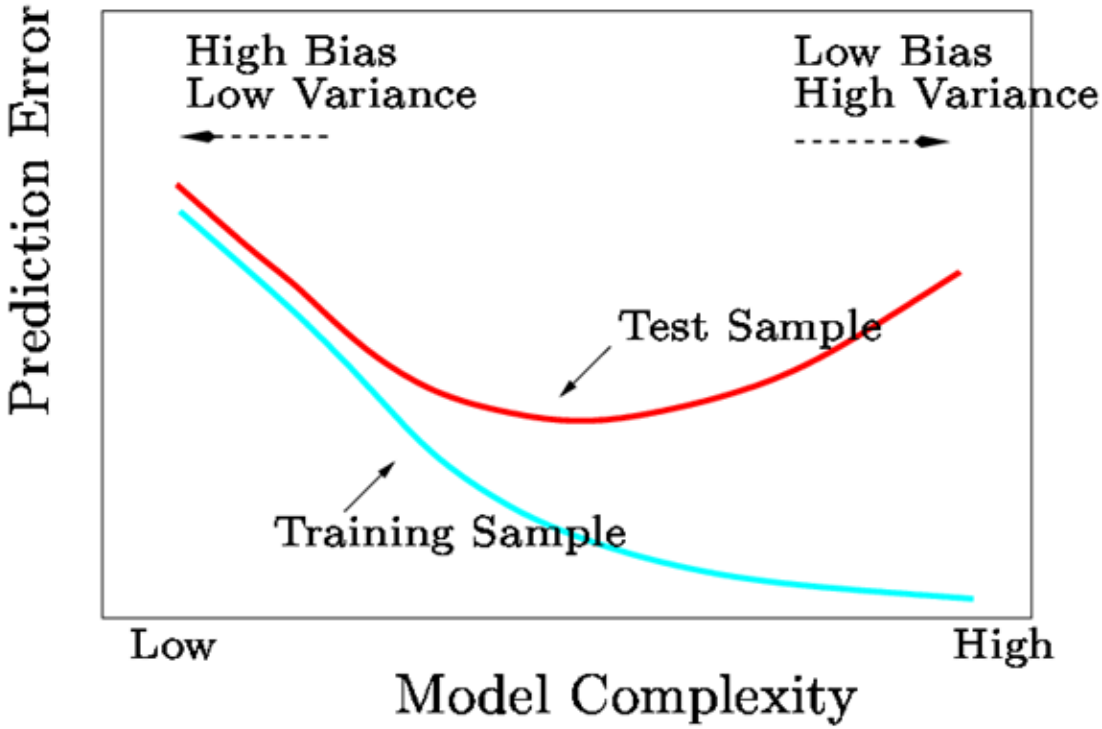


Figure 2: The KNN training error rate (blue, 200 observations) on the data from Figure 1 and test error rate (orange, 5,000 observations), as the level of flexibility (assessed using $1/K$) increases, or equivalently as the number of neighbors K decreases. The black dashed line indicates the Bayes error rate. The jumpiness of the curves is due to the small size of the training data set.

A Fundamental Picture

- In general training errors will always decline.
- However, test errors will decline at first (as reductions in bias dominate) but will then start to increase again (as increases in variance dominate).
- We must always keep the picture mentioned already in mind when choosing a learning method. More flexible/complicated is not always better!



K-Nearest Neighbors

- Simple, intuitive, model-free.
- Good option when p is very small.
- Curse of dimensionality: when p is large, no neighbours are "near". All observations are close to the boundary.

K-Nearest Neighbors

- Consider the Stock Market Data in R package `ISLR`

```
import pandas as pd
import numpy as np
df = pd.read_csv('../data/Smarket.csv')
df.head()
```

```
##    Year   Lag1   Lag2   Lag3   Lag4   Lag5  Volume  Today Directi
## 0  2001  0.381 -0.192 -2.624 -1.055  5.010  1.1913  0.959
## 1  2001  0.959  0.381 -0.192 -2.624 -1.055  1.2965  1.032
## 2  2001  1.032  0.959  0.381 -0.192 -2.624  1.4112 -0.623      Do
## 3  2001 -0.623  1.032  0.959  0.381 -0.192  1.2760  0.614
## 4  2001  0.614 -0.623  1.032  0.959  0.381  1.2057  0.213
```

K-Nearest Neighbors

- `Lag1` through `Lag5`: the percentage returns for each of the five previous trading days
- `Volume`: the number of shares traded on the previous day, in billions
- `Today`: the percentage return on the date in question
- `Direction`: whether the market was Up or Down on the date `Today`

- The goal is to predict whether the index will increase or decrease on a given day using the past 5 days' percentage changes in the index.

K-Nearest Neighbors

- We'll build our model using the `KNeighborsClassifier()` function, which is part of the neighbors submodule of `SciKitLearn (sklearn)`. We'll also grab a couple of useful tools from the `metrics` sub-module

```
from sklearn import neighbors
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

- We split the data as training data and test data, and use the test data to check prediction accuracy.

K-Nearest Neighbors

- We'll first create a vector corresponding to the observations from 2001 through 2004, which we'll use to train the model. We will then use this vector to create a held out data set of observations from 2005 on which we will test. We'll also pull out our training and test labels.

```
df.Year.unique()
```

```
## array([2001, 2002, 2003, 2004, 2005], dtype=int64)
```

```
X_train = df[df.Year<=2004][['Lag1','Lag2']]
y_train = df[df.Year<=2004]['Direction']

X_test = df[df.Year==2005][['Lag1','Lag2']]
y_test = df[df.Year==2005]['Direction']
```

K-Nearest Neighbors

- Now the `neighbors.KNeighborsClassifier()` function can be used to fit a model.
  - https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- To ensure reproducibility of results, we set a random seed before we apply `neighbors.KNeighborsClassifier()` because if several observations are tied as nearest neighbors, then Python will **randomly** break the tie.

```
from numpy import random
random.seed(1)
knn = neighbors.KNeighborsClassifier(n_neighbors = 1)
knn_fit = knn.fit(X_train, y_train)
print(knn_fit.effective_metric_)
```

```
## euclidean
```

```
print(knn_fit.classes_)
```

```
## ['Down' 'Up']
```

- The model is used to predict the market's movement for the dates in 2005.

```
pred = knn_fit.predict(X_test)
```

K-Nearest Neighbors

- The `confusion_matrix()` function can be used to produce a confusion matrix in order to determine how many observations were correctly or incorrectly classified. The `classification_report()` function gives us some summary statistics on the classifier's performance.

```
print(confusion_matrix(y_test, pred).T) #(43+83)/252
```

```
## [[43 58]
##  [68 83]]
```

```
print(classification_report(y_test, pred, digits=3))
```

```
##               precision    recall  f1-score   support
##
##         Down      0.426     0.387     0.406       111
##           Up      0.550     0.589     0.568       141
##
##     accuracy                          0.500       252
##    macro avg      0.488     0.488     0.487       252
## weighted avg      0.495     0.500     0.497       252
```

- Or we can use `accuracy_score` from the module `sklearn.metrics`.

```
print(accuracy_score(pred, y_test))
```

```
## 0.5
```

K-Nearest Neighbors

- The results using $k = 1$ are not very good, since only 50% of the observations are correctly predicted. Of course, it may be that $k = 1$ results in an overly flexible fit to the data. Below, we repeat the analysis using $k = 3$.

```
from numpy import random
random.seed(1)
knn = neighbors.KNeighborsClassifier(n_neighbors=3)
pred = knn.fit(X_train, y_train).predict(X_test)
print(accuracy_score(pred, y_test))
```

```
## 0.5317460317460317
```

```
print(confusion_matrix(y_test, pred).T)
```

```
## [[48 55]
##  [63 86]]
```

```
print(classification_report(y_test, pred, digits=3))
```

```
##               precision    recall  f1-score   support
##
##         Down      0.466     0.432     0.449       111
##           Up      0.577     0.610     0.593       141
##
##     accuracy                          0.532       252
##    macro avg      0.522     0.521     0.521       252
## weighted avg      0.528     0.532     0.529       252
```

Logistic Regression

- Let $\mathcal{C}$ be the set of collection of responses of Y. For example, email is one of $\mathcal{C}$ = (spam, not spam), digit class is one of $\mathcal{C}$= $\{0, 1, \ldots, 9\}$.
- Is there an ideal $C(X)$? Suppose the K elements in $\mathcal{C}$ are numbered $1, 2, \ldots, K$. For any x, let

$$p_k(x) = P(Y = k|X = x), k = 1, \ldots, K.$$

  These are the **conditional class probabilities** at $X = x$.

- Then the **Bayes classifier** at x is

$$C(x) = j, \text{ if } p_j(x) = \max\{p_1(x), \ldots, p_K(x)\}$$

  For responses with two categories, we just check which probability is greater than 50%.

Logistic Regression

- The Bayes classifier produces the lowest possible test error rate, called the Bayes error rate.
- The error rate of the Bayes classifier at $X = x_0$ will be

$$1 - \max_j Pr(Y = j|X = x_0).$$

- In general, the overall Bayes error rate is given by

$$1 - E\left(\max_j Pr(Y = j|X)\right)$$

- We can build parametric models for representing the conditional class probabilities $p_k(x)$ to construct a Bayes classifier.
- Logistic regression is such a method.

Logistic Regression

- Example: Credit Card Default

```
import pandas as pd
Default=pd.read_csv("../data/Default.csv")
print(Default.columns)
```

```
## Index(['default', 'student', 'balance', 'income'], dtype='object')
```

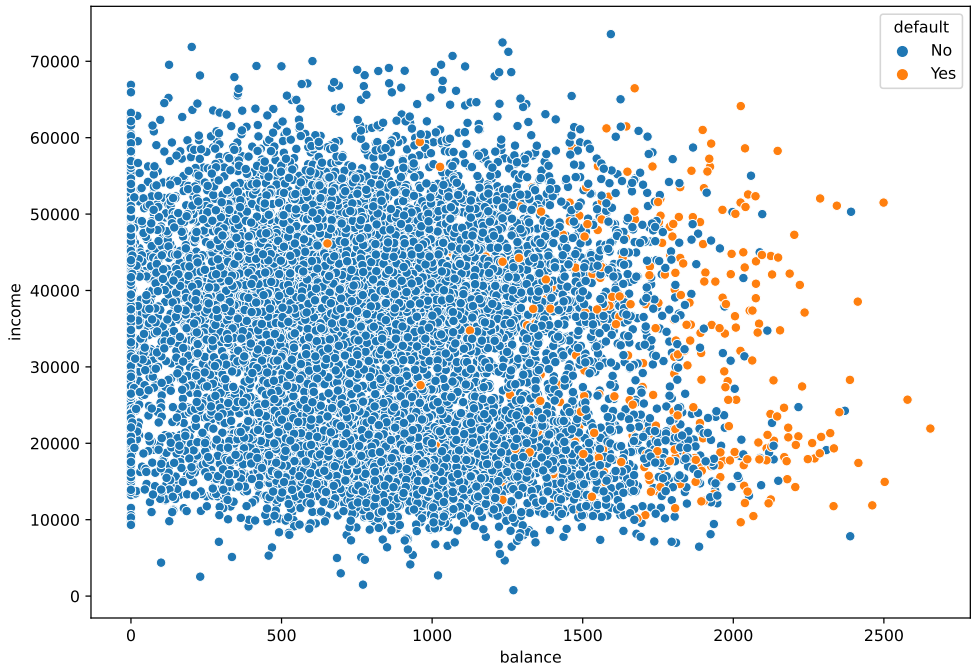
```
print(Default.head())
```

```
##   default student      balance        income
## 0      No      No   729.526495  44361.62507
## 1      No     Yes   817.180407  12106.13470
## 2      No      No  1073.549164  31767.13895
## 3      No      No   529.250605  35704.49394
## 4      No      No   785.655883  38463.49588
```

Logistic Regression

- Example: Credit Card Default

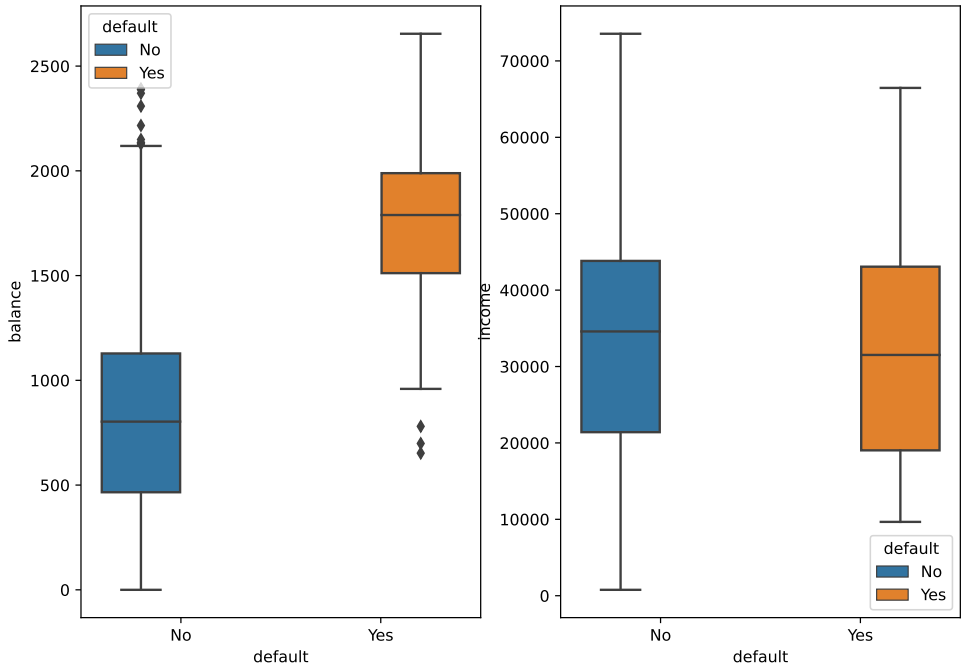
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.scatterplot(data=Default, x='balance', y='income',hue = 'default')
plt.show()
```



Logistic Regression

- Example: Credit Card Default

```
fig, ax = plt.subplots(1, 2)
sns.boxplot(x='default', y='balance', hue='default',
data=Default, ax=ax[0])
sns.boxplot(x='default', y='income', hue='default',
data=Default, ax=ax[1])
plt.show()
```



Logistic Regression

- Can we use Linear Regression?

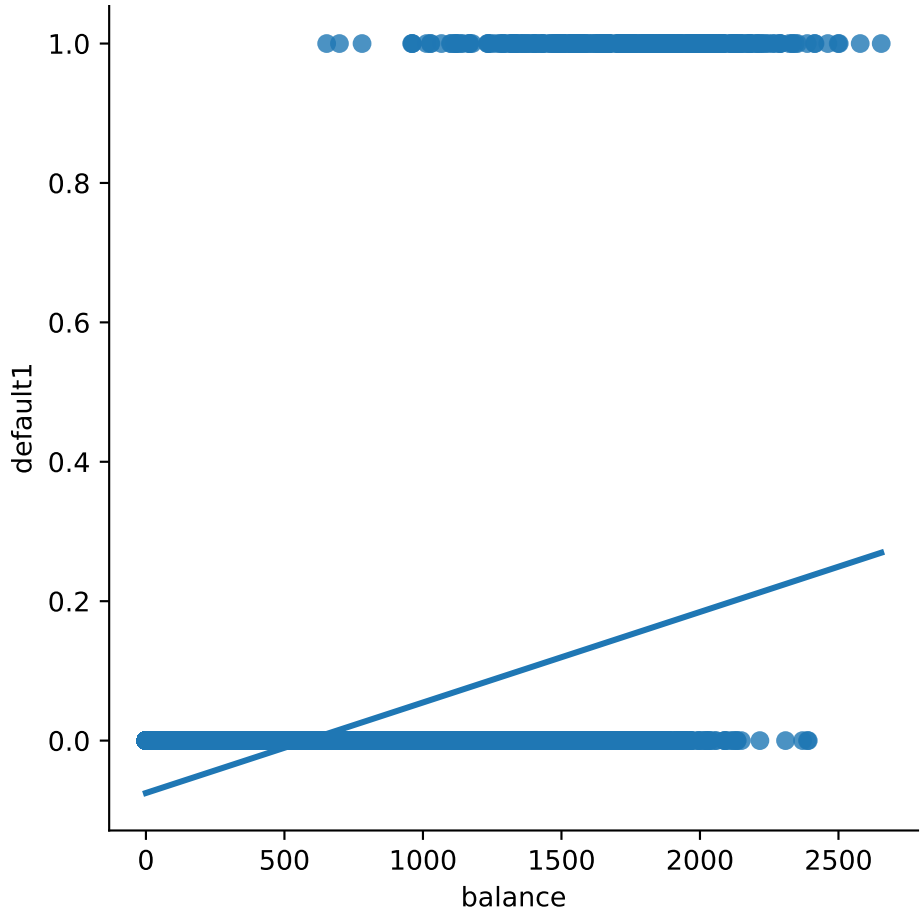
Suppose $Y = \begin{cases} 0 & \text{if No} \\ 1 & \text{if Yes} \end{cases}$, Can we simply perform a linear regression of Y on X and classify as Yes if $\hat{Y} > 0.5$?

```
Default['default1'] = [0 if x =="No"
else 1 for x in Default['default']]
```

Logistic Regression

- Can we use Linear Regression?

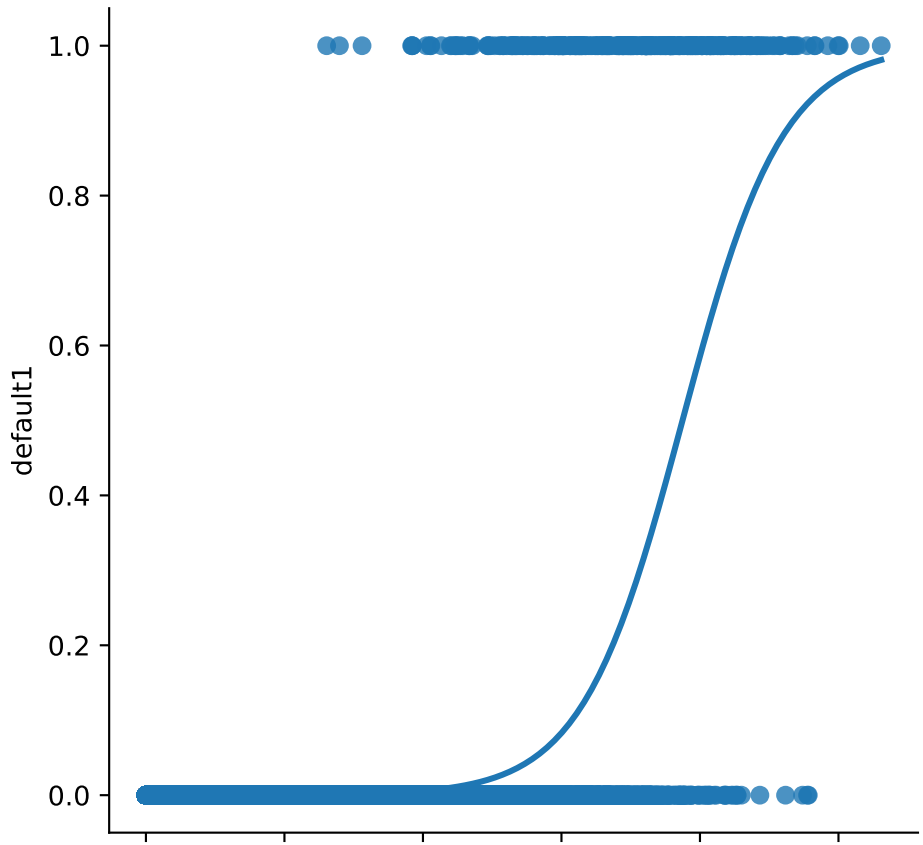
```
sns.lmplot(data=Default, x='balance', y='default1',
 ci=None, fit_reg=True)
```



Logistic Regression

- In this case of a binary outcome, linear regression does a good job as a classifier, and is equivalent to linear discriminant analysis which we discuss later.
- But linear regression might produce probabilities less than zero or bigger than one. So it can not give a good estimate of $E(Y|X = x) = Pr(Y = 1|X = x)$. Logistic regression is more appropriate.

```
sns.lmplot(data=Default, x='balance', y='default1',ci=None,
logistic=True, fit_reg=True)
```



Logistic Regression

- Logistic regression is the straightforward extension of linear regression to the classification setting when $y \in \{0, 1\}$: a two-class classification problem.
- Let $p(X) = Pr(Y = 1|X)$
  - For example, we want to use biomarker level to predict probability of cancer.
- Recall that logistic regression uses the form

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

  - $p(X)$ will lie between 0 and 1.
- Furthermore,

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

  - This function of $p(X)$ is called the logit or log odds (by log we mean natural log : ln).

Logistic Regression

- We use maximum likelihood to estimate the parameters
- Most statistical packages can fit linear logistic regression models by maximum likelihood.

```
import statsmodels.formula.api as smf
fit1=smf.logit("default1~balance", data=Default).fit()
```

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

```
print(fit1.summary())
```

```
##                            Logit Regression Results
## ==============================================================================
## Dep. Variable:                   default1   No. Observations:                10000
## Model:                             Logit   Df Residuals:                     9998
## Method:                              MLE   Df Model:                            1
## Date:                   Sun, 09 Apr 2023   Pseudo R-squ.:                  0.4534
## Time:                           19:50:28   Log-Likelihood:                -798.23
## converged:                         True   LL-Null:                       -1460.3
## Covariance Type:             nonrobust   LLR p-value:                6.233e-290
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept    -10.6513      0.361    -29.491      0.000     -11.359      -9.943
## balance        0.0055      0.000     24.952      0.000       0.005       0.006
```

Logistic Regression

- What is our estimated probability of default for someone with a balance of $1000?
- With a balance of $2000?
- What value of Balance will give a predicted Default rate of 50%?

Logistic Regression

```
new_data = pd.DataFrame({'balance': [1000, 2000]})
predicted_probs=fit1.predict(new_data)
predicted_probs
```

```
## 0    0.005752
## 1    0.585769
## dtype: float64
```

```
#log(p/(1-p)) = beta_0+beta_1x
#fit1.params
beta0=fit1.params[0]
beta1=fit1.params[1]

print(-beta0/beta1)
```

```
## 1936.9870007497384
```

Logistic Regression

- To avoid using dataframe in predictions, we fit logistic regression using sklearn

```
import numpy as np
from sklearn.linear_model import LogisticRegression
log_regression = LogisticRegression()
fit=log_regression.fit(Default['balance'].values.reshape(-1, 1),
Default['default1'])
#fit.intercept_
#fit.coef_

new_values = np.array([1000, 2000]).reshape(-1, 1)
fit.predict_proba(new_values)
```

```
## array([[0.99424785, 0.00575215],
##        [0.41423073, 0.58576927]])
```

```
fit.predict(new_values)
```

```
## array([0, 1], dtype=int64)
```

Logistic Regression

- Lets do it again, using student as the predictor

```
fit2=smf.logit("default1~student", data=Default).fit()
```

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

```
print(fit2.summary())
```

```
##                            Logit Regression Results
## ==============================================================================
## Dep. Variable:                default1   No. Observations:                10000
## Model:                          Logit   Df Residuals:                     9998
## Method:                           MLE   Df Model:                            1
## Date:                Sun, 09 Apr 2023   Pseudo R-squ.:                0.004097
## Time:                        19:50:38   Log-Likelihood:                -1454.3
## converged:                       True   LL-Null:                       -1460.3
## Covariance Type:            nonrobust   LLR p-value:                 0.0005416
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975
## ------------------------------------------------------------------------------
## Intercept      -3.5041      0.071    -49.554      0.000      -3.643      -3.36
## student[T.Yes]  0.4049      0.115      3.520      0.000       0.179       0.63
## ==============================================================================
```

Logistic Regression with Several Variables

- Suppose that there are p features: $X_1, \ldots, X_p$.
- Just like before

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

- And just like before

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p.$$

Logistic Regression with Several Variables

```
fit3=smf.logit("default1~balance+income+student", data=Default).fit()
```

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

```
print(fit3.summary())
```

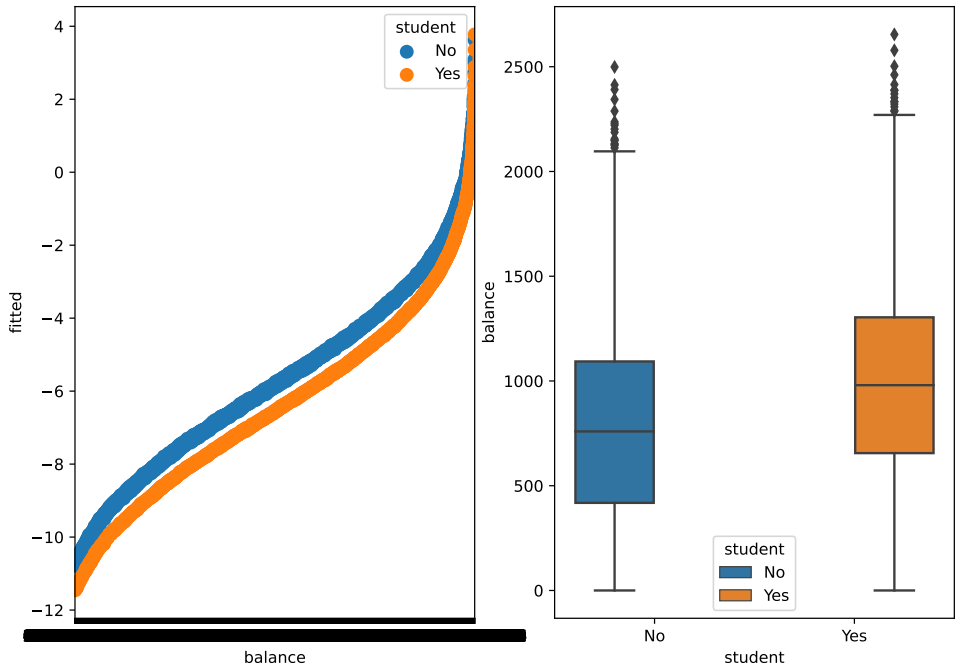
```
##                            Logit Regression Results
## ==============================================================================
## Dep. Variable:                default1   No. Observations:                10000
## Model:                           Logit   Df Residuals:                     9996
## Method:                            MLE   Df Model:                            3
## Date:                 Sun, 09 Apr 2023   Pseudo R-squ.:                  0.4619
## Time:                         19:50:42   Log-Likelihood:                -785.77
## converged:                       True   LL-Null:                       -1460.3
## Covariance Type:           nonrobust   LLR p-value:                3.257e-292
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975
## ------------------------------------------------------------------------------
## Intercept     -10.8690      0.492    -22.079      0.000     -11.834      -9.90
## student[T.Yes]  -0.6468      0.236     -2.738      0.006      -1.110      -0.18
## balance          0.0057      0.000     24.737      0.000       0.005       0.00
## income        3.033e-06    8.2e-06      0.370      0.712    -1.3e-05    1.91e-0
## ==============================================================================
##
```

Logistic Regression with Several Variables

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

Logistic Regression with Several Variables

```
Default['fitted']=fit3.fittedvalues
fig, ax = plt.subplots(1, 2)
sns.pointplot(x='balance', y='fitted', hue='student',
data=Default, ax=ax[0])
sns.boxplot(x='student', y='balance', hue='student',
data=Default, ax=ax[1])
plt.show()
```



License