## Classification with scikit-learn

- Here we will look into the problem of classification, a situation in which a response is a **categorical variable**.
- We will introduces a number of classification techniques, and convey their corresponding strengths and weaknesses by visually inspecting the decision boundaries for each model.
    1. Logistic Regression
    2. Linear Discriminant Analysis
    3. Knearest Neighbour
- Here we will use **scikit-learn**, an easy-to-use, general-purpose toolbox for machine learning in Python

## Scikit-learn

- Scikit-learn is a library that provides a variety of both supervised and unsupervised machine learning techniques.
- **Supervised machine learning** refers to the problem of inferring a function from labeled training data, and it comprises both regression and classification.
- **Unsupervised machine learning**, on the other hand, refers to the problem of finding interesting patterns or structure in the data; it comprises techniques such as clustering and dimensionality reduction.
- In addition to statistical learning techniques, scikit-learn provides utilities for common tasks such as model selection, feature extraction, and feature selection.

## Estimators

- Scikit-learn provides an object-oriented interface centered around the concept of an **Estimator**.
- According to the scikit-learn tutorial :
  "*An estimator is any object that learns from data; it may be a classification, regression or clustering algorithm or a transformer that extracts/filters useful features from raw data.*"
- Usually, the data is comprised of a two-dimensional numpy array `X` of shape `(n_samples, n_predictors)` , (*in other words number of rows and columns*) that holds the so-called **feature matrix** and a one-dimensional numpy array `y` that holds the responses.
- The `Estimator.fit` method sets the state of the estimator based on the training data.
- Some estimators allow the user to control the fitting behavior.

- For example, the `sklearn.linear_model.LinearRegression` estimator allows the user to specify whether or not to fit an intercept term.
- This is done by setting the corresponding constructor arguments of the estimator object:

```
In [3]: from sklearn.linear_model import LinearRegression
        est = LinearRegression(fit_intercept=False)
```

```
from sklearn.linear_model import LinearRegression
est = LinearRegression(fit_intercept=False)
```

- During the fitting process, the state of the estimator is stored in instance attributes that have a trailing underscore ('_').
- For example, the coefficients of a `LinearRegression` estimator are stored in the attribute `coef_`:

```
## Using Previous Code to define "est"
import numpy as np

# random training data
X = np.random.rand(10, 2)
y = np.random.randint(2, size=10)
est.fit(X, y)
est.coef_   # access coefficients

# Output : array([ 0.33176871,  0.34910639])
```

## Making Prediction with Estimators

- Estimators that can generate predictions provide a `Estimator.predict` method.
- In the case of regression, Estimator.predict will return the predicted regression values; it will return the corresponding class labels in the case of classification.
- Classifiers that can predict the probability of class membership have a method `Estimator.predict_probability` that returns a two-dimensional numpy array of shape `(n_samples, n_classes)` where the classes are lexicographically ordered.

## Understanding Classification

Although regression and classification appear to be very different they are in fact similar problems.

- In regression our predictions for the response are real-valued numbers
- on the other hand, in classification the response is a mutually exclusive class label
- Example "*Is the email spam?*" or "*Is the credit card transaction fraudulent?*".

## Binary Classsification Problems

- If the number of classes is equal to two, then we call it a binary classification problem; if there are more than two classes, then we call it a multiclass classification problem.
- In the following we will assume binary classification because it's the more general case, and — we can always represent a multiclass problem as a sequence of binary classification problems.

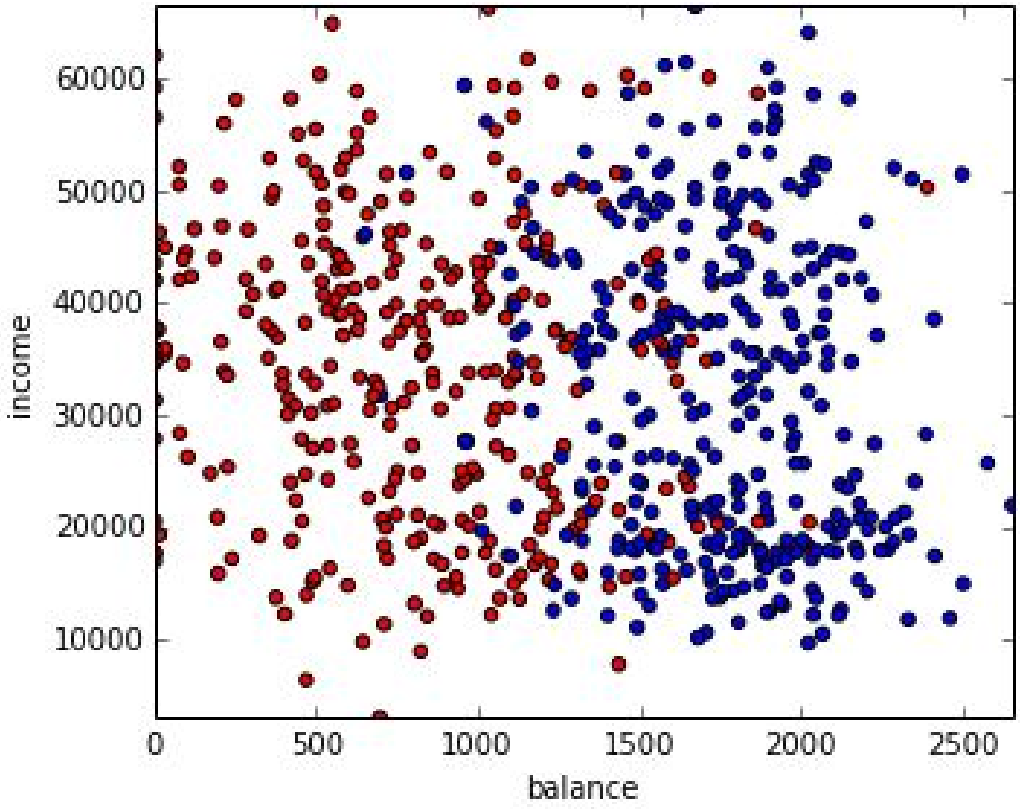
## Credit Card Fraud

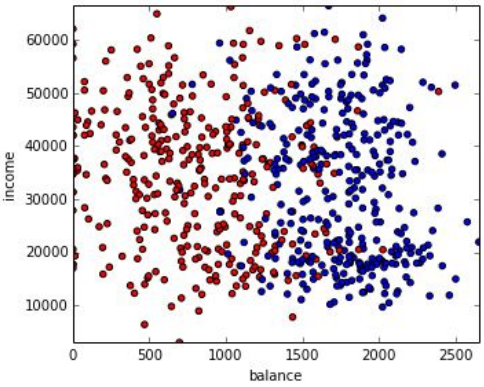
- We can also think of classification as a function estimation problem where the function that we want to estimate separates the two classes.
- This is illustrated in the example below where our goal is to predict whether or not a credit card transaction is fraudulent
- *The dataset is provided by James et al.,* ***Introduction to Statistical Learning****.*

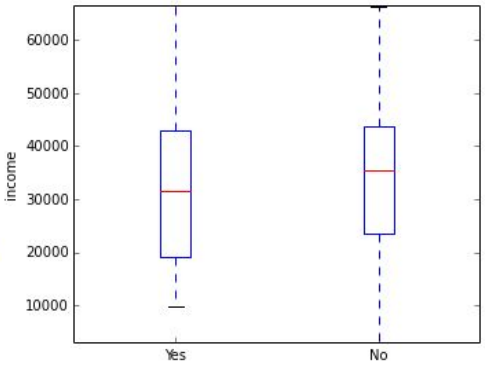
```
: import pandas as pd

df = pd.read_csv('https://d1pqsl2386xqi9.cloudfront.net/notebooks/Default.csv', index_col=0)

# downsample negative cases -- there are many more negatives than positives
indices = np.where(df.default == 'No')[0]
rng = np.random.RandomState(13)
rng.shuffle(indices)
n_pos = (df.default == 'Yes').sum()
df = df.drop(df.index[indices[n_pos:]])

df.head()
```

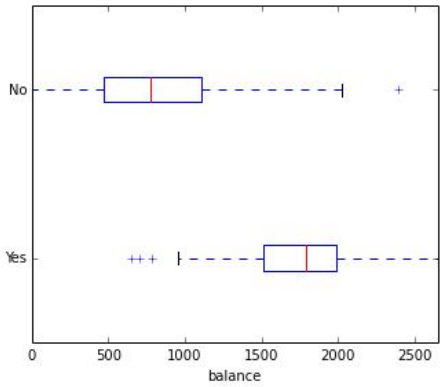
| | default | student | balance | income |
|---|---|---|---|---|
| **20** | No | No | 1095.072735 | 26464.631389 |
| **38** | No | No | 351.453472 | 35087.488648 |
| **61** | No | No | 766.234379 | 46478.294257 |
| **78** | No | No | 728.373251 | 45131.718265 |
| **79** | No | No | 76.991291 | 28392.093412 |

- On the left you can see a scatter plot where fraudulent cases are red dots and non-fraudulent cases are blue dots.
- A good separation seems to be a vertical line at around a balance of 1400 as indicated by the boxplots on the next slide.




60000
50000
40000
30000
20000
10000
income
0
500
1000
1500
2000
2500
balance

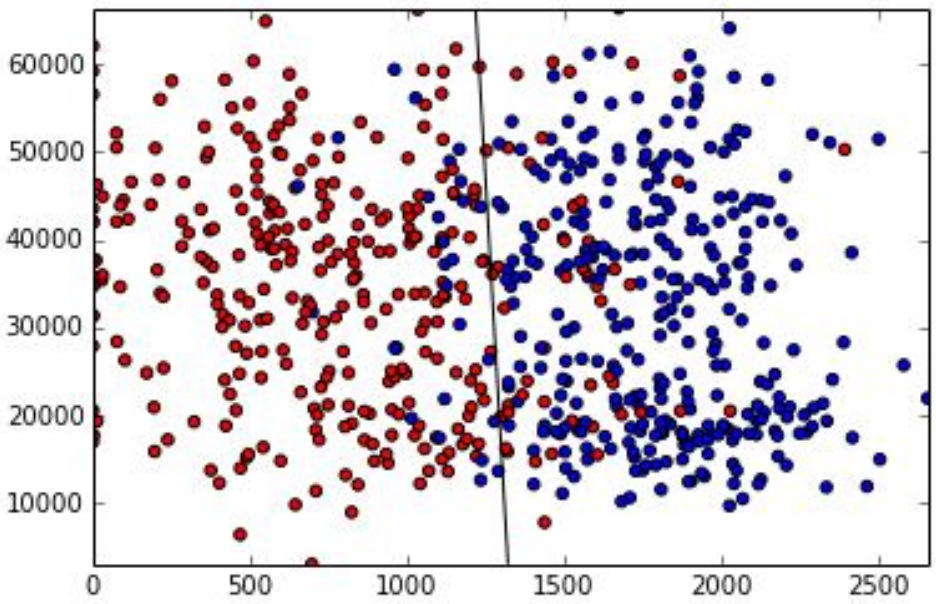

60000
50000
40000
30000
20000
10000
income
Yes
No


No
Yes
0
500
1000
1500
2000
2500
balance

## Simple Approach - Linear Regression

- A simple (or perhaps simplistic) approach to binary classification is to simply encode default as a numeric variable with `'Yes' == 1` and `'No' == -1`.
- Then we fit an Ordinary Least Squares regression model and use this model to predict the response as 'Yes' if the regressed value is higher than 0.0 and 'No' otherwise.
- The points for which the regression model predicts 0.0 lie on the so-called decision surface — since we are using a linear regression model, the decision surface is linear as well.

```
from sklearn.linear_model import LinearRegression

# get feature/predictor matrix as numpy array
X = df[['balance', 'income']].values

# encode class labels
classes, y = np.unique(df.default.values, return_inverse=True)
y = (y * 2) - 1  # map {0, 1} to {-1, 1}

# fit OLS regression
est = LinearRegression(fit_intercept=True, normalize=True)
est.fit(X, y)
```




- Points that lie on the left side of the decision boundary will be classified as negative;
- Points that lie on the right side, positive.

## Confusion Matrix

- We can assess the performance of the model by looking at the confusion matrix — a cross tabulation of the actual and the predicted class labels.
- The correct classifications are shown in the diagonal of the confusion matrix. The off-diagonal terms show you the **classification errors**.
- A condensed summary of the model performance is given by the **misclassification rate** determined simply by dividing the number of errors by the total number of cases.

```
from sklearn.metrics import confusion_matrix as sk_confusion_matrix

# the larger operator will return a boolean array which we will cast as integers
y_pred = (2 * (est.predict(X) > 0.0)) - 1

def confusion_matrix(y_test, y_pred):
    cm = sk_confusion_matrix(y, y_pred)
    cm = pd.DataFrame(data=cm, columns=[-1, 1], index=[-1, 1])
    cm.columns.name = 'Predicted label'
    cm.index.name = 'True label'
    error_rate = (y_pred != y).mean()
    print('error rate: %.2f' % error_rate)
    return cm

confusion_matrix(y, y_pred)
```

| Predicted label | -1 | 1 |
|---|---|---|
| True label | | |
| -1 | 282 | 51 |
| 1 | 29 | 304 |

## Cross Validation

- In this example we are assessing the model performance on the same data that we used to fit the model.
- This might be a biased estimate of the models performance, for a classifier that simply memorizes the training data has zero training error but would be totally useless to make predictions.
- It is much better to assess the model performance on a separate dataset called the test data.
- Scikit-learn provides a number of ways to compute such held-out estimates of the model performance.
- One way is to simply split the data into a **training set** and **testing set**.

```
from sklearn.cross_validation import train_test_split

# create 80%-20% train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# fit on training data
est = LinearRegression().fit(X_train, y_train)

# test on data that was not used for fitting
y_pred = (2 * (est.predict(X) > 0.0)) - 1

confusion_matrix(y_test, y_pred)
```

| Predicted label | -1 | 1 |
|---|---|---|
| True label | | |
| -1 | 287 | 46 |
| 1 | 29 | 304 |

## Classification Techniques

- Different classification techniques can often be compared using the type of decision surface they can learn.
- The decision surfaces describe for what values of the predictors the model changes its predictions and it can take several different shapes: piece-wise constant, linear, quadratic, vornoi tessellation, ...

This next part will introduce three popular classification techniques:

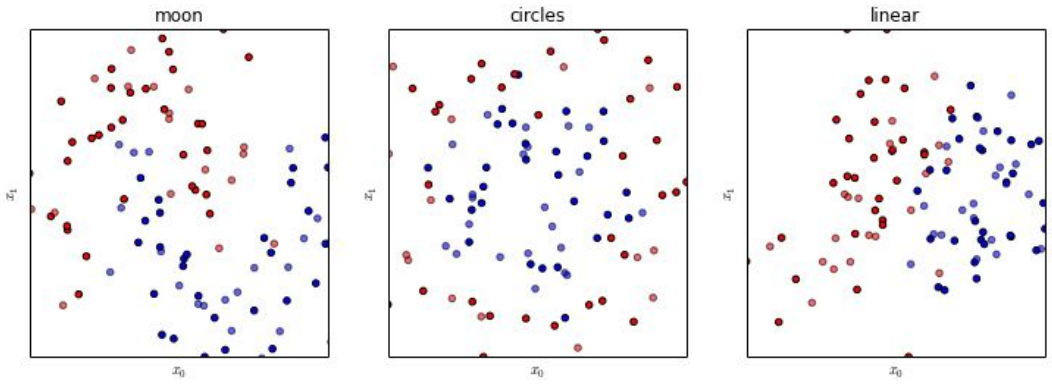
1 Logistic Regression,

2 Discriminant Analysis,

3 Nearest Neighbor.

We will investigate what their strengths and weaknesses are by looking at the decision boundaries they can model. In the following we will use three synthetic datasets that we adopted from this scikit-learn example.

## Synthetic Data Sets

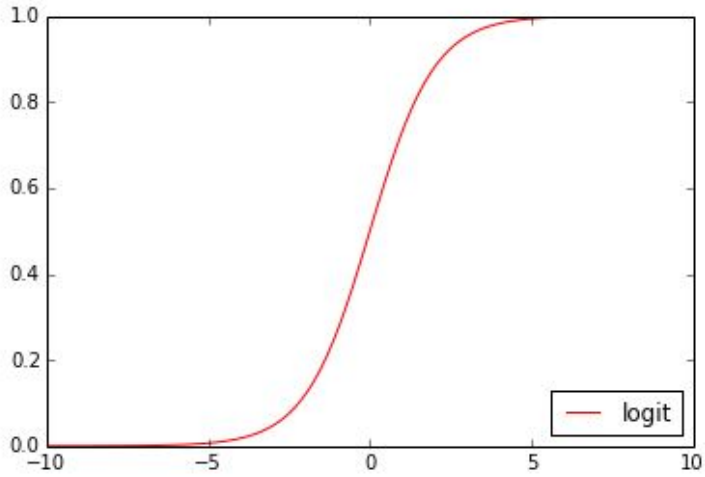



- The task in each of the above examples is to separate the red from the blue points.
- Testing data points are plotted in lighter color.
- The left example contains two intertwined moon sickles; the middle example is a circle of blues framed by a ring of reds; and the right example shows two linearly separable gaussian blobs.

## Method 1: Logistic Regression

- Logistic regression can be viewed as an extension of linear regression to classification problems.
- One of the limitations of linear regression is that it cannot provide class probability estimates.
- This is often useful, for example, when we want to inspect manually the most fraudulent cases.
- Basically, we would like to constrain the predictions of the model to the range $[0, 1]$ so that we can interpret them as probability estimates.
- In Logistic Regression, we use the logit function to clamp predictions from the range $[-\infty, \infty]$ to $[0, 1]$.

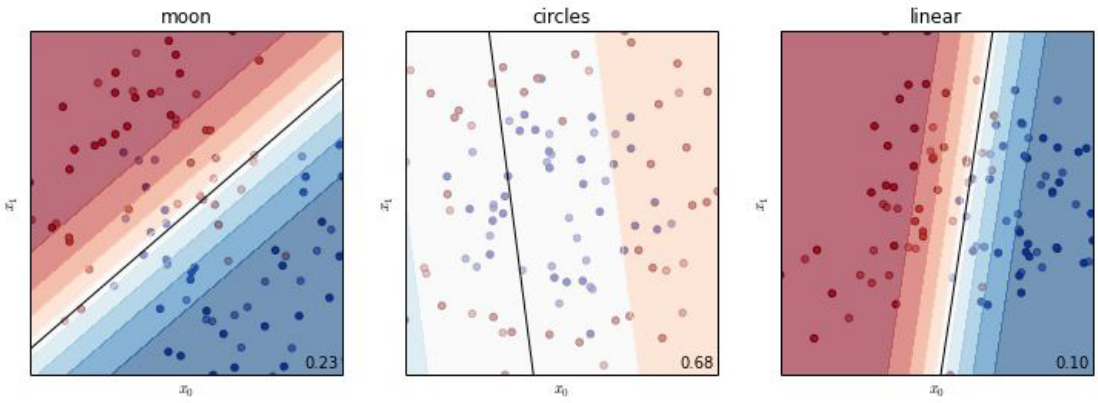
## Logistic Transformation




- Logistic regression is available in scikit-learn via the class `sklearn.linear_model.LogisticRegression`.
- Lets see how Logistic Regression does on our three toy datasets.

```
from sklearn.linear_model import LogisticRegression

est = LogisticRegression()
plot_datasets(est)
```

## Model Appraisal




- As we can see, a linear decision boundary is not a poor approximation for the moon datasets, although we fail to separate the two tips of the sickles in the center.
- The **circles** dataset, on the other hand, is not well suited for a linear decision boundary.
- The error rate of 0.68 is in fact worse than random guessing.
- For the linear dataset we picked in fact the correct model class — the error rate of 10% is due to the noise component in our data.
- The gradient shows you the probability of class membership — white shows you that the model is very uncertain about its prediction.

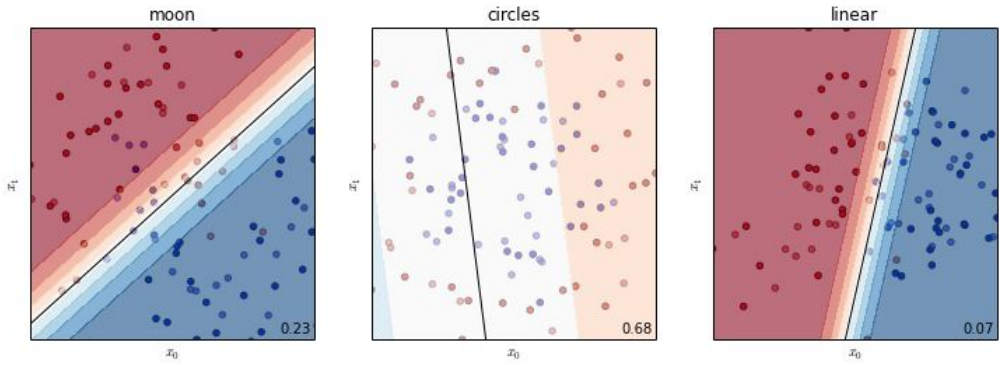
## Method 2: Linear Discriminant Analysis

- Linear discriminant Analysis (LDA) is another popular technique which shares some similarities with Logistic Regression.
- LDA too finds linear boundary between the two classes where points on side are classified as one class and those on the other as classified as the other class.

```
from sklearn.lda import LDA

est = LDA()
plot_datasets(est)
```

## Linear Discriminant Analysis : Model Appraisal

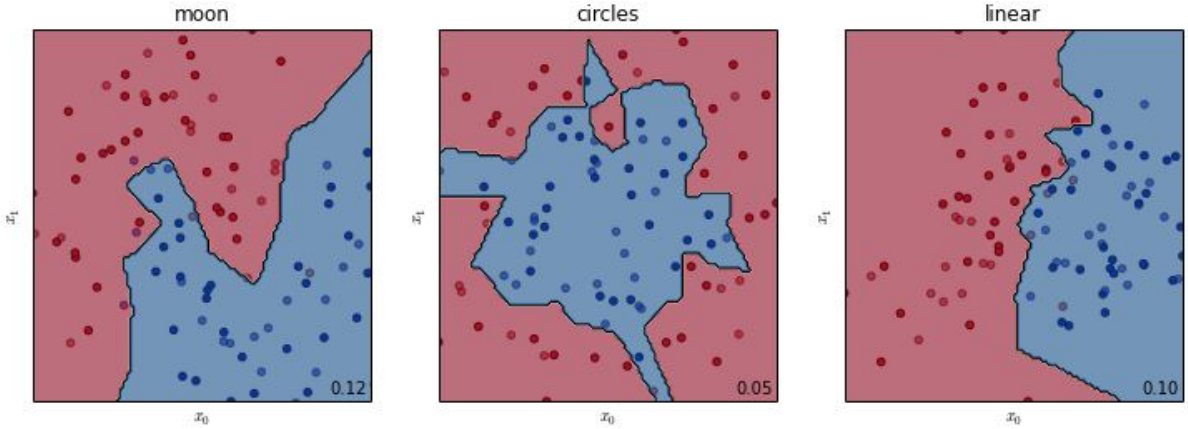
(Remark - almost same as logistic regression)




- The major difference between LDA and Logistic Regression is the way both techniques picks the linear decision boundary.
- Linear Discriminant Analysis models the decision boundary by making distributional assumptions about the data generating process
- Logistic Regression models the probability of a sample being member of a class given its feature values.

## Method 3: Nearest Neighbor

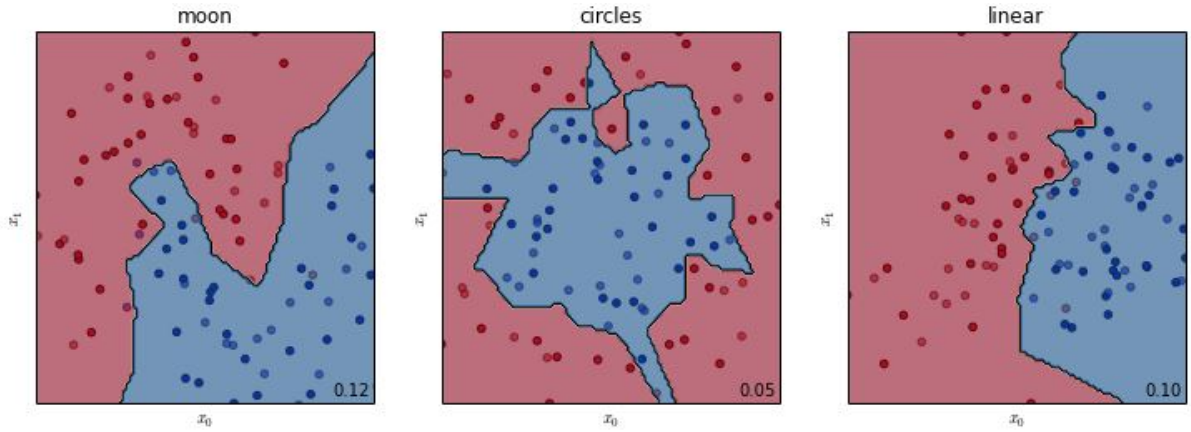
- Nearest Neighbor uses the notion of similarity to assign class labels; it is based on the smoothness assumption that points which are nearby in input space should have similar outputs.
- It does this by specifying a similarity (or distance) metric, and at prediction time it simply searches for the k most similar among the training examples to a given test example.
- The prediction is then either a majority vote of those k training examples or a vote weighted by similarity.
- The parameter k specifies the smoothness of the decision surface.
- The decision surface of a k-nearest neighbor classifier can be illustrated by the **Voronoi tesselation** of the training data, that show you the regions of constant respones.




- Yet Nearest Neighbor differs fundamentally from the above models in that it is a so-called non-parametric technique: the number of parameters of the model can grow infinitely as the size of the training data grows.
- Furthermore, it can model non-linear decision boundaries, something that is important for the first two datasets: moons and circles.

## Method 3: Nearest Neighbor

```
from sklearn.neighbors import KNeighborsClassifier

est = KNeighborsClassifier(n_neighbors=1)
plot_datasets(est)
```



## Adjusting Smoothness Parameter

- If we increase k we enforce the smoothness assumption.
- This can be seen by comparing the decision boundaries in the plots below where k=5 to those above where k=1.

```
est = KNeighborsClassifier(n_neighbors=5)
plot_datasets(est)
```