Introduction to scikit-learn

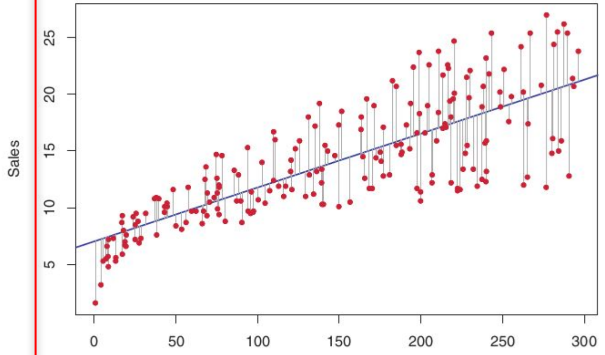
Konstantinos Bougiatiotis - bogas.ko@gmail.com
A. Kolovou - akolovou@di.uoa.gr

Machine Learning approaches

In general, a learning problem considers a **set of n samples** of data and tries to **predict properties** of the same data (or new). Each sample usually has several **attributes or features**. Learning problems fall into a few categories:

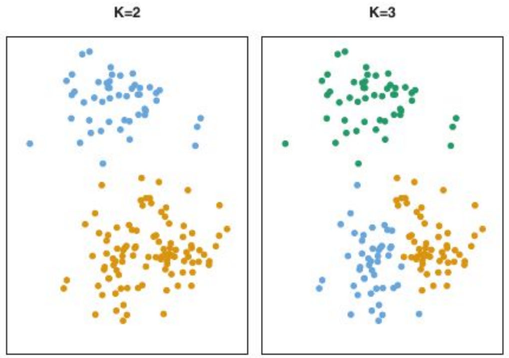
- **Supervised Learning**

Learn a function ***f*** that maps input features to output, given some training data.



- **Unsupervised Learning**

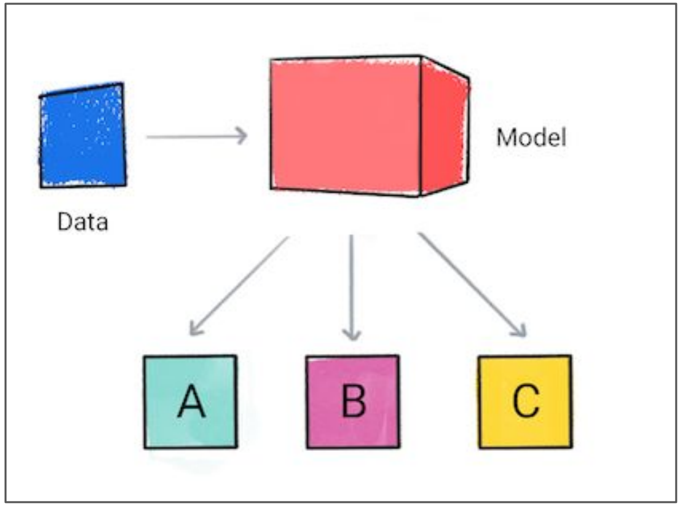
No output variable! The goal is to understand the relationship between features. e.g. clustering

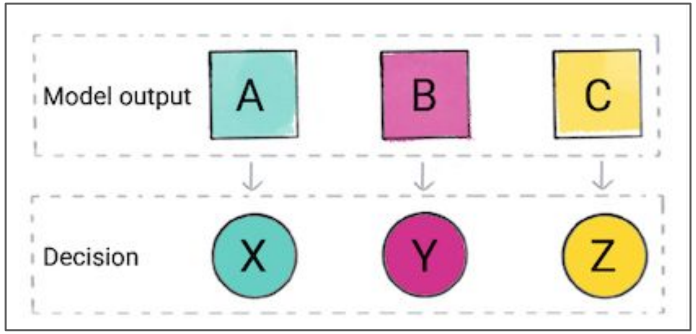


- **Semi-supervised learning**
- **Reinforcement learning**
- **...**

Choose the right kind of model

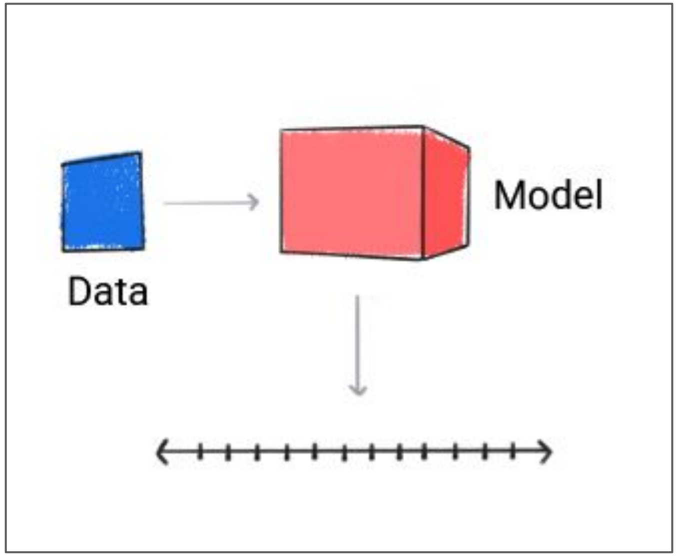
A **classification model** predicts what category the input data belongs to, for example, whether an input should be classified as A, B, or C.

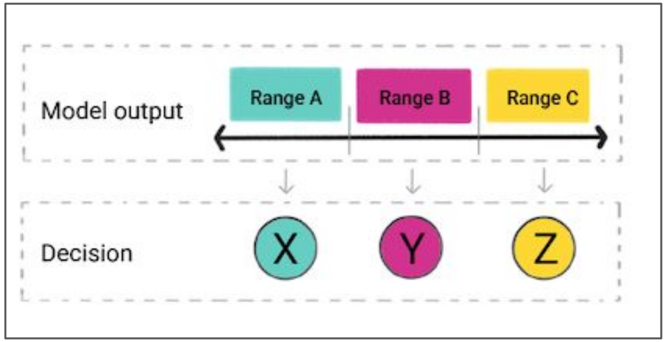




Choose the right kind of model

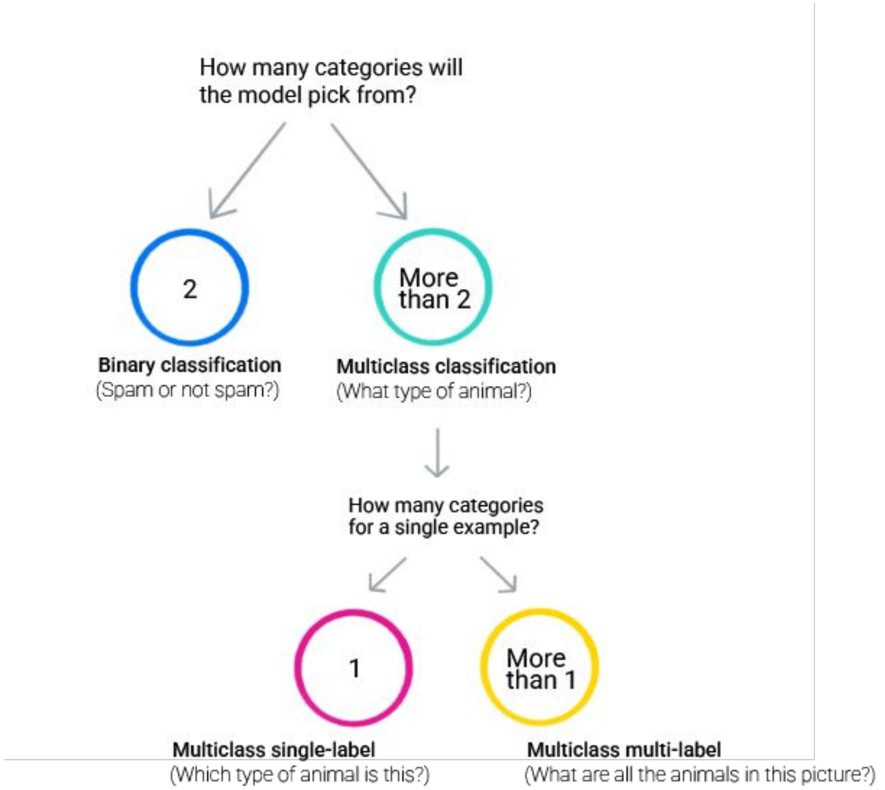
A **regression model** predicts where to place the input data on a number line.





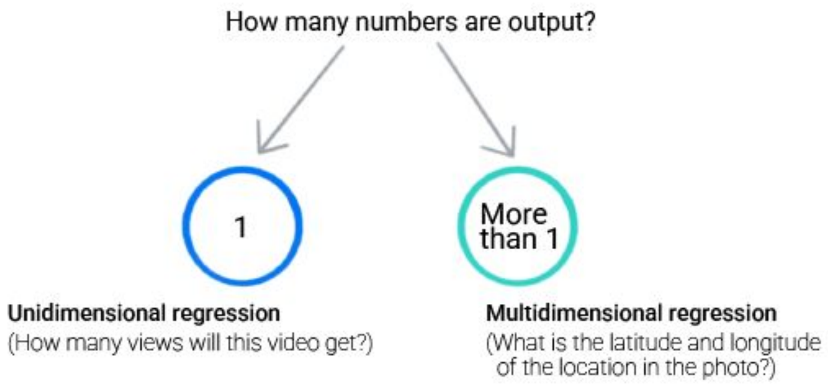
Classification flowchart

The model's output should accomplish the task defined in the ideal outcome. If you're using a classification model, the categorical prediction should provide the data needed to accomplish the ideal outcome.


How many categories will the model pick from?
2
Binary classification
(Spam or not spam?)
More than 2
Multiclass classification
(What type of animal?)
How many categories for a single example?
1
Multiclass single-label
(Which type of animal is this?)
More than 1
Multiclass multi-label
(What are all the animals in this picture?)

Regression flowchart

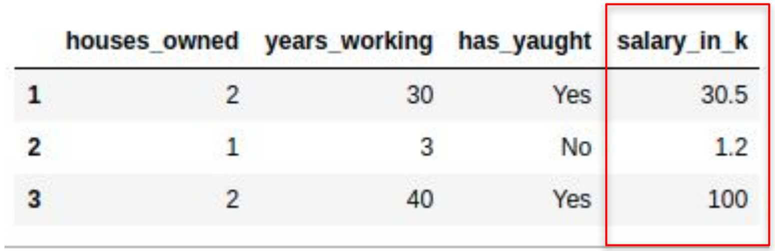
If you're using a regression model, the numeric prediction should provide the data needed to accomplish the ideal outcome.


How many numbers are output?
1
More than 1
Unidimensional regression
(How many views will this video get?)
Multidimensional regression
(What is the latitude and longitude of the location in the photo?)

Reminder: Regression VS Classification

In Regression, the output we want to predict is **continuous**.

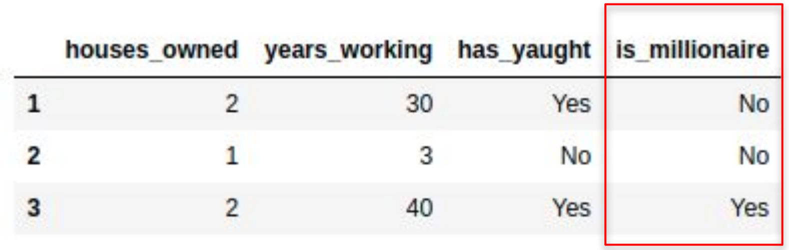
target



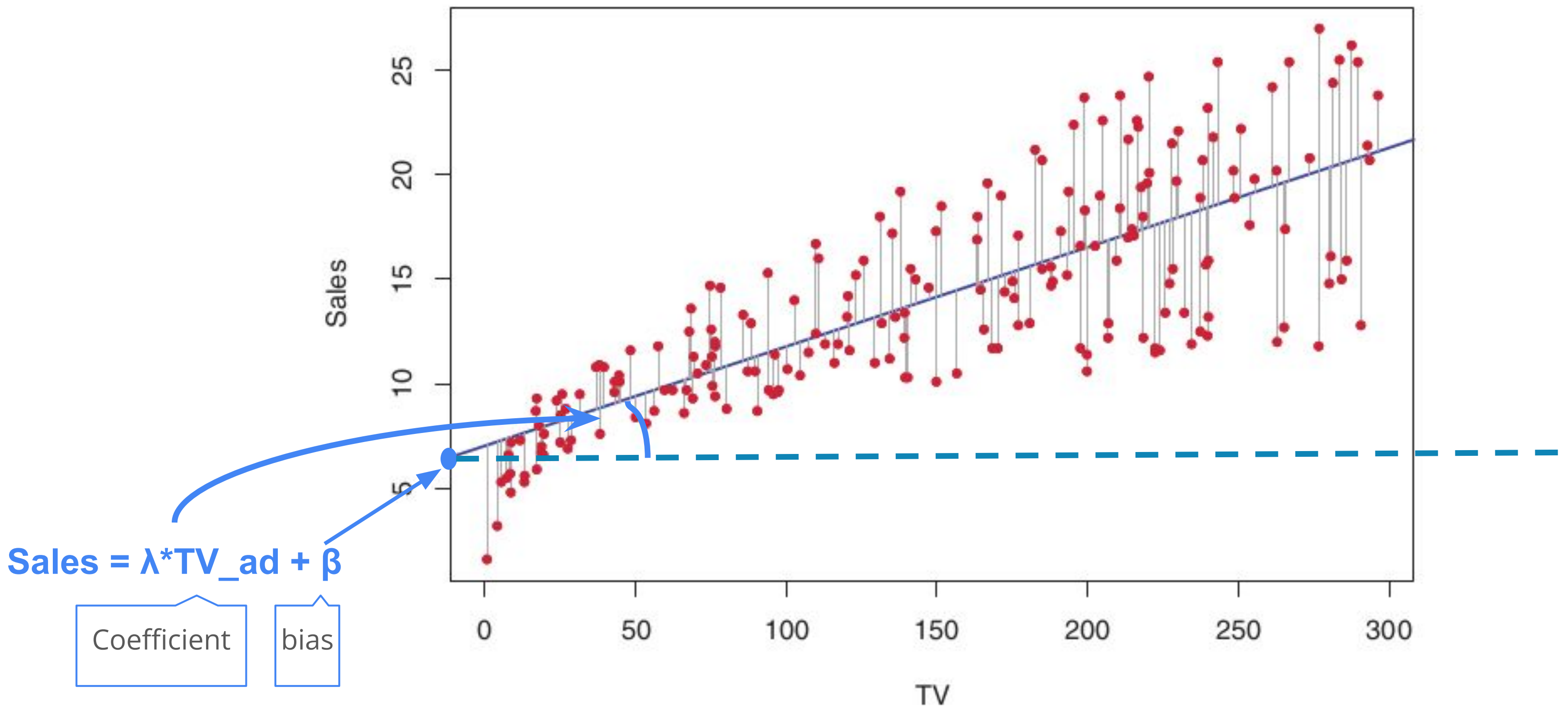
	houses_owned	years_working	has_yaught	salary_in_k
1	2	30	Yes	30.5
2	1	3	No	1.2
3	2	40	Yes	100

In Classification, the output we want to predict is **discrete**.

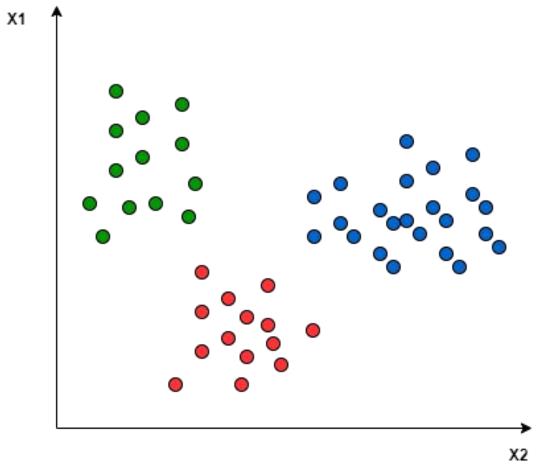
target



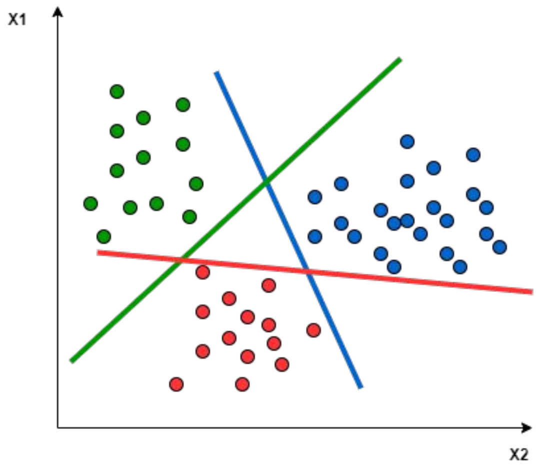
	houses_owned	years_working	has_yaught	is_millionaire
1	2	30	Yes	No
2	1	3	No	No
3	2	40	Yes	Yes



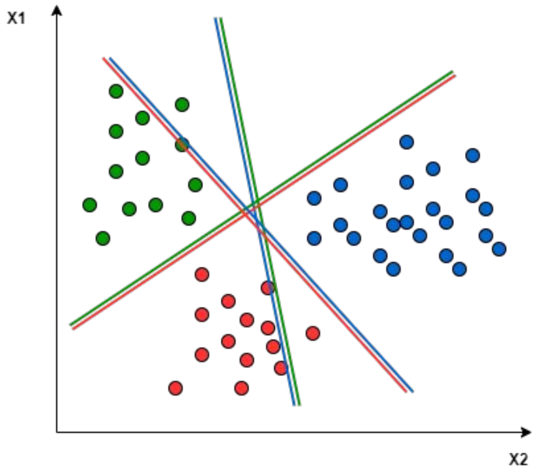
A simple linear regression model (Supervised Learning)



Unclassified data



One-to-Rest approach

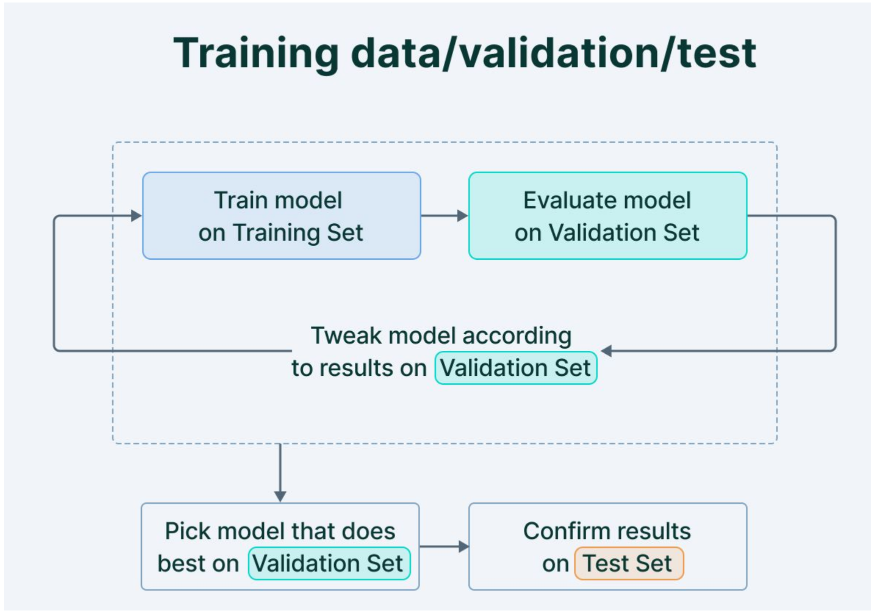


One-to-One approach

Supervised Learning - Classification (Multi-class case)

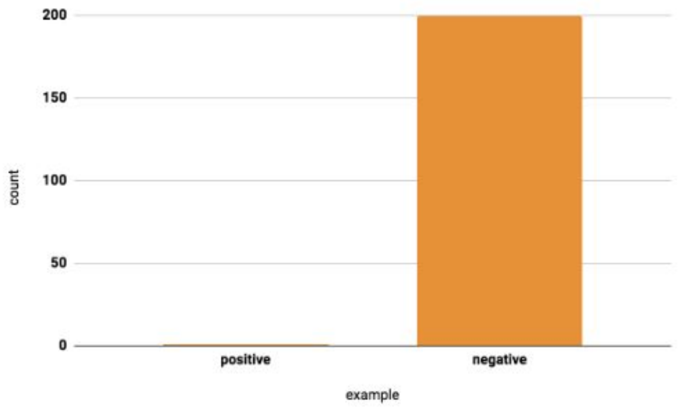
Data Split

After collecting your data and sampling where needed, the next step is to split your data into **training sets**, **validation sets**, and **testing sets**.

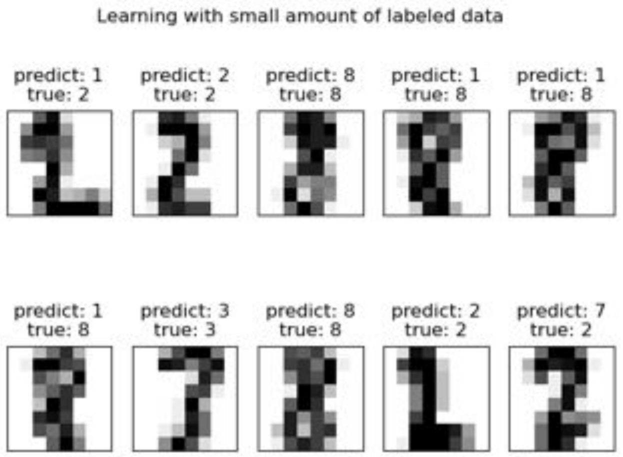

Training data/validation/test
Train model on Training Set
Evaluate model on Validation Set
Tweak model according to results on Validation Set
Pick model that does best on Validation Set
Confirm results on Test Set

Imbalanced Data

A classification data set with skewed class proportions is called imbalanced. Classes that make up a large proportion of the data set are called majority classes. Those that make up a smaller proportion are minority classes.



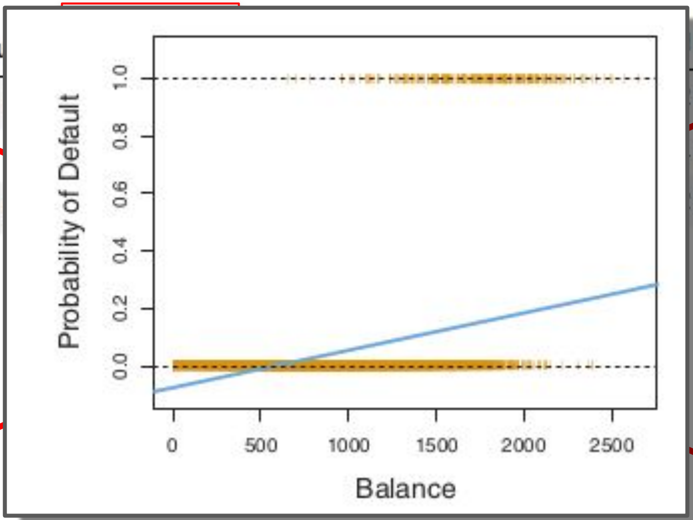
Classification example: sklearn digits dataset



Linear Regression for Classification?

target

	houses_owned	years_working	has_yau
1	2	30	
2	1	3	
3	2	40	



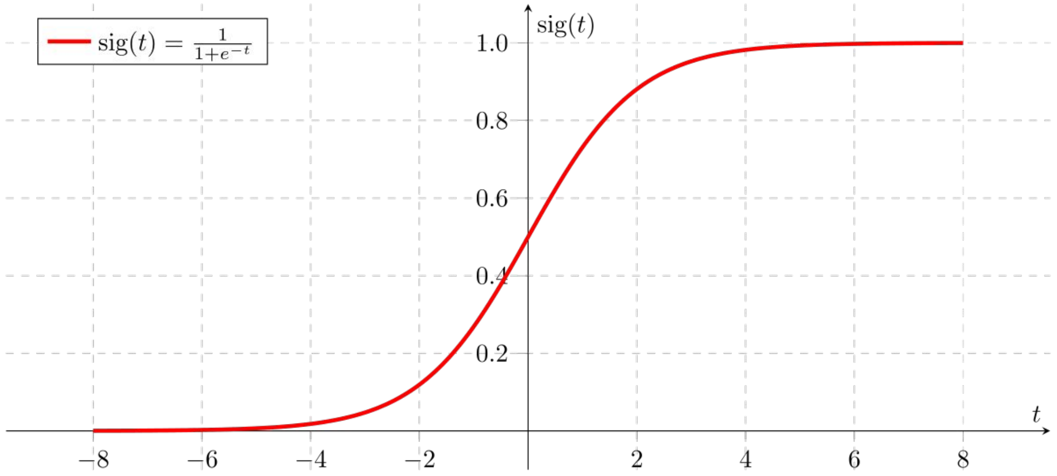
target

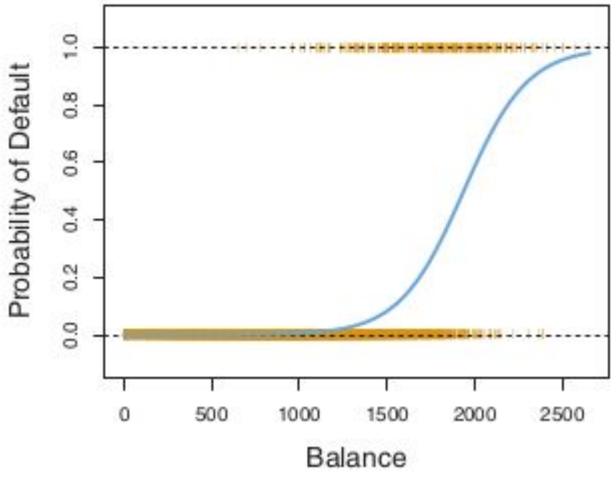
years_working	has_yaught	is_millionaire
30	Yes	1
3	No	0
40	Yes	1

Linear Regression

PROFIT!?

Solution: Logistic Regression





Loading an example dataset

scikit-learn comes with a few standard datasets, for instance the iris and digits datasets for classification and the diabetes dataset for regression.

```
from sklearn import datasets

iris = datasets.load_iris()

digits = datasets.load_digits()
```

A dataset is a dictionary-like object that holds all the data and some metadata about the data. This data is stored in the ***.data*** member, which is a ***n_samples***, ***n_features*** array. In the case of supervised problem, one or more response variables are stored in the ***.target*** member.

Loading an example dataset

For instance, in the case of the digits dataset, digits.data gives access to the features that can be used to classify the digits samples:

```
print(digits.data)
```

```
[[ 0.   0.   5. ...   0.   0.   0.]
 [ 0.   0.   0. ...  10.   0.   0.]
 [ 0.   0.   0. ...  16.   9.   0.]
 ...
 [ 0.   0.   1. ...   6.   0.   0.]
 [ 0.   0.   2. ...  12.   0.   0.]
 [ 0.   0.  10. ...  12.   1.   0.]]
```

Loading an example dataset

and digits.target gives the ground truth for the digit dataset, that is the number corresponding to each digit image that we are trying to learn:

```
digits.target
```

```
array([0, 1, 2, ..., 8, 9, 8])
```

Learning and predicting

In the case of the digits dataset, the task is to predict, given an image, which digit it represents. We are given samples of each of the 10 possible classes (the digits zero through nine) on which we *fit* an estimator to be able to *predict* the classes to which unseen samples belong.

In scikit-learn, an estimator for classification is a Python object that implements the methods `fit(X, y)` and `predict(T)`.

An example of an estimator is the class `sklearn.ensemble.RandomForestClassifier`. The estimator's constructor takes as arguments the model's parameters.

For now, we will consider the estimator as a black box:

```
from sklearn import ensemble

clf = ensemble.RandomForestClassifier(max_depth=3)
```

Choosing the parameters of the model

In this example, we set the value of `max_depth` manually. To find good values for these parameters, we can use tools such as grid search and cross validation.

Learning and predicting

The `clf` (for classifier) estimator instance is first fitted to the data; that is, it must *learn* from the data. This is done by passing our training set to the `fit` method. For the training set, we'll use all the images from our dataset, except for the last image, which we'll reserve for our predicting. We select the training set with the `[:-1]` Python syntax, which produces a new array that contains all but the last item from `digits.data`:

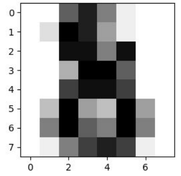
```
clf.fit(digits.data[:-1], digits.target[:-1])
```

Now you can *predict* new values. In this case, you'll predict using the last image from `digits.data`. By predicting, you'll determine the class from the training set that best matches the last image.

```
clf.predict(digits.data[-1:])
```

```
array([8])
```

The corresponding image is:

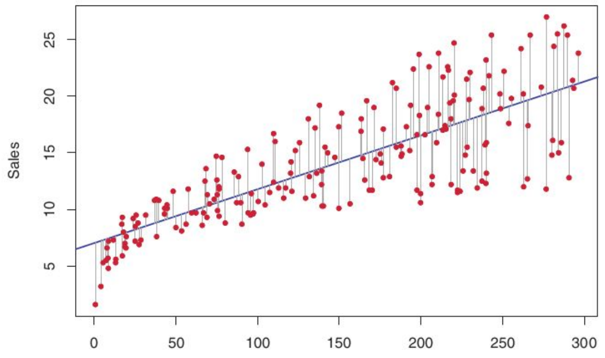


As you can see, it is a challenging task: after all, the images are of poor resolution. Do you agree with the classifier?

Machine Learning approaches

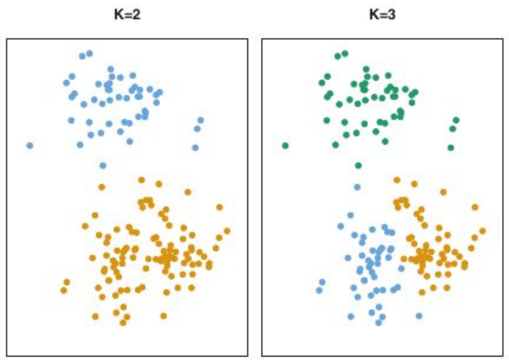
- **Supervised Learning**

Learn a function ***f*** that maps input features to output, given some training data.

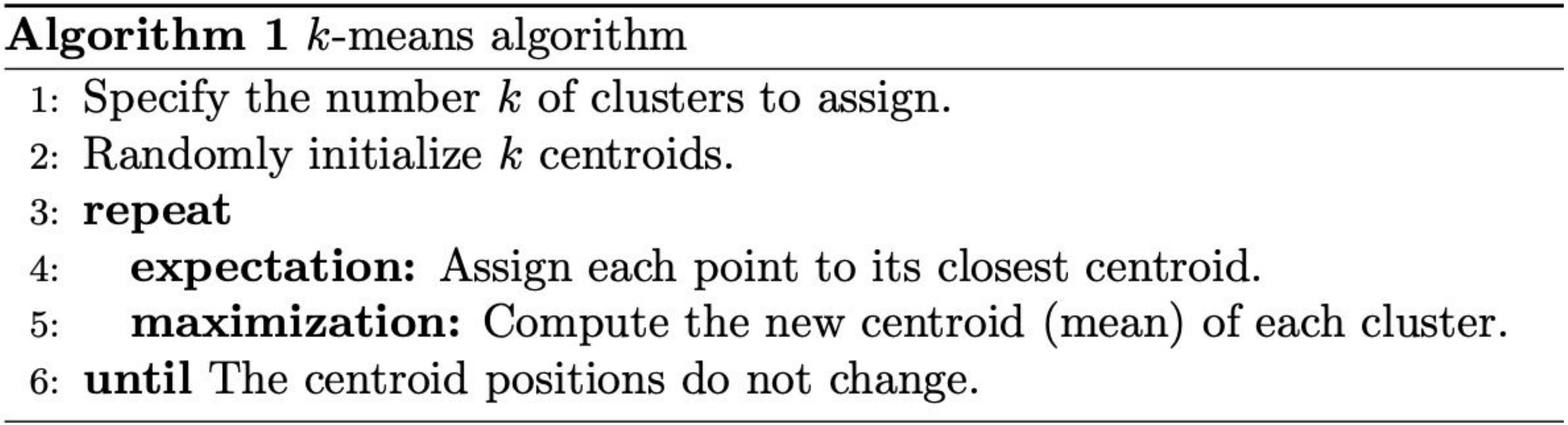


- **Unsupervised Learning**

No output variable! The goal is to understand the relationship between features. e.g. clustering



K-means (reminder)

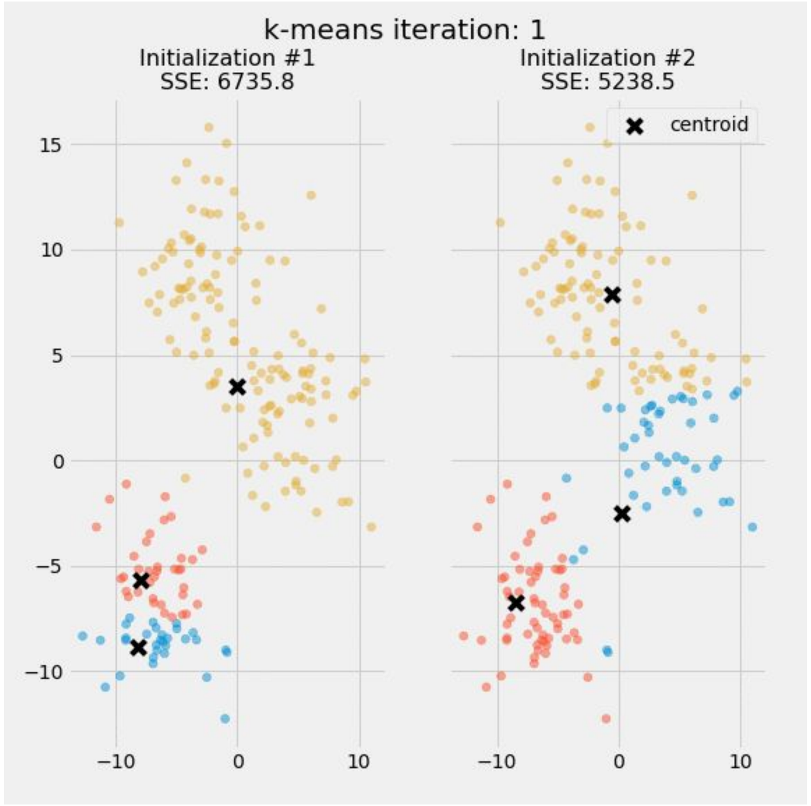


Algorithm 1 *k*-means algorithm

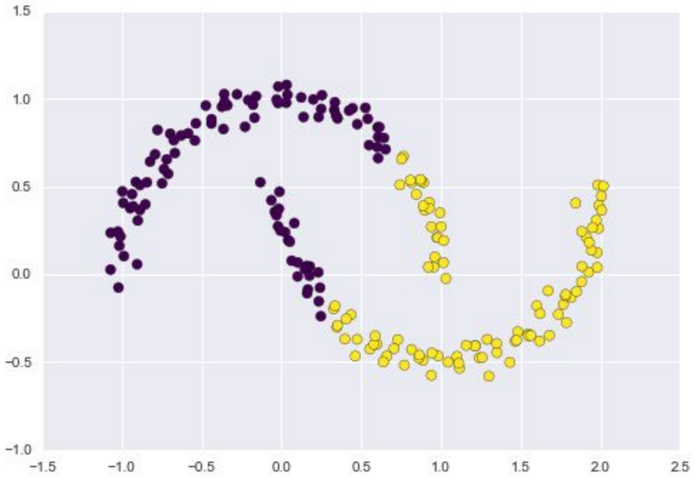
```
1: Specify the number $k$ of clusters to assign.
2: Randomly initialize $k$ centroids.
3: repeat
4:     expectation: Assign each point to its closest centroid.
5:     maximization: Compute the new centroid (mean) of each cluster.
6: until The centroid positions do not change.
```

- **Centroids**: The geometric center of a collection of data points

K-means (details)



1. **Not deterministic**
2. **Global optimum** maybe not be achieved
3. Sensitive to **feature scaling**
4. Limited to **linear cluster boundaries**
5. **K** must be **known**

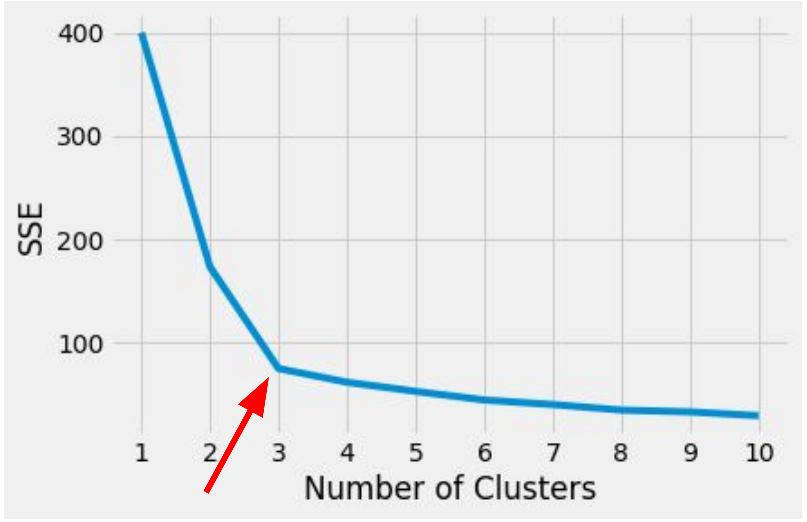


Elbow method

1. Define a **range** for K: [2,...K_max]
2. For each **k** in the range:
   a. **Fit** a K-means model with K=k
   b. **Calculate** the **score*** of the assignments
3. **Plot the score** versus the k
4. Keep the **k on the elbow** of the plot

This offers a **reasonable trade-off** of complexity and performance

*usually the **Sum of Squared Errors (SSE)**



Example code

```
# Imports

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import Kmeans



# Scale the data

scaler = StandardScaler()

scaled_features = scaler.fit_transform(digits.data)



# Calculate the score for different K-mean values

sse = []

for k in range(1, 11):

         kmeans = KMeans(n_clusters=k)

         kmeans.fit(scaled_features)

         sse.append(kmeans.inertia_)
```

```
# Plot the data

plt.style.use("fivethirtyeight")

plt.plot(range(1, 11), sse)

plt.xticks(range(1, 11))

plt.xlabel("Number of Clusters")

plt.ylabel("SSE")

plt.show()
```

!! ATTENTION !!

In this toy example we fitted the StandardScaler using the whole dataset for simplicity.

In reality, when we have data splitted in train/dev/test, **all the transformations/models are fitted on the train subset only!!**

Then, **the dev/test data are transformed using the already fitted (on the train) models**.

Example code:

```
# Split the data

num_test = 100

X_train, X_test, y_train, y_test =
digits.data[:num_test],
digits.data[num_test:],
digits.target[:num_test],
digits.target[num_test:]

# Fit the scaler on the train data only and
transform them at the same time

scaler = StandardScaler()

X_train_scaled =
scaler.fit_transform(X_train)

# Transform the test data using the same
scaler which is fitted on the test data

X_test_scaled = scaler.transform(X_test)
```

Thank you for your attention!

Class is not over though!

Go through the materials covered and try out the code in some of the toy datasets in order to get familiar with scikit-learn.

Read more/references:

- An introduction to Statistical Learning with Application in R, Gareth James, Daniela WItten, Trevor Hastie, Robert Tibshriani: 2.1.4, 2.1.5, 3.1, 4.1, 4.2, 4.3, 9
- Machine Learning: A Bayesian and Optimization Perspective, Sergios Theodoridis: 1, 3.3, 3.4, 11.10
- Scikit-learn documentation: Recognizing handwritten digits: https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html
- Wikipedia - Supervised Learning: https://en.wikipedia.org/wiki/Supervised_learning
- Wikipedia - Unsupervised Learning: https://en.wikipedia.org/wiki/Unsupervised_learning
- GridSearchCV for parameter tuning
- RandomForestRegressor model
- K-means for clustering
- Normalization procedures for the features