

# General vs Narrow AI

Some computer algorithms are capable of mimicking human intelligence, to reason and solve problems on their own, and to apply previously acquired knowledge on completely new types of problems. These algorithms fall into the domain of:

##### Answer the question

**50 XP**

##### Possible Answers

* Artificial Narrow Intelligence (Narrow AI)

press

* Artificial General Intelligence (General AI) **(A)**

press

* Correct! However, General AI is still science fiction, while Narrow AI dominates the current landscape of Data Science.

# Why Python?

Python is the best choice for developing machine learning solutions. Why is that the case?

##### Answer the question

**50 XP**

##### Possible Answers

* Python has a simple and beautiful syntax.

press

* Python is very versatile.

press

* Python is very flexible.

press

* Python offers rich AI-related libraries.

press

* The Python community is big and growing fast.

press

* All of the above. **(A)**

Correct! And precisely for all of these reasons Python's position as the top language for Data Science and Machine Learning will hardly be disputed in the near future.

# The elephant in the room

To further illustrate the concept of Narrow AI, let's see what happens when an algorithm trained for one problem, is given a completely unrelated input and asked for a prediction.

Specifically, what happens when you feed a picture of an elephant into a model that is trained to recognize handwritten digits?

To make things simple, a digit recognition model has been pre-trained. Your task is to feed the elephant image into it and see the result.

##### Instructions

**100 XP**

* Feed the elephant\_image into the digit\_predictor.

In [3]: elephant\_image

Out[3]:

array([[0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0,

0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,

0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1]])

# Load the test example

elephant\_image = load\_elephant()

# Apply the model on the test example

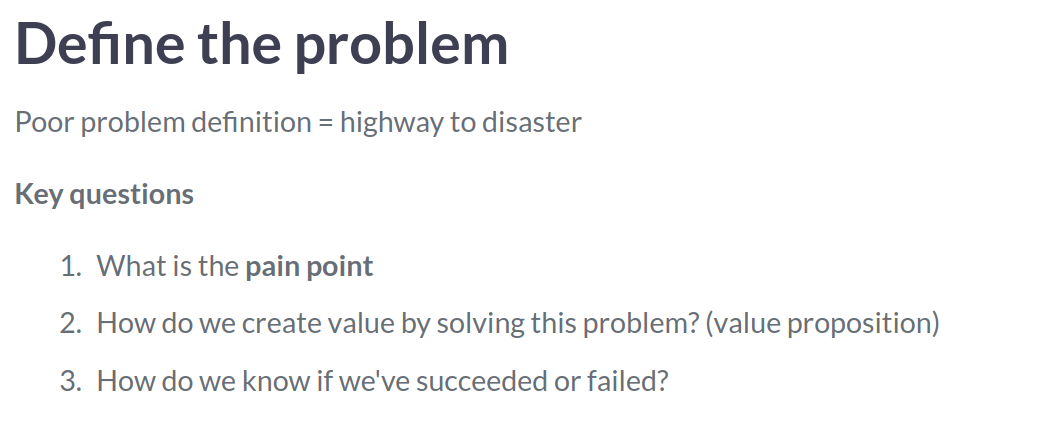
test\_digit\_predictor(elephant\_image, 'elephant')

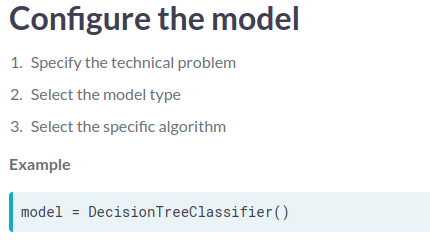
<script.py> output:

Actual digit: elephant

Recognized digit: 7

Interesting, right? Although being the pinnacle of technology, AI systems can also be easily misused and abused -- even tricked, if that is the intention!





# Parameters & hyper-parameters

Let's explore the impact of hyper-parameters on the performance of your model.

You are building a digit recognition algorithm using the LogisticRegression classifier and you are challenged to find the optimal value of its regularization parameter, C.

Lower values of C make the model more conservative and resilient to noise, while higher values give it more flexibility and capacity to capture the nuances. If you increase it too much, you risk overfitting.

Where is the "sweet spot"?

##### Instructions 1/4

**100 XP**

* [1](javascript:void(0))
  + Set the value of hyper-parameter C to 0.000001.

# Define the model

model = LogisticRegression(C=0.000001)

# Fit the model

model.fit(training\_inputs, training\_labels)

# Check model performance

check\_performance(model, testing\_inputs, testing\_labels)

<script.py> output:

Accuracy: 59.30 %

##### Instructions 2/4

**0 XP**

* [2](javascript:void(0))
  + Set the value of hyper-parameter C to 0.0001.

# Define the model

model = LogisticRegression(C=0.0001)

# Fit the model

model.fit(training\_inputs, training\_labels)

# Check model performance

check\_performance(model, testing\_inputs, testing\_labels)

<script.py> output:

Accuracy: 89.19 %

Set the value of hyper-parameter C to 0.01.

# Define the model

model = LogisticRegression(C=0.01)

# Fit the model

model.fit(training\_inputs, training\_labels)

# Check model performance

check\_performance(model, testing\_inputs, testing\_labels)

<script.py> output:

Accuracy: 94.59 %

Set the value of hyper-parameter C to 100.

# Define the model

model = LogisticRegression(C=100)

# Fit the model

model.fit(training\_inputs, training\_labels)

# Check model performance

check\_performance(model, testing\_inputs, testing\_labels)

<script.py> output:

Accuracy: 92.53 %

Good job! Your model was obviously over-regularized in the first iteration and increasing C gave it some slack to capture the finer variations in the data. However, increasing parameters beyond the "sweet spot" is not only useless, but often detrimental!

# Too much of a good thing

Let's inspect the issue of **overfitting** in a more visual way.

Say you have measured the temperature in your office for several days and you want to create a model to describe these oscillations.

You have (wisely) decided to tackle this challenge using a **polynomial model**. Your task is now to find the right value of the hyper-parameter representing the **degree of the polynomial.**

**But be careful!** As your measurements are quite noisy (because you used a cheap thermometer), you're in **danger of overfitting** if you exaggerate with model complexity.

A fit\_polynomial() function has been defined for you.

##### Instructions 1/3

**35 XP**

* [1](javascript:void(0))
  + Set the degree of the polynomial to 1.

# Set degree to 1

fit\_polynomial(degree=1)

* + Set the degree of the polynomial to 4.

# Set degree to 4

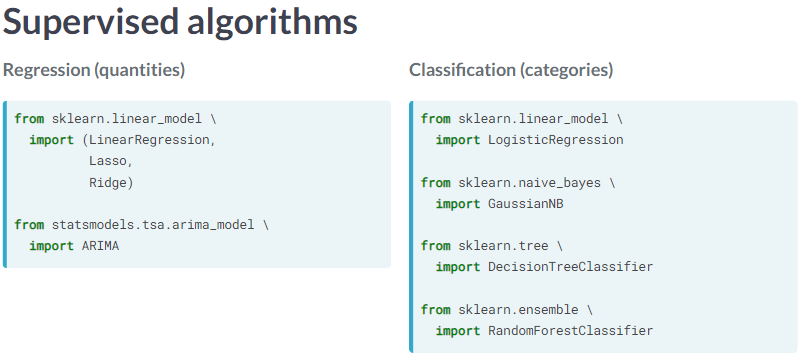
fit\_polynomial(degree=4)

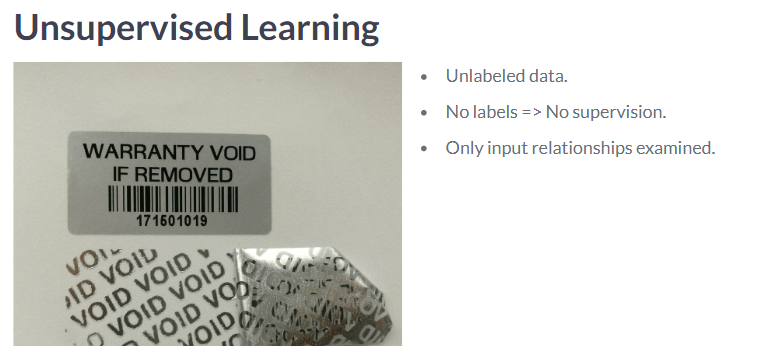
* + Set the degree of the polynomial to 30.

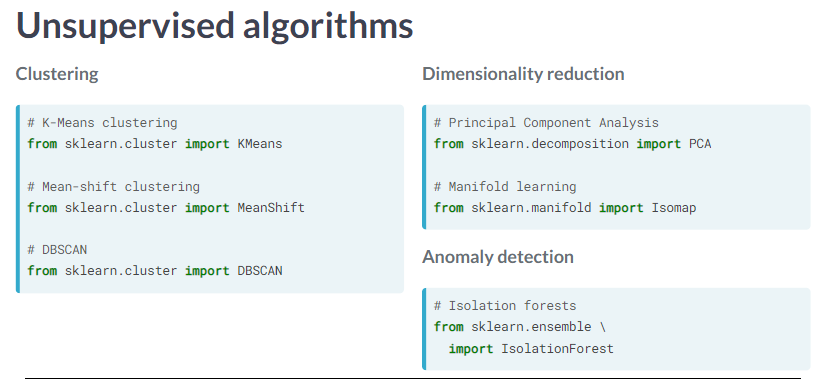
# Set degree to 30

fit\_polynomial(degree=30)

Setting the degree of the polynomial to 1 captured the trend, but was not really impressive. Setting it to 4 looked better, but increasing to 30 did not. Lesson learned: increasing the model complexity beyond the sweet spot just makes it capture more and more noise.







# Guess the flavor I

A model is trained based on a collection of pictures to recognize cats and dogs in pictures, annotated with appropriate labels. Into which domain does this model fall?

##### Answer the question

**50 XP**

##### Possible Answers

* 

This model falls into the domain of **reinforcement learning**.

press1

* 

This model falls into the domain of **unsupervised learning**.

press2

* 

This model falls into the domain of **supervised learning**. **(A)**

press

"Yes! As the model uses labeled data, it falls in the model of supervised learning."

# Guess the flavor II

You want to apply machine learning on a large collection of news articles and cluster them into groups, so that you could identify and count recurring topics, while ignoring the existing news categorization.

Which flavor of Machine Learning algorithms is the most appropriate for this task?

##### Answer the question

**50 XP**

##### Possible Answers

* 

**Reinforcement learning** is the most appropriate approach for this task.

press1

* 

**Unsupervised learning** is the most appropriate approach for this task. **(A)**

press2

* 

**Supervised learning** is the most appropriate approach for this task.

Press

"Yes! As you are trying to cluster data without using labels, unsupervised learning is appropriate for this task."

# Simple digit recognition

In this exercise,you want to build, train and test the simplest digit recognition algorithm.

However simple or complex the Machine Learning problem at hand may be, it will always contain the following steps:

* Data loading, preparation and splitting into the train and test partitions
* Model selection and training ("fitting")
* Model performance assessment

For this purpose, you will load a dataset containing 1797 images of individual digits, 8x8 pixels in size.

Just remember that you use regressors for predicting quantities and classifiers when dealing with categories at the output.

evaluate\_predictions() has been defined for you.

##### Instructions

**100 XP**

* Select the right model for the task at hand: use LinearRegression() if it’s a regression problem and DecisionTreeClassifier() if it’s a classification problem.
* Fit the model using the training inputs X\_train and training labels y\_train.
* Generate predictions on the X\_test DataFrame.
* Evaluate model performance using testing inputs y\_test and testing labels prediction\_results.

# Select the model appropriate for the task

model = DecisionTreeClassifier()

# Train the model

model.fit(X=X\_train , y=y\_train)

# Generate predictions

prediction\_results = model.predict(X=X\_test)

# Test the model

evaluate\_predictions(y\_true=y\_test,

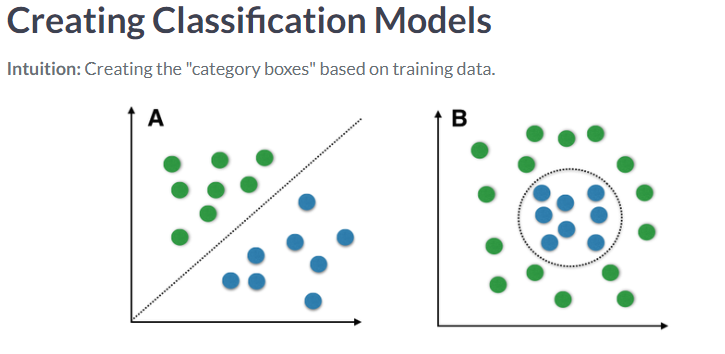
y\_pred=prediction\_results)

<script.py> output:

Accuracy: 0.00 % (Linear regression)

<script.py> output:

Accuracy: 84.01 % (DecisionTreeClassifier)



# Regression or Classification I

Your company has a problem with too many suppliers failing to meet their delivery deadlines, which creates a lot of downstream problems in your production and many angry customers.

Your boss tells you that you have a very rich database of supplier data and asks you to build an algorithm for early prediction of suppliers which are too risky to work with.

What kind of model should you use?

##### Answer the question

**50 XP**

##### Possible Answers

* 

**Regression** should be used to predict risky suppliers.

press1

* 

**Classification** should be used to predict risky suppliers. **(A)**

Press

Exactly. You are trying to find a separation boundary between the domains of low-risk and high-risk suppliers, which is textbook example of a classification problem.

# When regression means classification

OK, so you have a classification problem at your hands. Neat.

Now you talk to your client, which tells you that the algorithm needs to be implemented on their legacy systems, with very poor computing resources.

It turns out you can only run the simplest of models there and you have only two options: LinearRegression() and LogisticRegression().

But you need a classifier and these models have "regression" in their name, can you really pull this off?

If you paid attention in the previous video lesson, you know there's no reason to worry so just go ahead and select the right algorithm for your problem.

##### Instructions

**100 XP**

* Select the right algorithm for your classification problem.

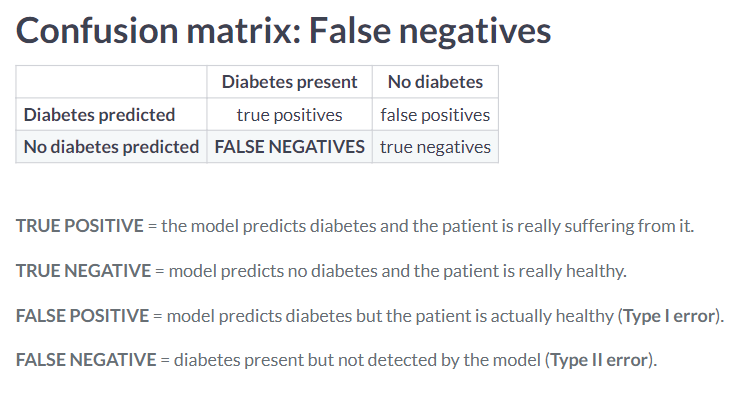
# Select the model

model = LogisticRegression()

# Train the model

model.fit(X\_train, y\_train)

Good job! Not all machine learning algorithms have intuitive names and some names are plain contradictory – Logistic Regression being the best example of that. Luckily this is not the norm, but the one you just saw is usually the most confusing one.



# SPAMtastic!

You have built an e-mail SPAM filtering algorithm.

Which metric is best suited to quantify the percentage of emails your algorithm flags as spam that are actually spam?

##### Answer the question

**50 XP**

##### Possible Answers

* 

Accuracy.

press1

* 

Precision. **(A)**

press2

* 

Recall.

Press

Correct! Still, you should examine recall also, in order to get a sense of how many real spam emails are actually flagged as such.

# Hold-out

You already know about the danger of overfitting, which occurs when your model learns the training data too well, but then performs poorly when faced with new data.

Because of that, you've been urged to always test your model using data that wasn't previously used for training.

But don't take our word for it, see for yourself!

You will use a dataset consisting of two classes. 60% of the data has been selected for training and stored in X\_train and y\_train. The remaining 40% is stored in variables X\_test and y\_test.

You will train a RandomForestClassifier() model and see the difference in performance:

* when it's applied on the very same data used to train it
* when it's applied on data just slightly different from the training set

##### Instructions 1/2

**50 XP**

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
  + Test the model on the same data it used for training.

# Model setup

model = RandomForestClassifier(n\_estimators=5, max\_depth=20)

# Model fitting

model.fit(X\_train, y\_train)

# Testing on training data

test\_and\_show\_accuracy(model,

X\_test=X\_train,

y\_test=y\_train)

<script.py> output:

Accuracy: 96.67 %

##### Instructions 2/2

**50 XP**

* [2](javascript:void(0))
  + Test the model on the hold-out dataset, that is, the data the model hasn't seen during training.

# Model setup

model = RandomForestClassifier(n\_estimators=5, max\_depth=20)

# Model fitting

model.fit(X\_train, y\_train)

# Testing on testing data

test\_and\_show\_accuracy(model,

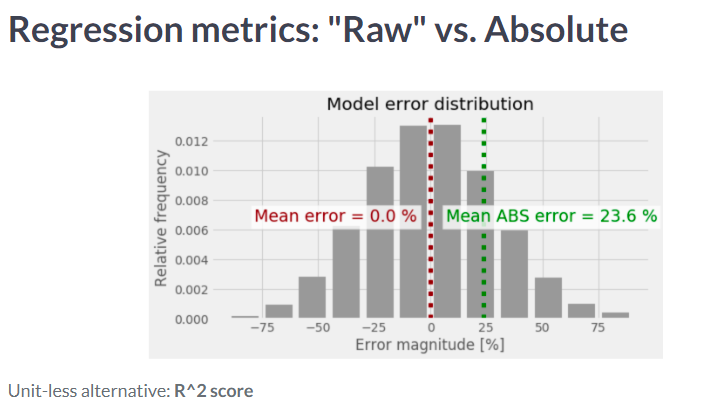
X\_test=X\_test,

y\_test=y\_test)

<script.py> output:

Accuracy: 81.33 %

Whoah! So when evaluating on training data, the accuracy reaches 97.1%, but on unseen data it drops to 79%? This is almost a 20% difference, something you should definitely realize and solve before putting your model in production!



# Tailor made

You need to estimate the goodness-of-fit of your regression model to your data in a relative, unit-less manner. Which metric should be your first choice?

##### Answer the question

**50 XP**

##### Possible Answers

* 

The **mean absolute error**.

press1

* 

The **median absolute error**.

press2

* 

The **R^2 score**. **(A)**

Press

Correct! Using R^2 score is a nice and simple measure of goodness-of-fit which all Data Scientists should understand and use.

# Going non-linear

You're about to see how it's possible to model non-linear relationships using linear regression models.

To make this "transition", you don't have to change anything in the model, you just have to generate:

* higher order features: (a)→(a,a2,a3,...)(a)→(a,a2,a3,...), and
* interaction features: (a,b)→(a∗b,a2∗b,a∗b2,...)(a,b)→(a∗b,a2∗b,a∗b2,...)

You will first try to fit a purely linear model to a quadratic process and check the R^2 score.

After that you'll use the function PolynomialFeatures() to generate, well, polynomial features and see how much better your fit is -- both visually and according to the R^2 score.

Finally, you've been provided with the custom function check\_model\_fit() that plots the model predictions against actual data and prints the R^2 score of your model.

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Select the appropriate method to call the model training procedure.

# Select the appropriate method to call the model training procedure

linear\_model.fit(X=x\_train, y=y\_train)

# Check the "goodness-of-fit" of the fitted model

check\_model\_fit(model=linear\_model,

x=x\_train,

y=y\_train)

<script.py> output:

R^2 score: 81.22 %

##### Instructions 2/2

**50 XP**

* [2](javascript:void(0))
* Using the original dataset x\_train, create a new dataset x2\_train with 2nd degree polynomial features.
* Fit the linear\_model using the newly created dataset.
* Check the goodness of fit of your fitted linear model.

# Create a new dataset x2\_train with 2nd degree polynomial features.

poly = PolynomialFeatures(degree=2)

x2\_train = poly.fit\_transform(x\_train)

# Fit the linear model using the newly created dataset

linear\_model.fit(X=x2\_train, y=y\_train)

# Check the "goodness-of-fit" of the fitted model

check\_model\_fit(model=linear\_model,

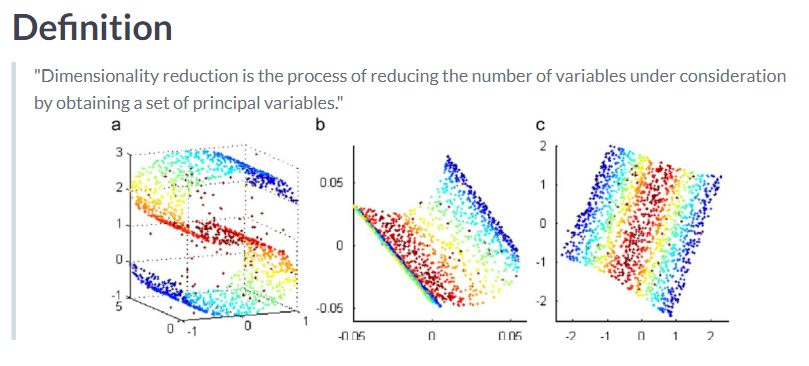
x=x2\_train,

y=y\_train)

<script.py> output:

R^2 score: 93.13 %

Amazing! A linear model that perfectly follows a quadratic process, that is the magic trick with polynomial features! And look how easy it is to produce them using scikit-learn's standard functions. Still, always make sure to understand the underlying principles thoroughly!



# Principal Component Analysis

To which family of dimensionality reduction algorithms does Principal Component Analysis belong?

##### Answer the question

**50 XP**

##### Possible Answers

* 

Principal Component Analysis is an example of **linear** dimensionality reduction algorithm.

**(A)**

press1

* 

Principal Component Analysis is an example of **non-linear** dimensionality reduction algorithm.

Press

Correct! Being a linear, fast and deterministic algorithm makes PCA the 1st choice to test when going with dimensionality reduction.

# Visitors from outer space

So you concluded you must resort to dimensionality reduction because of very limited computational resources you have available for crunching your hyper-dimensional dataset.

And for the same reason, you feel that the PCA algorithm is the best choice due to its speed and simplicity.

**Good. But did you check your data for outliers?** Let's see how they could impact your results.

A 3-dimensional dataset of 1000 samples (X\_raw), slightly "contaminated" with 5 outliers (X\_new), has been pre-loaded, as seen on Figure 1.

On Figure 2 you see that the impact of these outliers (in red) is trivial and creates no problem in extracting actual principal components.

But what happens if they are further away?

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Add 5 outliers with an outlier\_distance of 200.

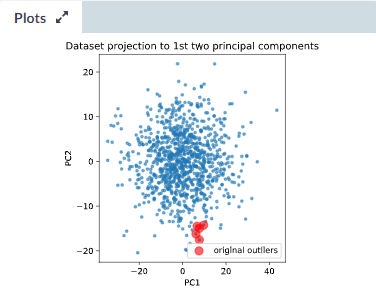
# Add outliers to the blob

X\_new, outliers = add\_outliers(X\_raw,

outlier\_distance= 200,

n\_outliers= 5)

plot\_3d\_data(X\_new, outliers)



##### Instructions 2/2

**50 XP**

* Extract first two principal components of the contaminated dataset X\_new using the helper function extract\_components().

# Add outliers to the blob

X\_new, outliers = add\_outliers(X\_raw, outlier\_distance=200, n\_outliers=5)

plot\_3d\_data(X\_new, outliers)

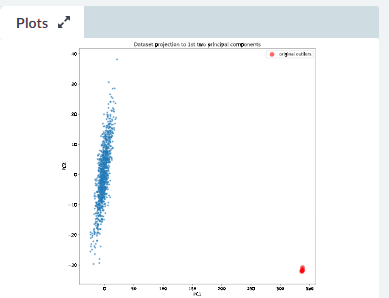
# Extract principal components

X\_2D, outliers\_2D = extract\_components(X\_new, outliers, n\_components=2)

# Plot the PCA results

plot\_2d\_data(X\_2D, outliers\_2D)

Correct answer... but that doesn't seem right, does it? The 1st principal component is now completely skewed towards the outliers, but how can 5 outliers outweigh 1000 normal samples? The problem is in the magnitude of these outliers, which dwarfs the variations observed in normal samples. Key takeaway: before any dimensionality reduction, always make sure to properly explore your data and treat outliers if needed.



# Lucky number K

Beginners in Machine Learning often have very optimistic ideas that Machine Learning can produce amazing insights with little to no human involvement and decision making.

The truth is that the performance of your algorithms is heavily influenced by parameters that you as a human define before the model has seen any data.

In the case of clustering, most algorithms still require you to be explicit about the number of clusters you are looking for. But not all!

Which of the following clustering algorithms determines the number of clusters on its own?

##### Answer the question

**50 XP**

##### Possible Answers

* 

K-means clustering determines the number of clusters on its own.

press1

* 

Spectral clustering determines the number of clusters on its own.

press2

* 

DBSCAN determines the number of clusters on its own. **(A)**

Press

That's it! DBSCAN determines the number of clusters on its own, but that doesn't mean that the discovered clusters will always be meaningful! Make sure to always inspect the results and adjust algorithm hyper-parameters if needed.

# Elbow reading

Determining the right number of clusters is one of the most crucial steps in developing a clustering model.

In this exercise, you will apply K-means clustering and the "elbow method" to determine the correct number of clusters present in the dataset at hand.

The data is loaded in the variable X and you have been provided with two functions for your convenience, plot\_clusters() and plot\_elbow\_curve(), to facilitate the discovery process.

Your task is to specify the range of numbers of clusters over which to scan in order to produce the "elbow curve".

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Set the values of start and stop parameters of the range() function in the first line to 2 and 6, respectively.

k\_range = list(range(2, 6))

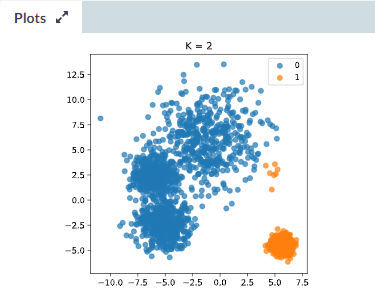
summed\_distances = []

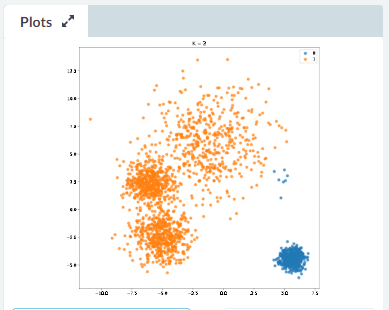
for k in k\_range:

kmeans.set\_params(n\_clusters=k).fit(X)

summed\_distances.append(kmeans.inertia\_)

plot\_elbow\_curve(k\_range, summed\_distances)





##### Instructions 2/2

**50 XP**

* [2](javascript:void(0))

#### Question

According to the shown curve and scatter plot, what is the most reasonable number of clusters in this dataset?

##### Possible Answers

* 

3

* 

4 **(A)**

* 

5

That's correct. The sum of squared distances stops dropping significantly for values of K above 4.

# DBSCAN

DBSCAN is another very popular clustering algorithm, belonging to density-based algorithms.

For beginners it can seem very attractive because it doesn't require the number of clusters to be defined in advance.

But there's no free lunch and relying on DBSCAN to find the right number of clusters completely on its own can be a big trap.

Let's illustrate this by playing with DBSCAN's hyper-parameter eps, which defines the maximum distance between points within the same cluster.

##### Instructions 1/3

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Set the value of eps to 0.1 and check the clustering result.

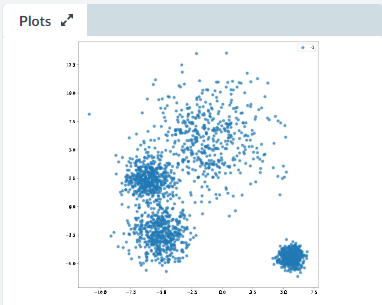
# Set eps to 0.1

eps = 0.1

dbscan.set\_params(eps=eps)

clusters = dbscan.fit\_predict(X)

plot\_clusters(X, clusters)



##### Instructions 2/3

**35 XP**

* [2](javascript:void(0))
* [3](javascript:void(0))
* Setting the maximum distance t 0.1 did not yield any clusters. Try setting the value of eps to 0.5, and check the clustering result again.

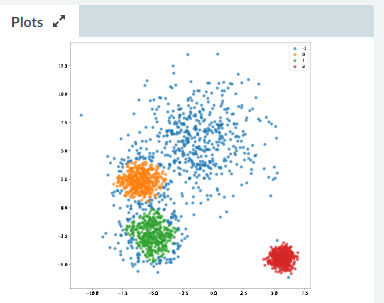
# Set eps to 0.5

eps = 0.5

dbscan.set\_params(eps=eps)

clusters = dbscan.fit\_predict(X)

plot\_clusters(X, clusters)



##### Instructions 3/3

**30 XP**

* [3](javascript:void(0))
* You found 3 clusters! However there are still a lot of points that don't belong to any cluster. Set the value of eps to 2 and see what happens.

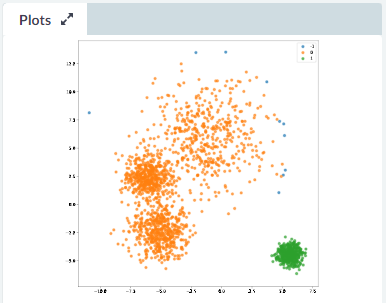
# Set eps to 2

eps = 2

dbscan.set\_params(eps=eps)

clusters = dbscan.fit\_predict(X)

plot\_clusters(X, clusters)



Increasing eps to 2 did cluster more points, but it also merged two clusters together. None of the options you tried seems close to acceptable, right? DBSCAN is a density-based algorithm and as such might be completely inappropriate for certain datasets, because it tends to merge clusters with overlapping regions. Choose your algorithm wisely and always try multiple options!

# How unsupervised really?

When evaluating the performance of Anomaly Detection models, you most often use metrics from the domain of:

##### Answer the question

**50 XP**

##### Possible Answers

* 

Supervised learning **(A)**

press1

* 

Unsupervised learning

Press

Correct. Just because you don't have enough labeled data to apply supervised LEARNING, doesn't mean that you shouldn't use any amount available to test the performance of your model.

# The go-to algorithm

Despite being a bit more computationally intensive then other methods, an algorithm is commonly used for **anomaly detection**. Which algorithm is it?

##### Answer the question

**50 XP**

##### Possible Answers

* 

**One-Class SVM** is commonly used for anomaly detection.

press1

* 

**Isolation Forest** is commonly used for anomaly detection. **(A)**

press2

* 

**Robust covariance** is commonly used for anomaly detection.

Press

Correct. For most applications and given the general availability of computational power today, Isolation Forests are the go-to algorithm for anomaly detection.

# The odd one out

You saw previously that the IsolationForest() algorithm is a great first choice when in need of anomaly or outlier detection.

In this exercise you want to examine how the ratio of inliers to outliers (a.k.a. signal to noise ratio) affects its ability to detect anomalies.

The IsolationForest() algorithm has been loaded for you in the variable called isolation\_forest, and a helper function make\_fake\_data() was loaded as well. Your task is to gradually increase the number of outliers and observe the difference in results in each iteration.

##### Instructions 1/3

**35 XP**

* [1](javascript:void(0))
  + Set the number of outliers to 50 (5% of inliers).

# Generate data comprising of the "clean" and "noisy" components

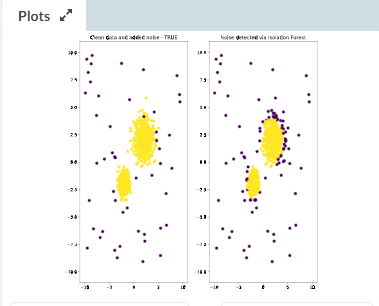
noisy\_data, true\_labels = make\_fake\_data(n\_blobs=2, n\_inliers=1000, n\_outliers=50)

# Detect anomalies

predicted\_anomalies = isolation\_forest.fit\_predict(noisy\_data)

# Plot results

plot\_detected\_anomalies(noisy\_data, true\_labels, predicted\_anomalies)



Set the number of outliers to 200 (20% of inliers).

# Generate data comprising of the "clean" and "noisy" components

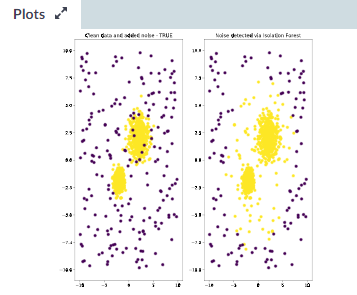
noisy\_data, true\_labels = make\_fake\_data(n\_blobs=2, n\_inliers=1000, n\_outliers=200)

# Detect anomalies

predicted\_anomalies = isolation\_forest.fit\_predict(noisy\_data)

# Plot results

plot\_detected\_anomalies(noisy\_data, true\_labels, predicted\_anomalies)



* Set the number of outliers to 500 (50% of inliers).

# Generate data comprising of the "clean" and "noisy" components

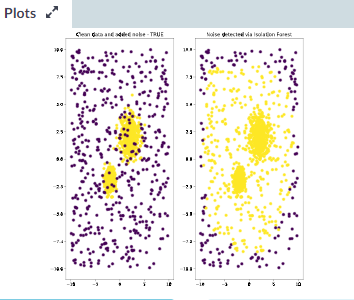
noisy\_data, true\_labels = make\_fake\_data(n\_blobs=2, n\_inliers=1000, n\_outliers=500)

# Detect anomalies

predicted\_anomalies = isolation\_forest.fit\_predict(noisy\_data)

# Plot results

plot\_detected\_anomalies(noisy\_data, true\_labels, predicted\_anomalies)



Good. Although you clearly see that Isolation Forests, even without any tuning, do an amazing job of detecting anomalies, they can not do miracles if your data is severely contaminated. It's good to remember that in that case they will tend to overestimate the number of anomalies, raising a lot of false alarms

# Elon's tweets

In this exercise, you will attempt the impossible: detecting patterns in Elon Musk's tweets!

You will apply two unsupervised learning algorithms:

* Dimensionality reduction, to translate your text data into a 2D space.
* Clustering, to find groups of similar tweets.

The go-to model for dimensionality reduction is Principal Component Analysis (PCA), while the KMeans algorithm represents the same in the domain of clustering.

* Tweets in their raw form were loaded into the variable named tweets\_raw.
* They have also been translated into a machine-digestible, vectorized form, contained in the variable tweets\_matrix.
* To write less code, we want you to use the functions for combined fitting and transformation/prediction - .fit\_transform() and .fit\_predict()

Be aware that this is real data from Twitter and as such there is always a risk that it may contain profanity or other offensive content (in this exercise, and any following exercises that also use real Twitter data).

##### Instructions

**100 XP**

##### Instructions

**100 XP**

* Set the algorithm for dimensionality reduction and the number of dimensions to 2.
* Apply dimensionality reduction.
* Configure the clustering model to find two clusters in the input data.
* Find clusters and display the results.

In [1]: tweets\_matrix

Out[1]:

array([[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

...,

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0]])

# Set the number of dimensions to 2

dimensionality\_reducer = PCA(n\_components= 2)

# Apply dimensionality reduction

tweets\_reduced = dimensionality\_reducer.fit\_transform(tweets\_matrix)

# Configure the clustering model

clustering\_model = KMeans(n\_clusters=2)

# Find clusters

tweet\_clusters = clustering\_model.fit\_predict(tweets\_reduced)

# Show the clustering results

print\_cluster\_tweets(tweet\_clusters, tweets\_raw)

<script.py> output:

================================================================================

CLUSTER # 0

--------------------------------------------------------------------------------

30 Glad you're ok @tylerthecreator https://t.co/...

39 RT @Tesla: Tesla Safety Update https://t.co/8D...

45 @JamesLandino im actually cat girl here's self...

49 Fire and Ice https://t.co/ulgSZ1wlPl

56 RT @TeslaOwnersUK: @elonmusk @Model3Owners It ...

59 My twitter feed rn https://t.co/48Ay7CxR56

62 RT @cleantechnica: Tesla Powerwall Keeps A Rem...

65 Fresh puro from my meme dealer https://t.co/Ed...

71 How'd they know!? https://t.co/AZh4EOrgmc

78 RT @Tesla: Overhead views from a Model 3 https...

83 @vicentes @Grimezsz Wanna buy some Bitcoin? h...

86 @Kirinodere https://t.co/2exdO1Hoks

90 Order Tesla without test drive &amp; you get 3...

93 Software V9.0 with latest Autopilot is rolling...

112 @BoredElonMusk Digital Style! https://t.co/aBk...

113 RT @Teslarati: Tesla updates Referral Program ...

125 Had to been done ur welcome https://t.co/7jT0f...

134 Just released lower cost, mid-range Tesla Mode...

135 RT @cleantechnica: Rewind: Akon used to have 2...

138 Dog saves her puppies https://t.co/czPVEwpPhi

140 RT @nichegamer: @elonmusk Every time I think o...

142 Tomorrow brings a lemur https://t.co/rm6S17h35q

143 Special Circumstances https://t.co/Kk36qj6XJt

150 Insightful article by David @Pogue https://t.c...

159 Model 3 motor &amp; gearbox still in good cond...

160 Works best if you play games full screen &amp;...

161 https://t.co/iiIWjhKiLS

169 Love Your Name https://t.co/fRU7nTWnML

182 RT @InsideEVs: Tesla Out Delivers Porsche In Q...

183 Visual approximation https://t.co/sMn3Pv476Y

...

872 @S\_Padival @lopezlinette Why did @lopezlinette...

873 @thesheetztweetz @CNBC @Lebeaucarnews Lopez wr...

874 @S\_Padival @lopezlinette https://t.co/ystkbF7p1W

875 @S\_Padival @lopezlinette .@Lopezlinette posted...

887 RT @cleantechnica: A Sinister Cellar Of The TS...

888 RT @cleantechnica: Business Insider Resorts To...

889 Note, no one is in the car or controlling remo...

891 RT @ElectrekCo: Tesla Powerwalls save the day ...

912 Whoop whoop https://t.co/Xq3JlcQizc

913 The Spice Flows https://t.co/YMyhVGIxHc

920 Who likes short shorts? https://t.co/hoKsDT8xdS

924 Excession https://t.co/h4AYwxbS3h

925 Space Laser https://t.co/8CQ6L9si3f

926 @stui999 https://t.co/AZ4aidXs33

928 Spacebow https://t.co/Ht4GCsaQo9

929 RT @Bob\_Richards: Wow. That was beautiful. Tha...

930 RT @AntVenom: @SpaceX @elonmusk @Space\_Station...

931 RT @NASAJPL: Bon voyage, #ECOSTRESS! (That's i...

935 The Spice. Must. Flow. https://t.co/ov4aLIwzKz

936 Open the Knaack HAL https://t.co/EG47rvPppF

948 Paint shop https://t.co/G2rIe1qYG4

965 @pulletsforever All Tesla vehicles come with c...

974 You're welcome! https://t.co/NcsGhhB1Ja

981 @cubamark That goes without saying https://t.c...

983 RT @elonmusk: @Penenberg For sure! When doing ...

984 RT @SpaceX: At Naval Air Facility El Centro in...

988 @Penenberg For sure! When doing US govt crash ...

997 @justwidle2 @Tesla Then you will really hate t...

998 RT @Tesla: Tesla team celebrates Pride Month [...

999 RT @Teslarati: Tesla Model 3 beats runner-up P...

Name: Text, Length: 238, dtype: object

================================================================================

CLUSTER # 1

--------------------------------------------------------------------------------

0 Legally required officers of a corporation are...

1 @FredericLambert Good point, might put that on...

2 Deleted my Tesla titles last week to see what ...

3 @SnazzyQ @FredericLambert Signing off Twitter ...

4 @SnazzyQ @FredericLambert Don't worry, we'll m...

5 @CGasparino @Tesla The FOIA on this will be so...

6 @Model3Owners Not going up. We will just offer...

7 @Lindowitz @FredericLambert It's a reasonable ...

8 @CGasparino @Tesla Exactly, this is total bs. ...

9 @sk2sno @rtk86 @sheldonth @FactsDataTruth @Tes...

10 @FredericLambert It was underpriced. Anyway, i...

11 @rtk86 @sheldonth @FactsDataTruth @Tesla Ok, w...

12 @ctgm @FredericLambert Yeah, or free lifetime ...

13 @FredericLambert Didn't you get free Superchar...

14 @annerajb @FredericLambert @pjj\_knowles @yames...

15 @officialjaden Yes. Will require tapping indic...

16 @pjj\_knowles @yames51 @DigitalDunzey No, but I...

17 @mangeHDbackup Isn't it obvious?

18 @yames51 @DigitalDunzey Worth it

19 @vincent13031925 @cleantechnica @CNBC @busines...

20 Tesla Autopilot Drive on Navigation going to w...

21 @sheldonth @FactsDataTruth @Tesla Not exactly....

22 @fb1975 @DigitalDunzey Haha

23 @DigitalDunzey On Insta, 10% of followers like...

24 @\_Akhaten @nichegamer @raissabontempo @SomeGra...

25 @nichegamer @raissabontempo @SomeGrayAreas @re...

26 @SomeGrayAreas @reddit @Twitter Reddit's twitt...

27 @Mcgillligan Exactly

28 @FactsDataTruth @Tesla You're right. Earlier M...

29 On Twitter, likes are rare &amp; criticism is ...

...

961 @TacocaTs @hipquark @lisaprank @jovanik21 I ju...

962 @hipquark @lisaprank @jovanik21 Lisa, I popula...

963 @coronadetucson Bronco rocks

964 @rajmathai @Tesla @Teslarati No standard autom...

966 @ArtemR @JoeyOstrander Truck has a lot more ro...

967 @clprenz @JoeyOstrander I hate whole idea of b...

968 @TheSeanBooker Sir, this will not be some a da...

969 @JoeyOstrander 400 to 500 mile option definite...

970 @JasonRBNY @lisaprank @jovanik21 Soul meuniere

971 @MattPTurner Should be easy to add as a featur...

972 @ZacksJerryRig For sure

973 @DMC\_Ryan July

975 @LikeTeslaKim @phycho\_hippie 6

976 @lisaprank @jovanik21 Was actually someone els...

977 @AllIsStar721 @lisaprank @jovanik21 Randomly s...

978 @lisaprank I think Nik @jovanik21 did an illus...

979 @TheHoff525 Yes. Highly recommend doing so btw.

980 @clintdebusk That will be standard

982 @phycho\_hippie 300,000 lb towing capacity

985 @wbwhitehouse It will

986 @sharynassimi

987 @MuellerJeffrey Wow, great idea! Since it will...

989 Pickup truck will have power outlets allowing ...

990 The Tesla Truck will have dual motor all-wheel...

991 @erocksoren Seems like trear gate should rotat...

992 @aMidLifeCrisis It will parallel park automati...

993 What would you love to see in a Tesla pickup t...

994 @justinfiaschett @EV20018 @justwidle2 @Tesla G...

995 @justwidle2 @Tesla Don't buy our car if that's...

996 @EV20018 @justwidle2 @Tesla "Virtue signaling"...

Name: Text, Length: 762, dtype: object

Fantastic! Clustering helps you find patterns in the data, but the result is strongly influenced with how the data was prepared before being served to the learning algorithm, as well as your choice of the number of clusters you are looking for. Here you can see three groups of points with your eyes, but you have asked the algorithm to split the points into two clusters -- and that's what it did. It is a very rough divide and it looks like the algorithm has mainly put all retweets into one bucket and all other tweets in the second one.

# Fruits of knowledge

When you want to know why your model has made a certain decision for a specific single record, you are engaging in so-called "local model interpretation".

Currently, the most popular algorithm for this purpose has a very "fruity" acronym. Which one is it?

##### Answer the question

**50 XP**

##### Possible Answers

* 

LEMON

press1

* 

LIME **(A)**

press2

* 

ORANGE

press3

* 

PAPAYA

Press

Yes! LIME stands for "Local Interpretable Model-Agnostic Explanations". Although it's a fairly recent algorithm, it already has implementations in several Machine Learning libraries in Python, such as "eli5".

# Predicting customer churn

Congratulations! You have just been hired as a Junior Data Scientist for a big telecommunications company.

On your first day, you are asked to help with a big problem the company is struggling with: predicting customer churn.

Luckily for you, your colleague has already prepared the dataset for you. You just need to use it to train a predictive model and determine its performance.

The exercise datasets have been loaded for your convenience:

* client\_data holds the inputs (gender, tenure, monthly costs, number of dependents, etc)
* client\_churned holds information on whether this client churned or not ('Yes', 'No')

As for the models, you have the RandomForestClassifier and LinearRegression at your disposal -- choose wisely!

print\_metrics

##### Instructions

**100 XP**

##### Instructions

**100 XP**

* Select your model.
* Divide the data into the training and testing set.
* Train the model.
* Generate predictions on the test set and evaluate model performance using accuracy, precision and recall as metrics.

# Define your model

model = RandomForestClassifier()

# Divide the data into the training and testing set and train the model

X\_train, X\_test, y\_train, y\_test = train\_test\_split(client\_data , client\_churned)

model.fit(X\_train, y\_train)

# Generate predictions on the test set

predictions\_test = model.predict(X\_test)

# Evaluate the model predictions using metrics appropriate for this problem class

print\_metrics(target\_test=y\_test,

predictions=predictions\_test,

metrics=['accuracy', 'precision', 'recall'])

<script.py> output:

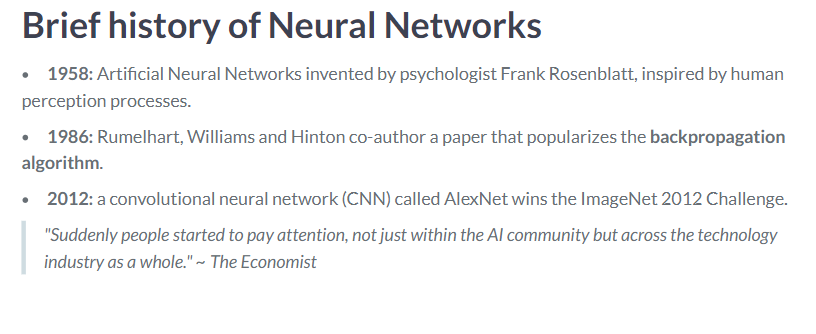
----------------------------------------

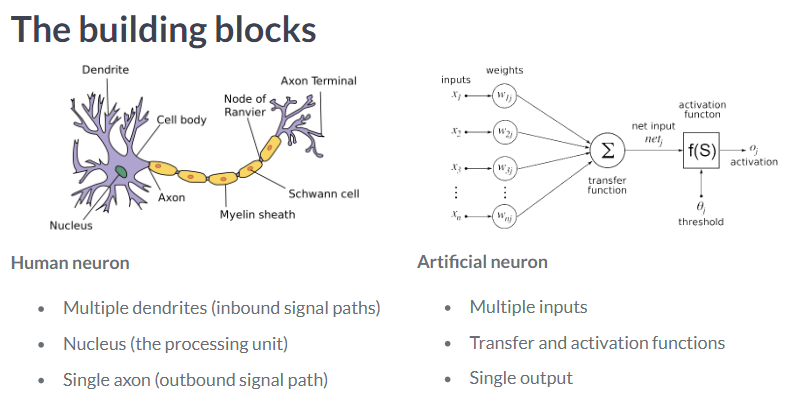
accuracy: 79.74 %

precision: 65.71 %

recall: 46.73 %

Good job, but what does this mean? When your model flags a client as a churner, it is correct in 67% of cases (precision) and you are able to detect 43% of the churners (recall) -- these are not the same thing and it's very important to understand the difference. All in all, not a great performance, but not a bad one either. In any case, you now understand the core end-to-end flow of building Supervised Learning algorithms!







# The nucleus

Artificial neural networks are built from artificial neurons.

Just like human neurons, artificial neurons have input pathways, bringing the signals to the neuron, and output pathways, leading the resulting signal further.

The values of the inputs signals have been aggregated (usually just summed up), then the final output value of the neuron is defined by a function.

What is this function called?

##### Answer the question

**50 XP**

##### Possible Answers

* 

Transfer function.

press1

* 

Activation function. **(A)**

press2

* 

Input function.

Press

Correct! The activation function is usually a non-linear function, like a sigmoid or a rectified linear function.

# How to train your dragon

For a long time, neural networks were more of a theoretical concept than a practical tool. The reason for this was the lack of an efficient training algorithm.

This all changed in 1986 when a group of authors published an infamous paper. Which revolutionary algorithm did this paper further improved and popularized?

##### Answer the question

**50 XP**

##### Possible Answers

* 

Moonwalk algorithm.

press1

* 

Backpropagation algorithm. **(A)**

press2

* 

Back-to-the-future algorithm.

Press

Yes! The backpropagation algorithm gave a huge boost to the field of AI by radically improving the efficiency with which neural network models were trained.

# Your first neural net

All that buzz around AI and Deep Learning, but we already promised you it is not so scary when you take a look under the hood.

You're going to create your first neural network, to solve a classification task.

You are working with labelled sensor data which has:

* 16 input columns, representing 16 features coming from a movement sensor of a smartwatch.
* a target column representing one of 4 movement types: walking, running, sitting, lying.

All the ingredients have been loaded for you: the Sequential() model and the Dense() layer -- you just have to put the ingredients in the right order, just like stacking pancakes!

##### Instructions

**100 XP**

* Initialize a Sequential() network.
* Set a fully connected Dense() hidden layer with 8 units (neurons), making sure to specify the correct input size to match the dimensions of the input data.
* Set another fully connected Dense() layer at the output, making sure to specify the correct number of output units, defined by the number of output classes of your problem.
* Compile the model!

# Initialize the model

model = Sequential()

# Add the hidden and the output layer, specify the layer type, number of units and input/output dimensions

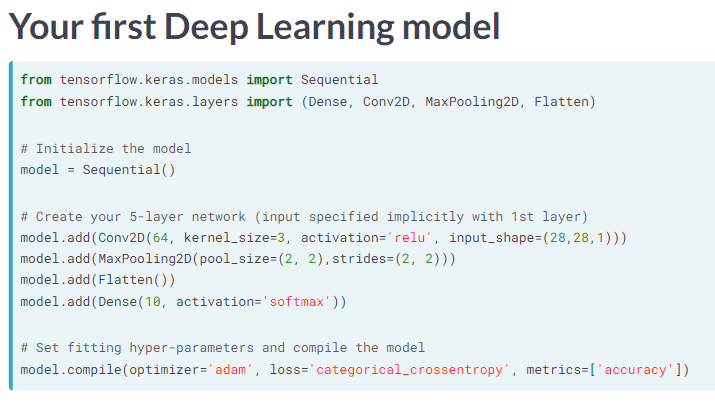
model.add(Dense(units= 8, input\_dim=16 , activation='relu'))

model.add(Dense(units= 4, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

Well done! How about that? Already making your first neural networks, are you?



# Layers

There are many types of layers in Deep Learning. One type in particular could be singled out and given credit for their rise in power and popularity, enabling Neural Nets to be come champions of image classification benchmarks.

Which one is it?

##### Answer the question

**50 XP**

##### Possible Answers

* 

Convolutional layer **(A)**

press1

* 

Dropout layer

press2

* 

MaxPooling layer

press3

* 

Flattening layer

Press

Yes! Convolutional layers, which are at the core of the Convolutional Neural Networks, were one of the biggest game changers in the evolution of Neural Network algorithms.

# Guess the architecture

Streamify, a new audio streaming platform, entered the market. From day one, they are running into an issue: with hundreds of new artists and songs uploaded daily, there is rarely enough time to appropriately classify the content into genres, which significantly hurts their users' experience.

Streamify wants you to train a Deep Learning model to perform this task automatically. You take up the challenge.

Since you're dealing with a gigantic amount of audio data, whose main dimension is time, which of the following networks is the perfect candidate to extract patterns?

##### Answer the question

50 XP

##### Possible Answers

* 

Convolutional neural network

press1

* 

Recurrent neural network **(A)**

press2

* 

Feedforward neural network

Exactly. Whenever you need to understand patterns in TIME, recurrent neural networks are the first logical choice.

# Rolling in the deep

You have been asked by the local police department to produce a Deep Learning model for license plate reading.

Your input data are images of digits, 28 pixels wide and 28 pixels tall, each with a label stating which of the 10 possible digits is present on the picture.

The Sequential() model is already loaded, which you will use to build a Deep Neural Network using the following layers:

* Conv2D() - 2D convolutional layer
* MaxPooling2D() - pooling layer
* Flatten() - flattening layer
* Dense() - fully connected layer

##### Instructions

100 XP

* Initialize the model and set a **2D convolutional layer** with 64 filters of size 3x3 at the input.
* Add a MaxPooling2D() layer, with default parameters, followed by a **flattening layer**, to reshape the signal from a 2-dimensional to a 1-dimensional format.
* Add a fully connected Dense() layer with a softmax activation function and 10 neurons for 10 target classes present in our training set.
* Compile the model.

# Initialize the model

model = Sequential()

# Create your 5-layer network (input specified implicitly with 1st layer)

model.add(Conv2D(filters=64, kernel\_size=(3 , 3), input\_shape=(28 , 28, 1)))

model.add(MaxPooling2D())

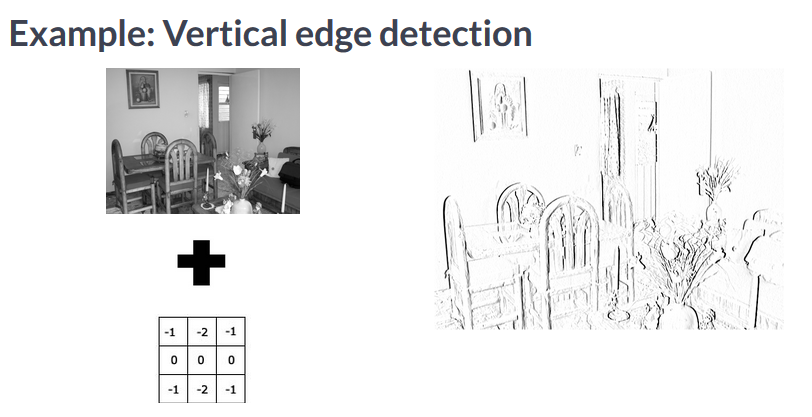
model.add(Flatten())

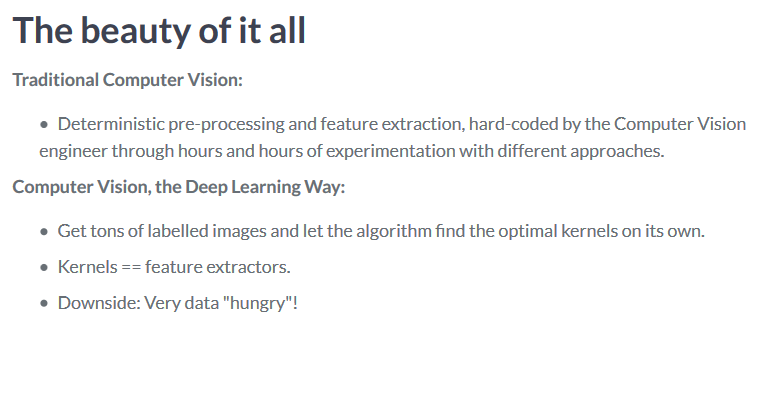
model.add(Dense(units=10, activation='softmax'))

# Set fitting hyper-parameters and compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

Great job! So, you see, when you know exactly what you need to do, you can construct a very powerful Deep Neural Network in less than 10 lines of code. That's how far the ecosystem of AI frameworks has gone and the progress is not slowing down.





# The Beauty of the Beast

Convolutional Neural Networks have come to dominate the domain of Computer Vision. What is the key reason, and fundamental difference compared to traditional Computer Vision algorithms, why this is the case?

##### Answer the question

50 XP

##### Possible Answers

* 

CNNs autonomously learn optimal feature transformations leading to best possible performance.

**(A)**

press1

* 

CNNs convert low-resolution images to high-resolution ones.

press2

* 

CNNs learn from a small amount of data.

# One-liner modeling

Modern Deep Learning libraries already help you by abstracting an ever bigger part of work that was once required to build neural networks.

As you could already see, in just 10 lines of code you can specify an extremely powerful network that could be trained for days, using terabytes of data.

But sometimes you want to go even further into abstraction: when you have a typical network specification that you often use, changing only a handful of parameters, it's a good idea to enclose it in a function.

That's what you will do in this exercise, and later reuse the created function for further exercises.

##### Instructions

100 XP

* Specify the default number of kernels as 32.
* Specify the default kernel size as (3, 3).
* Fix the optimizer type as "adam".
* Set "accuracy" as the optimization metric.

def make\_deep\_net(input\_shape, n\_output\_classes, n\_kernels=(3,3), kernel\_size= (3,3)):

# Initialize the sequential model

model = Sequential()

# Add the convolutional layer (containing implicitly the input layer)

model.add(Conv2D(input\_shape=input\_shape, filters=n\_kernels, kernel\_size=kernel\_size, activation='relu'))

# Add the flattening layer

model.add(Flatten())

# Add the fully connected layer

model.add(Dense(n\_output\_classes, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', metrics=['accuracy'], loss='categorical\_crossentropy')

return model

Good job defining the make\_deep\_net() function! It's going to make it easier to build larger networks in the coming exercises.`

# One-line evaluation

Although constant progress is being made, not all outputs of your modeling and evaluation algorithms come in a human readable shape.

Very often they come in plain arrays of unnamed values, requiring you to read through the documentation in order to interpret what each value means.

In such cases it is again clever to make wrapper functions that format and return the results in a neat and understandable way.

The evaluation result, stored in the variable score, is an array whose first element is the loss and the second accuracy of the model over the given dataset.

##### Instructions

100 XP

* Place the input variables x\_test and y\_test into the appropriate arguments of the model's .evaluate() method.
* Use the first element of the score array to print out the loss and the second element to print out the resulting accuracy.

def evaluate\_deep\_net(model, x\_test, y\_test):

# Generate the test predictions and evaluate against the ground truth

score = model.evaluate(x=x\_test, y=y\_test)

# Print the evaluation results in a human readable form

print('Test loss: %.2f' % score[0])

print('Test accuracy: %.2f %%' % (100\*score[1]))

That's it! Making convenient wrapper functions for 'humanizing' the outputs of your model is a very good practice that will make your work with building and evaluating Machine Learning models much more pleasureable.

# Deep Learning for Digit Recognition

Deep Learning models excel at classifying unstructured data, such as images and text. Common problems solved by Deep Learning include image classification, object detection, text translation, and text summarization.

In this exercise, you will use the functions defined in previous exercises (make\_deep\_net() and evaluate\_deep\_net()) to build a Deep Neural Network for recognizing hand-written digits.

You will train and test your model using the well known MNIST dataset, which contains a collection of images of individual hand-written digits, each 28x28 pixels big.

The dataset is pre-loaded and split into the training and test sets:(x\_train, y\_train) and (x\_test, y\_test).

##### Instructions 1/4

25 XP

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* [4](javascript:void(0))
* Construct the neural network, taking into consideration the input data dimensions and the number of output classes.

# Construct the Deep Neural Network

deep\_net = make\_deep\_net(input\_shape=[28, 28, 1],

n\_output\_classes=10)

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz

8192/11490434 [..............................] - ETA: 0s

3301376/11490434 [=======>......................] - ETA: 0s

4866048/11490434 [===========>..................] - ETA: 0s

10723328/11490434 [==========================>...] - ETA: 0s

11493376/11490434 [==============================] - 0s 0us/step

2000 train samples

333 test samples

2000 train samples

333 test samples

##### Instructions 2/4

25 XP

* [2](javascript:void(0))
* [3](javascript:void(0))
* [4](javascript:void(0))
* Call the fitting function of the model. Set the model training and validation data, and use batch\_size=128 and only **one training epoch** during the fitting.

# Construct the Deep Neural Network

deep\_net = make\_deep\_net(input\_shape=[28, 28, 1],

n\_output\_classes=10)

# Train the Deep Neural Network

deep\_net.fit(x=x\_train, y=y\_train,

validation\_data=(x\_test, y\_test),

batch\_size=128,

epochs=1)

<script.py> output:

Train on 2000 samples, validate on 333 samples

128/2000 [>.............................] - ETA: 3s - loss: 2.3086 - accuracy: 0.0938

384/2000 [====>.........................] - ETA: 1s - loss: 2.1644 - accuracy: 0.4062

512/2000 [======>.......................] - ETA: 1s - loss: 2.1120 - accuracy: 0.4531

640/2000 [========>.....................] - ETA: 0s - loss: 2.0604 - accuracy: 0.4797

768/2000 [==========>...................] - ETA: 0s - loss: 2.0006 - accuracy: 0.5143

896/2000 [============>.................] - ETA: 0s - loss: 1.9344 - accuracy: 0.5446

1024/2000 [==============>...............] - ETA: 0s - loss: 1.8741 - accuracy: 0.5596

1152/2000 [================>.............] - ETA: 0s - loss: 1.8157 - accuracy: 0.5833

1280/2000 [==================>...........] - ETA: 0s - loss: 1.7547 - accuracy: 0.6039

1408/2000 [====================>.........] - ETA: 0s - loss: 1.7051 - accuracy: 0.6158

1536/2000 [======================>.......] - ETA: 0s - loss: 1.6472 - accuracy: 0.6354

1664/2000 [=======================>......] - ETA: 0s - loss: 1.5922 - accuracy: 0.6484

1920/2000 [===========================>..] - ETA: 0s - loss: 1.4880 - accuracy: 0.6729

2000/2000 [==============================] - 2s 793us/sample - loss: 1.4607 - accuracy: 0.6780 - val\_loss: 0.8240 - val\_accuracy: 0.7958

2000 train samples

333 test samples

* Evaluate model performance.

# Construct the Deep Neural Network

deep\_net = make\_deep\_net(input\_shape=[28, 28, 1],

n\_output\_classes=10)

# Train the Deep Neural Network

deep\_net.fit(x=x\_train, y=y\_train,

validation\_data=(x\_test, y\_test),

batch\_size=128,

epochs=1)

# Estimate the network performance

evaluate\_deep\_net(deep\_net, x=x\_test, y=y\_test)

<script.py> output:

Train on 2000 samples, validate on 333 samples

128/2000 [>.............................] - ETA: 2s - loss: 2.3031 - accuracy: 0.1250

512/2000 [======>.......................] - ETA: 0s - loss: 2.0577 - accuracy: 0.4922

896/2000 [============>.................] - ETA: 0s - loss: 1.8496 - accuracy: 0.5770

1280/2000 [==================>...........] - ETA: 0s - loss: 1.6443 - accuracy: 0.6336

1664/2000 [=======================>......] - ETA: 0s - loss: 1.4733 - accuracy: 0.6737

2000/2000 [==============================] - 1s 378us/sample - loss: 1.3432 - accuracy: 0.7020 - val\_loss: 0.7299 - val\_accuracy: 0.7898

32/333 [=>............................] - ETA: 0s - loss: 0.6142 - accuracy: 0.7812

333/333 [==============================] - 0s 106us/sample - loss: 0.7299 - accuracy: 0.7898

Test loss: 0.73

Test accuracy: 78.98 %

2000 train samples

333 test samples

##### Instructions 4/4

25 XP

* [4](javascript:void(0))
* Compare the change in model performance if you go from one to **three training epochs**.

# Construct the Deep Neural Network

deep\_net = make\_deep\_net(input\_shape=[28, 28, 1],

n\_output\_classes=10)

# Train the Deep Neural Network

deep\_net.fit(x=x\_train, y=y\_train,

validation\_data=(x\_test, y\_test),

batch\_size=128,

epochs=3)

# Estimate the network performance

evaluate\_deep\_net(deep\_net, x=x\_test, y=y\_test)

<script.py> output:

Train on 2000 samples, validate on 333 samples

Epoch 1/3

128/2000 [>.............................] - ETA: 2s - loss: 2.3070 - accuracy: 0.0703

384/2000 [====>.........................] - ETA: 1s - loss: 2.1187 - accuracy: 0.4089

640/2000 [========>.....................] - ETA: 0s - loss: 1.9870 - accuracy: 0.4938

896/2000 [============>.................] - ETA: 0s - loss: 1.8396 - accuracy: 0.5603

1152/2000 [================>.............] - ETA: 0s - loss: 1.7028 - accuracy: 0.6024

1408/2000 [====================>.........] - ETA: 0s - loss: 1.5846 - accuracy: 0.6307

1792/2000 [=========================>....] - ETA: 0s - loss: 1.4145 - accuracy: 0.6747

2000/2000 [==============================] - 1s 463us/sample - loss: 1.3388 - accuracy: 0.6895 - val\_loss: 0.7281 - val\_accuracy: 0.7868

Epoch 2/3

128/2000 [>.............................] - ETA: 0s - loss: 0.5764 - accuracy: 0.8672

384/2000 [====>.........................] - ETA: 0s - loss: 0.5698 - accuracy: 0.8542

768/2000 [==========>...................] - ETA: 0s - loss: 0.5422 - accuracy: 0.8659

1152/2000 [================>.............] - ETA: 0s - loss: 0.5194 - accuracy: 0.8689

1536/2000 [======================>.......] - ETA: 0s - loss: 0.4949 - accuracy: 0.8678

1664/2000 [=======================>......] - ETA: 0s - loss: 0.4871 - accuracy: 0.8690

1792/2000 [=========================>....] - ETA: 0s - loss: 0.4878 - accuracy: 0.8672

2000/2000 [==============================] - 1s 267us/sample - loss: 0.4903 - accuracy: 0.8655 - val\_loss: 0.4364 - val\_accuracy: 0.8679

Epoch 3/3

128/2000 [>.............................] - ETA: 0s - loss: 0.2819 - accuracy: 0.8906

512/2000 [======>.......................] - ETA: 0s - loss: 0.3748 - accuracy: 0.8809

896/2000 [============>.................] - ETA: 0s - loss: 0.3778 - accuracy: 0.8839

1280/2000 [==================>...........] - ETA: 0s - loss: 0.3406 - accuracy: 0.8969

1664/2000 [=======================>......] - ETA: 0s - loss: 0.3226 - accuracy: 0.9038

2000/2000 [==============================] - 0s 179us/sample - loss: 0.3418 - accuracy: 0.9005 - val\_loss: 0.3340 - val\_accuracy: 0.8979

32/333 [=>............................] - ETA: 0s - loss: 0.2035 - accuracy: 0.9688

333/333 [==============================] - 0s 100us/sample - loss: 0.3340 - accuracy: 0.8979

Test loss: 0.33

Test accuracy: 89.79 %

As expected. Starting from one and then gradually increasing the number of training epochs normally leads to better performance, but only up to a certain threshold. You must also keep in mind that training real-life models takes a lot of computing time and power, so you're looking for the number of training epoch that is just good enough, not perfect.