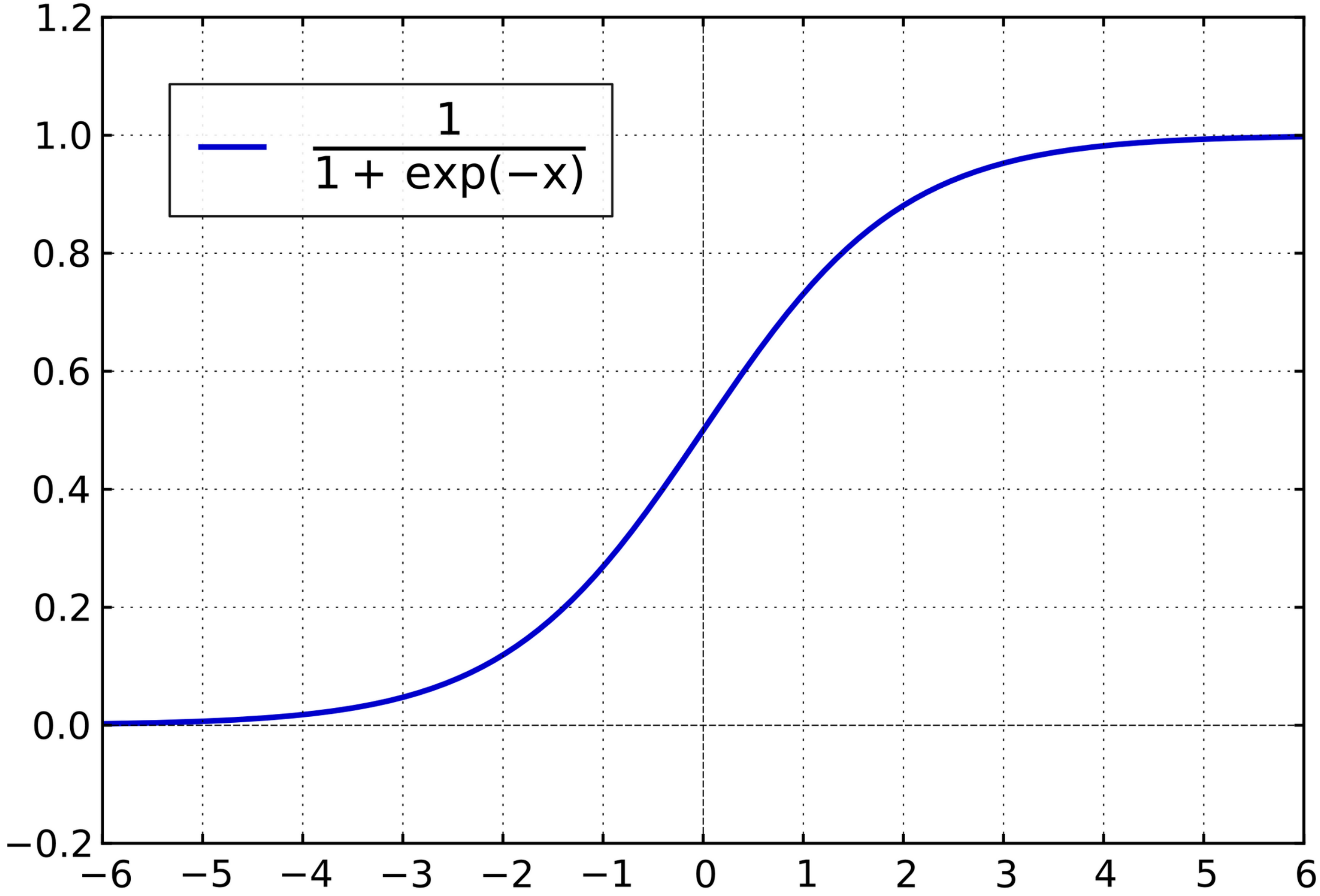
Logistic Regression

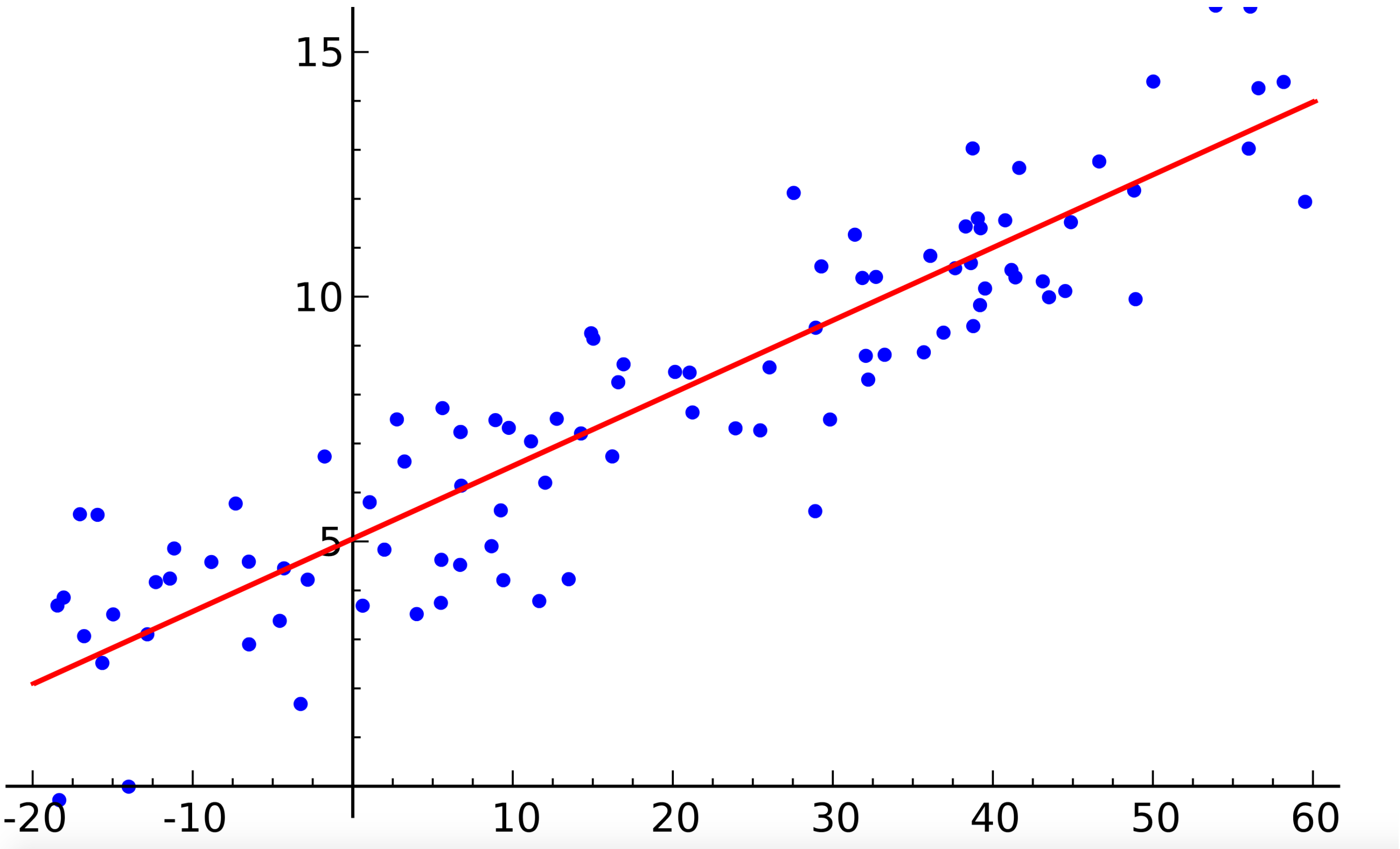
- classification algorithm
- statistical model widely used for predicting binary classes (2 possible outcomes: 0 and 1)
- Spam detection, detection of certain diseases ...
- computes the probability of an event occurrence using the logit/sigmoid function



1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−6
−5
−4
−3
−2
−1
0
1
2
3
4
5
6
$\frac{1}{1+\exp(-x)}$

Excursus Linear Regression

- algorithm used for regression problems (with a numeric outcome)
- simple linear regression: we try to find the best linear function ax + b fitting to our data
- a is our slope parameter, b our intercept (a and b = our "Theta"-weights, we try to optimize)



- logistic regression bases upon linear regression
- why do we not use linear regression for classification problems as well?
    1. only two possible outcomes: the linear slope doesn't map this very well
    2. output values bigger than 1 and smaller than 0 can occur)

In [10]:

```
# Logistic Regression Algorithm

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train,y_train)
print("Test Accuracy %.2f%%" % (lr.score(x_test,y_test)*100))
```

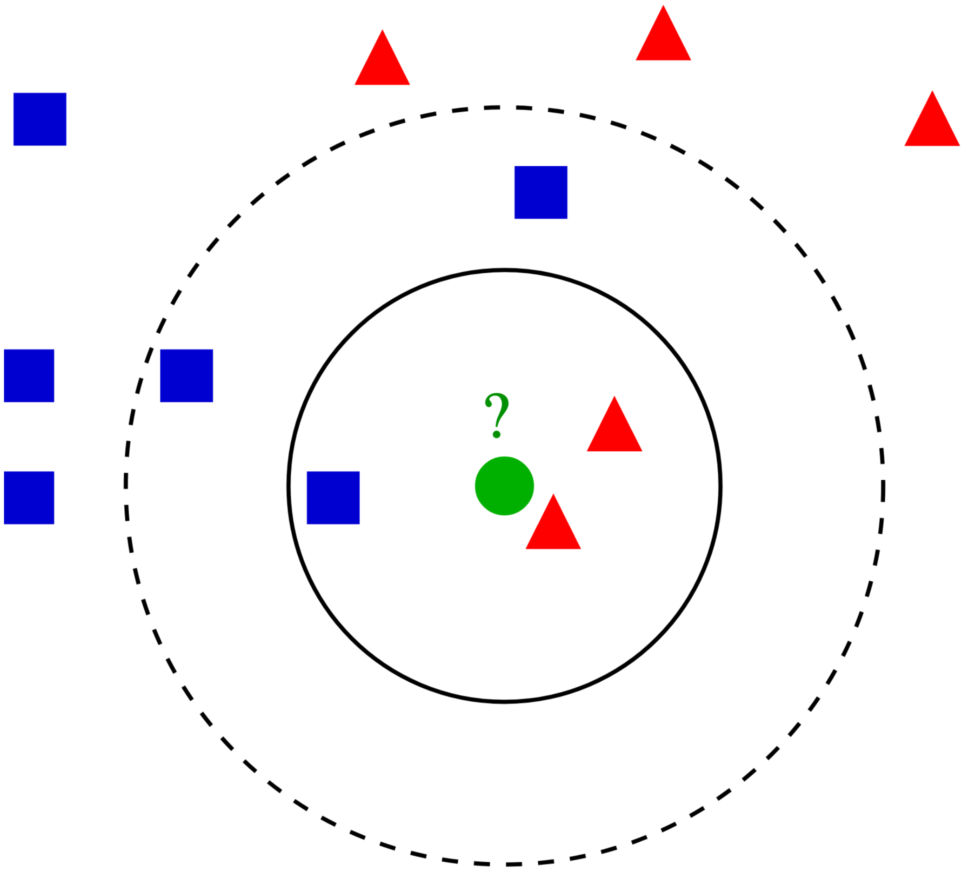
```
Test Accuracy 86.89%
```

In [11]:

```
prediction = lr.predict(x_test)
pred_prob = lr.predict_proba(x_test)

print("Predictions vs. actual Classes:")
print("Pred:  ", prediction[:20])
print("Class:", y_test[:20])
print("\nPrediction Probs: \n")
print(pred_prob[:20])
```

```
Predictions vs. actual Classes:
Pred:  [0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 0 1]
Class: [0 1 0 0 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 1]

Prediction Probs: 

[[0.88944519 0.11055481]
 [0.54319211 0.45680789]
 [0.52339135 0.47660865]
 [0.95430018 0.04569982]
 [0.83296797 0.16703203]
 [0.75869545 0.24130455]
 [0.91481407 0.08518593]
 [0.89223814 0.10776186]
 [0.95133633 0.04866367]
 [0.97491908 0.02508092]
 [0.36443036 0.63556964]
 [0.10378138 0.89621862]
 [0.92980393 0.07019607]
 [0.10644516 0.89355484]
 [0.0603269  0.9396731 ]
 [0.25492222 0.74507778]
 [0.89874266 0.10125734]
 [0.18347126 0.81652874]
 [0.97463813 0.02536187]
 [0.26027727 0.73972273]]
```

K-nearest Neighbor



- relatively simple and often used classification algorithm (can also be used for regression)
- we diagnose the class of a certain sample by looking at the closest known points (neighbors) around: their "majority class" will become our sample's class
- hyperparameter k := number of neighbors
- distance function, vote function
- dataset = (x, y), x := Features, y := classes
- in sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

KNeighborsClassifier

- class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)

Parameters

- `n_neighbors` : int, optional (default = 5). Number of neighbors to use by default for kneighbors queries
- `weights` : str or callable, optional (default = 'uniform').
    - uniform : uniform weights. All points in each neighborhood are weighted equally.
    - distance' : weight points by the inverse of their distance.

In [12]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

heartdisease = pd.read_csv("data/heart.csv")
```

Exercises K-nearest Neighbor

In [14]:

```
# Take two columns of your choice and assign them to a new dataframe x_train1 and
 x_test1
```

In [15]:

```
# KNN Algorithm
# import the KNeighborsClassifier and save two instances for your two dataframes i
n a variable "knn" and "knn1".
# We want to start with 10 neighbors.
```

In [16]:

```
# Train the model on your new smaller dataset
# How does it perform? Check the test data
```

In [17]:

```
# Train the model on your old dataset with all features
# How does it perform? Check the test data
```

In [18]:

```
# Compare prediction and actual classes for one of your trained models
```

In [19]:
```
# Bonus, if you are familiar with python:
# We try to find the best number of neighbors between 1 and 20.
# Realize it with a for-loop.
```