Cursurile + Seminariile Aici (moodle as guest)

Basics

Data sets:

- **Training Set:** Data we use for training the AI
- **Test Set:** Data we use for predictions

Supervised, Unsupervised, Reinforcement:

- **Supervised Learning:** Feeding labeled data to the AI. You (as a human) can probably already know the answers
- **Unsupervised Learning:** Feeding arbitrary, unlabeled data and letting the AI draw conclusions or find patterns and cluster them together
- **Reinforcement:** The Agent (AI character) performs actions, sees the results (gets a reward) and adjusts. Tries to maximize reward

Types of algorithms

- **Classification** = classification of data (or true/false output)
- **Regression** = predicting a value
- **Anomaly Detection** = obviously
- **Clustering** = finding patterns or grouping data

Variables:

- **Independent Variable** - doesn't depend on anything else
- **Dependent Variable** - depends on some other variable

Values:

- **Numerical Value** - From an interval (variabila continua)
- **Categorical Value** - A categorization (variabila aleatoare)

Statistics types:

- **descriptive** - describes the data
  - *central tendencies measures* = mean, median, etc
  - *variability measures* = spread, range, variance, StD, etc
- **inferential** - makes predictions - applies probability to draw a conclusion

Matrice Hessiana: matrice a derivatelor de ordinul doi pentru o functie Exemplu:

- $f(x, y) = 5x^2 + xy^3 - y^2$
- Calculam **df/dx** si **df/dy**

Matricea Hessiana, notata ∇f say f''(x,y), este:

```
d(df/dx) / dx   d(df/dx) / dy
d(df/dy) / dx   d(df/dy) / dy
```

Standard deviation:

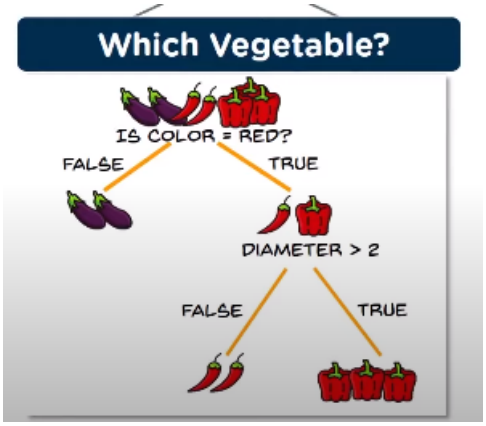
```
let dataset_mean = the average of dataset
rhs(x) = (x - dataset_mean)^2
StD = mean | dataset.map | x -> rhx(x)
```

Decision Tree

Suppose we have this dataset:

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

To create the tree, we split the table into 3 tables based on the **most significant variable**, Outlook: Sunny, Overcast and Rain. We chose Outlook because it's independent.

Another example: 

Entropy: impurity (uncertainty) in the dataset; low entropy means good classification **Information Gain:** decrease in entropy (which is good)

Entropy Calculation:

```
For each classifier (Giraffe, Elephant, Monkey or Cat), we get its probability.
Entropy = Sum | probability(animal type) * log2(probability(animal type))
```

To do the split:

```
Split the dataset by each property and calculate gain for each.
The split with the highest gain becomes the one.
```

Then, we do this all again for each category.

Note: this probably only works for discrete properties

Confusion Matrix: describes the performance of a model It calculates the accuracy of your prediction: compares actual results with predicted results.

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

- *True Positive* = yes and correct yes prediction
- *True Negative* = no and correct no prediction
- *False Positive* = no and yes prediction
- *False Negative* = yes and no prediction

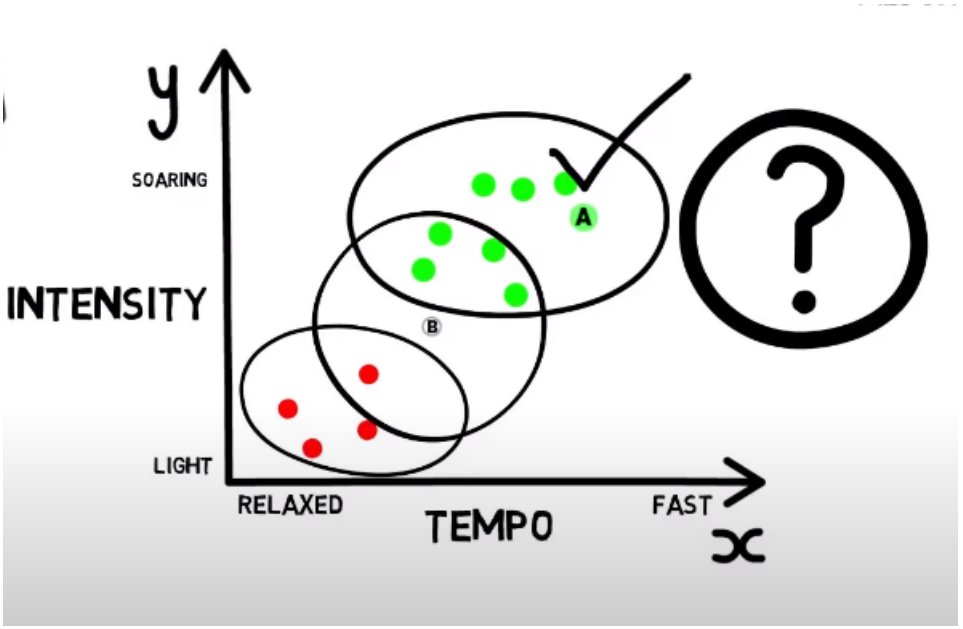
Note: it can have many row and columns; Predicted on one side and Actual on another.

Note: The Confusion Matrix is always square.

K-Nearest Neighbors

Simplest ML algorithm (supervised):

Based on our previous entries, if we have a new entry B, we draw a circle with a predefined radius and see more of which fit around it.



K-Means Clustering

Grouping together data, like red and green in the above example.

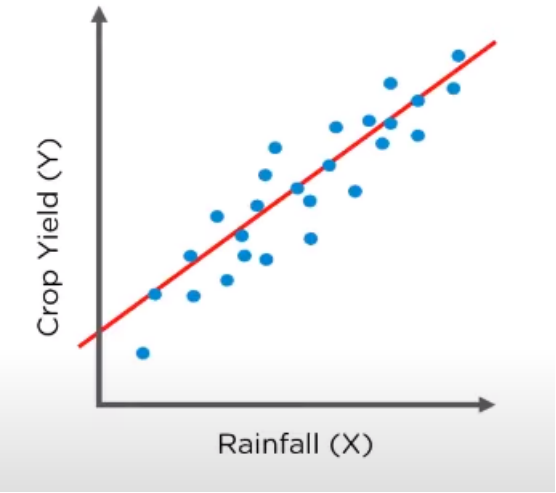
```
For each group, we pick a random point. These are called centeriods.
Start:
    For each point in our data set, we calculate its distance to each centeroid. We assign each of the data points to
    Reposition each centeroid to the actual center of its assigned data points.
    Unassign data points from centeroids, then go back to Start.
    (We stop if centeroids did not change)
```

Simple Linear Regression

It predicts a value based on another value based on a trained dataset. We have dependent and independent variables.

Ex: We have 2 variables: Rainfall (independent) and Crop Yield (dependent on Rain).

Draw the line: Should be the least total 'Y' distance from all points to the line y = m * x + c



Least Squares algorithm

Find the mean point (media aritmetica) = (MeanX, MeanY) We have the line equation MeanY = m * MeanX + c

```
For each point (x,y)
    let x' = x - MeanX
    let y' = y - MeanY
m = (Sum x' * y') / (Sum (x'^2))
c = m * MeanX / MeanY
```

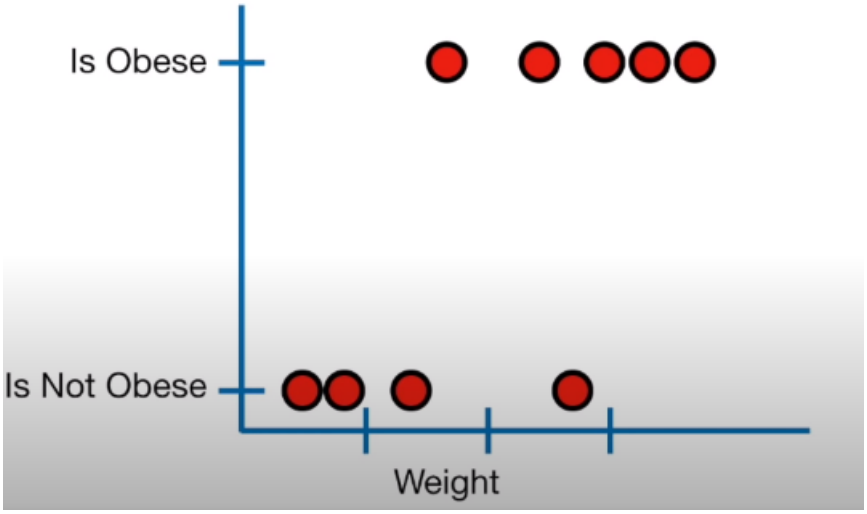
Residual Error (of a prediction) = 'Y' distance from a point to the line $\mathbf{R}^2$ = Variance explained by the model / Total variance - A percentage between 0 and 100% - 0% = does not

Multiple Linear Regression

Y = m1*x1* + *m2*x2 + ... + mn*xn + c

Logistic Regression

Predicts a binary classification based on another value (outputs yes or no). Still on an XY axis, but the shape of the line is curved like an S (Sigmoid shape)



Our plot will look like a _/‾ Based on the *weight* as in the example, we get a number between 0 and 1 (0 = not obese, 1 = obese). If > 0.5, then obese Else, not obese. Simple as that.

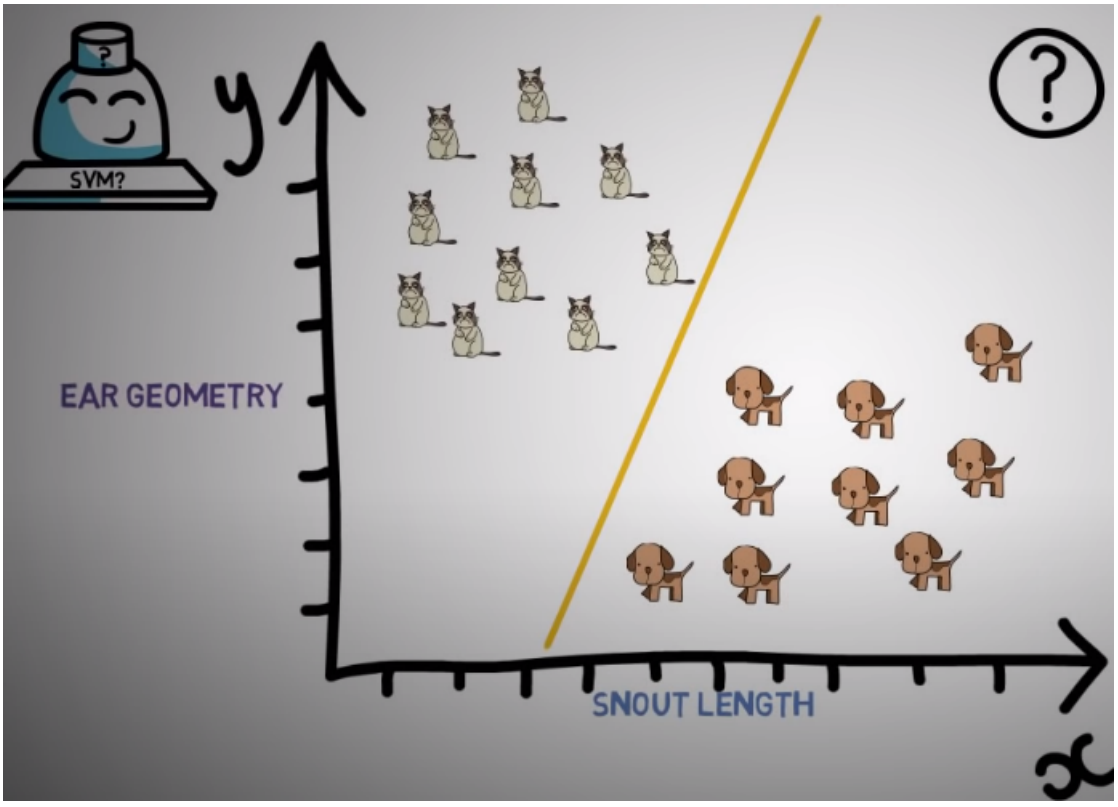
Odds(event) = Prob(event) / (1 - Prob(event))

- log(**Odds**) = m*x + c

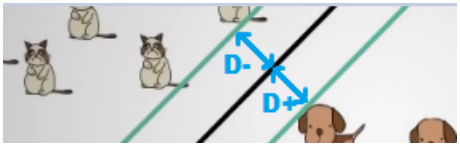
Support Vector Machines

Introducere: Articolul gaseste o rezolvare eficienta pentru

Support Vector machine: Algoritm care imparte datasetul in doua - trage o linie (hiperplan) intre date. Apoi, daca avem date noi, putem spune daca e cat sau dog - daca e la stanga sau la dreapta de hiperplan. Sunt mai multe posibilitati de a trage o astfel de linie.



support vector: drepte (hiperplane) paralalele cu linia din mijloc, care trec prin cele mai apropiate 2 puncte fata de linia din mijloc; sunt 2 support vectors - unul pe stanga, unul pe dreapta. Idea, dreapta noastra de pe mijloc ar trebui sa fie la distanta egala de cei doi support vectors.



D+ / D-: distantele de la support vectori la linia din mijloc

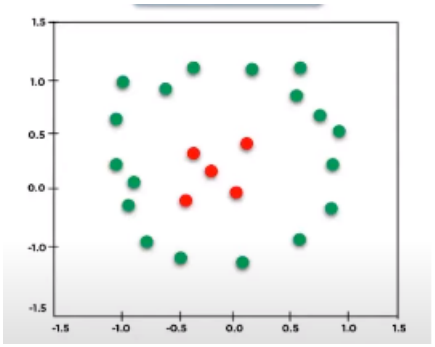
- **Ecuatia unei drepte:** $y = w * x + b$
- **Ecuatia unui hiperplan:** $y = w_1x_1 + w_2x_2 + ... + w_nx_n + b$

Ideal, un SVM incearca sa obtina cea mai mare distanta intre cele doua grupuri de date

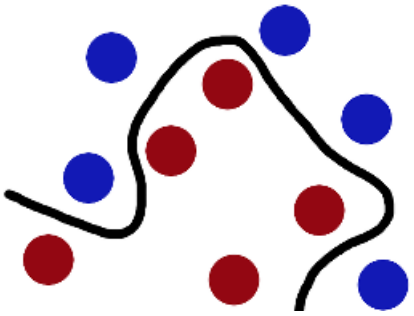
Dar daca setul nostru de date are o singura dimensiune?

Folosim un Kernel Function (Kernel Trick) ca sa transformam setul de date din 1D in 2D adaugand o coordonata

Dar daca setul nostru de date are o forma neobisnuita si nu putem trasa o linie ca sa le separam?



or



Transformam setul din 2D in 3D si folosim un plan in loc de linie.

Kernel Functions (Tricks)

Depending on what our data looks like, we will use different kernel functions:

Our separation line is curved -> K(x, y) = (x, y, x * y)

Gradient Descent

Optimization technique for Linear Regression, Logistics Regression, anything really

For a 2D graphic, let's say we want to calculate y knowing x. We know y = mx + c. We calculate m with Least Squares:

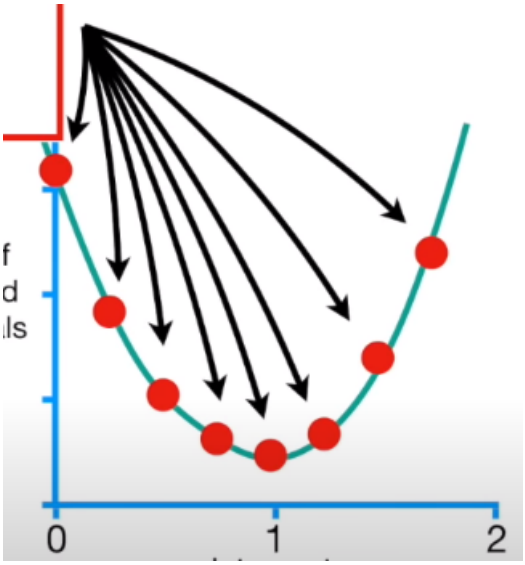
```
For each point (x,y)
    let x' = x - MeanX
    let y' = y - MeanY
m = (Sum x' * y') / (Sum (x'^2))
```

How do we calculate c though so it's the best? We can do a brute force approach, but it's terribly slow. So we use Gradient Descent:

```
c = random value (let's say 0)
Res(x) = residual error = actual y - predicted y
Ry(c) = Sum | dataset.map | x -> (Res(x) - c - m*x)^2
Adica:  (Res(x1) - c - m*x1)^2 +
```

```
        (Res(x2) - c - m*x2)^2 +
        (Res(x3) - c - m*x3)^2 + ...
```

Since we know x1, x2, ..., the only variable in this equation is **c**. Now we have an equation for the curve! Each point is at (c, Ry(c)). It looks like this:



Next, we calculate the derivative of Ry(c):

```
Ry'(c) = Sum | dataset.map | x -> (-2) (Res(x) - c - m*x)
```

Gradient descent will use this equation to find where the sum of squared residuals is lowest on the curve. Here's how it's done:

```
c = random number
StepSize = Ry(c) / 10
if StepSize <= 0.001 then stop
c = c - StepSize
```

Now we have an optimal **c** for our original y = m*x + c

Proiect

fie Pairs = un set de **(x, y)**, unde y este un clasificator (ex: **caine** sau **pisica**)

SVM are nevoie de solutia pentru problema de optimizare *unconstrained* urmatoare:

Problema min w Primal Form (1)

Problema: $w^T w$ / 2 + C * sum(Losses)

Losses = aplicam E(w; x, y) pe fiecare element din Pairs

E(w; x, y) = loss function Doua *loss functions* comune sunt:

- **L1-SVM** = max (1 - y * w^T * x, 0)
- **L2-SVM** = **L1-SVM**^{2}

Problema de mai devreme are si o forma *duala*:

Problema min a Dual Form (2)

Problema: $a^T Q' a / 2 - e^T a = f(a)$

- orice element din **a** is in [0, U]
- Q' = Q + D
- **L1-SVM**: U = C si $D_{ii} = 0$
- **L2-SVM**: U = infinit si $D_{ii} = 1/(2C)$

Din cauza ca **w** are dimensiuni foarte mari, problema **Problema min a Dual Form (2)** se rezolva cu un *kernel trick* (ex: folosind un *closed form* $Phi(x_i)^T Phi(x_j)$; Problema aceasta este **SVM nonlinear**.

Daca datele nu sunt *mapate*, putem antrena algoritmul cu mai multe date. Cazul acesta foloseste **SVM linear**.

Articolul rezolva probleme pentru **SVM lineare** foarte mari.

2. A Dual Coordinate Descent Method

Ce inteleg eu:

- Dual coordinate = pe doua coordonate (axele XY)
- Descent = Pentru ecuatia y = mx + c, se incearca sa se gaseasca cel mai bun **c** astfel incat distanta de la dreapta de separare la puncte este cat mai egal distribuita intre toate punctele (adica fix pe *mijloc*).
    - Se numeste Descent pentru ca acest **c** se situeaza pe o curba convexa in forma de **U**; pe axa X avem **c** iar pe axa Y avem Ry(c) ca in sectiunea de mai sus. Trebuie gasit cel mai de jos Ry(c) pe grafic, si astfel pe **c**.