L1: Entropy is a measure of the purity of a dataset (interval) S The higher the entropy, the lower the purity of the dataset

$$entropy(S) = -\sum_i P_i . \log_2 P_i$$ Pi - proportion of examples from class i

- Ex.: Consider a split between 70 and 71. What is the entropy of the left and right datasets (intervals)?
- values of temperature:

64	65	68	69	70	71	72	73	74	75	80	81	83	85
yes	no	yes	yes	yes	no	no	no	yes	yes	no	yes	yes	no

$$entropy(S_{left}) = -\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5} = 0.722\,bits$$

$$entropy(S_{right}) = -\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} = 0.991\,bits$$

Total entropy of the split = weighted average of the interval entropies

$$totalEntropy = \sum_i^n w_i\, entropy(S_i)$$

w_i – proportion of values in interval *i*, *n* – number of intervals

Algorithm: evaluate all possible splits and choose the best one (with the lowest total entropy); repeat recursively until stopping criteria are satisfied (e.g. user specified number of splits is reached)

Normalization and standardization

Performed for each attribute

Normalization (also called min-max scaling):

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Standardization:

$$x' = \frac{x - \mu(x)}{\sigma(x)}$$

x – original value
x' – new value

x – all values of the attribute; a vector
min(x) and max(x) – min and max values of the attribute (of the vector x)
μ(x) - mean value of the attribute
σ(x) - standard deviation of the attribute

Euclidean and Manhattan distance

Distance measures for numeric attributes

- A, B – examples with attribute values $a_1, a_2, \ldots, a_n$ & $b_1, b_2, \ldots, b_n$
- E.g. A= [1, 3, 5], B=[1, 6, 9]

Euclidean distance (L2 norm) – most frequently used

$$D(A,B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}$$

D(A,B) = sqrt ((1-1)^2+(3-6)^2+(5-9)^2)=5

Manhattan distance (L1 norm)

$$D(A,B) = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$

D(A,B)=|1-1|+|3-6|+|5-9|=7

Weighted distance – each attribute is assigned a weight according to its importance (requires domain knowledge)

- Weighted Euclidean:

$$D(A,B) = \sqrt{w_1|a_1 - b_1|^2 + w_2|a_2 - b_2|^2 + \ldots + w_n|a_n - b_n|^2}$$

Hamming distance = Manhattan for binary vectors

- Counts the number of different bits

$$D(A,B) = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$

A = [1 0 0 0 0 0 0 0 0 0]
B = [0 0 0 0 0 0 1 0 0 1]
D(A,B) = 3

Similarity coefficients
f00: number of matching 0-0 bits
f01: number of matching 0-1 bits
f10: number of matching 1-0 bits
f11: number of matching 1-1 bits

Calculate these coefficients for the example above!
Answer: f01 = 2, f10 = 1, f00 = 7 , f11 = 0

Minkowski distance – generalization of Euclidean & Manhattan

$$D(A,B) = (|a_1 - b_1|^q + |a_2 - b_2|^q + \ldots + |a_n - b_n|^q)^{1/q}$$

q – positive integer

Simple Matching Coefficient (SMC) - matching 1-1 and 0-0 / num. attributes
SMC = (f11+f00)/(f01+f10+f11+f00)
Ex.: A = [1 0 0 0 0 0 0 0 0 0]
B = [0 0 0 0 0 0 1 0 0 1]
f01 = 2, f10 = 1, f00 = 7 , f11 = 0
SMC = (0+7) / (2+1+0+7) = 0.7

An alternative: Jaccard coefficient

- counts matching 1-1 and ignores matching 0-0

J=f11/(f01+f10+f11)

A = [1 0 0 0 0 0 0 0 0 0]
B = [0 0 0 0 0 0 1 0 0 1]
f01 = 2, f10 = 1, f00 = 7 , f11 = 0
J = 0 / (2 + 1 + 0) = 0 (A and B are dissimilar)

Useful for sparse data (both binary and non-binary)
Widely used for classification of text documents

$$\cos(A,B) = \frac{A \bullet B}{\|A\|\|B\|}$$

- $\bullet$ - vector dot product, ||*A*|| - length of vector *A*

Geometric representation: measures the angle between A and B

- Cosine similarity=1 => angle(A,B)=0º
- Cosine similarity =0 => angle (A,B)=90º

Two document vectors:
d_1 = 3 2 0 5 0 0 0 2 0 0
d_2 = 1 0 0 0 0 0 0 1 0 2

$$\cos(A,B) = \frac{A \bullet B}{\|A\|\|B\|}$$

$d_1 \bullet d_2$= 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5
$||d_1||$ = (3*3+2*2+0*0+5*5+0*0+0*0+0*0+2*2+0*0+0*0)$^{1/2}$ = (42) $^{1/2}$ = 6.481
$||d_2||$ = (1*1+0*0+0*0+0*0+0*0+0*0+0*0+1*1+0*0+2*2) $^{1/2}$ = (6) $^{1/2}$ = 2.245
=> cos(d_1, d_2) = 0.3150

Pearson correlation coefficient between data objects (instances) x and y with dimensionality *n*

$$\text{corr}(\mathbf{x},\mathbf{y}) = \frac{\text{covar}(\mathbf{x},\mathbf{y})}{\text{std}(\mathbf{x})\text{std}(\mathbf{y})}$$

where:

$$mean(\mathbf{x}) = \frac{\sum_{k=1}^n x_k}{n} \qquad std(\mathbf{x}) = \sqrt{\frac{\sum_{k=1}^n (x_k - mean(\mathbf{x}))^2}{n-1}}$$

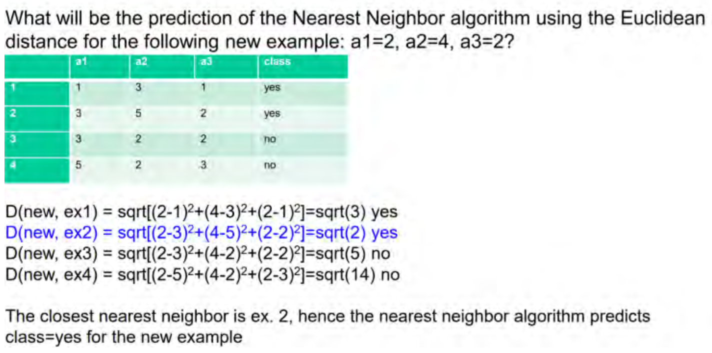
$$\text{co var}(\mathbf{x},\mathbf{y}) = \frac{1}{n-1}\sum_{k=1}^n (x_k - mean(x))(y_k - mean(y))$$

Range: [-1, 1]

- -1: perfect negative correlation
- +1: perfect positive correlation
- 0: no correlation

L2 KNN:

•categorical (nominal) - their values belong to a pre-specified, finite set of possibilities
• numeric (continuous) - their values are numbers


What will be the prediction of the Nearest Neighbor algorithm using the Euclidean distance for the following new example: a1=2, a2=4, a3=2?

	a1	a2	a3	class
1	1	3	1	yes
2	3	5	2	yes
3	3	2	2	no
4	5	2	3	no

D(new, ex1) = sqrt[(2-1)^2+(4-3)^2+(2-1)^2]=sqrt(3) yes
D(new, ex2) = sqrt[(2-3)^2+(4-5)^2+(2-2)^2]=sqrt(2) yes
D(new, ex3) = sqrt[(2-3)^2+(4-2)^2+(2-2)^2]=sqrt(5) no
D(new, ex4) = sqrt[(2-5)^2+(4-2)^2+(2-3)^2]=sqrt(14) no

The closest nearest neighbor is ex. 2, hence the nearest neighbor algorithm predicts class=yes for the new example

Training
Classification (prediction for a new example)
• Compare each unseen example with each training example
• If m training examples with dimensionality n => lookup for 1 unseen example takes m*n computations, i.e. O(mn)
Variations more efficiently: KD-trees & ball trees

Choice of k
K-Nearest Neighbor is very sensitive to to the value of k
• rule of thumb: k ≤ sqrt(#training_examples)
• commercial packages typically use k=10
•more nearest neighbors increases the robustness to noisy examples
also for **regression** : average value of the class values (numerical) of the k nearest neighbours

Nominal Data:
difference = 0 if attribute values are the same
difference = 1 if they are not

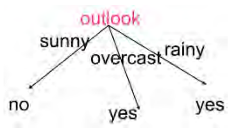
Example: 2 attributes = temperature and windy
temperature values: low and high **windy** values: yes and no ex.1= {high, no} ex.2 = {high, yes} d(A,B) =$(0+1)^{1/2}$=1 (Euclidean distance)

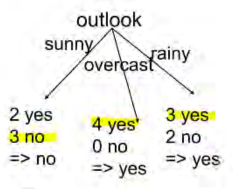
Weighted nearest neighbor
Idea: Closer neighbors should count more than distant neighbors
• Distance-weighted nearest-neighbor algorithm
• Find the k nearest neighbors
• Weight their contribution based on their distance to the new example
• bigger weight if they are closer
• smaller weight if they are further
• e.g. the vote can be weighted according to the distance – weight w = 1/distance^2

Decision boundary: Each training example has an associated **Voronoi region**; it contains the data points for which this is the closest example

Discussion:
- Often very accurate
- Slow for big datasets
- Distance-based - **requires normalization**
- Not effective for high-dimensional data (data with many features) -Solution – dimensionality reduction and feature selection
- Sensitive to the value of k

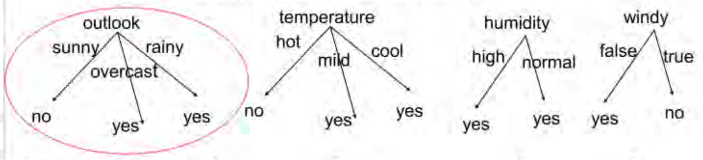
1-rule
Generates 1 rule that tests the value of a single attribute





Which is the **best rule (i.e. the best attribute)**?
• The one with the **smallest error rate** (i.e. with the highest accuracy) on training data

- 1R algorithm
  - Generate a rule (decision stump) for each attribute
  - Evaluate each rule on the training data and calculate the number of errors
  - Choose the one with the smallest number of errors



Errors on training data: 4 5 4 5

Final rule - rule 1:
if outlook=sunny then play=no
elseif outlook=overcast then play=yes
elseif outlook=rainy then play=yes

1R – discussion
• Simple and efficient algorithm, easy to understand
• Numerical datasets require discretization
• 1R has an in-built procedure to do this

Rule-Based Algorithms: PRISM - rule-based covering algorithm – Accuracy on training data always 100%
consider each class in turn and
• construct a set of if-then rules that cover all examples from this class and do not cover any examples from the other classes

Which test to add at each step?
The one that maximizes accuracy p/t:
• t: total number of examples (from all classes) covered by the rule (t comes from total)
• p: examples from the class under consideration, covered by the rule (p comes from positive)
• t-p: number of errors made by the rule
• Select the test that maximises the accuracy p/t

- Start with an empty rule: `if ? then recommendation = hard`
- 9 possible tests for the 4 attributes based on num. attribute values (3+2+2+2):

p/t (accuracy)

`age = young`	2/8
`age= pre-presbyoptic`	1/8
`age = presbyoptic`	1/8
`spectacle prescription = myope`	3/12
`spectacle prescription = hypermetrope`	1/12
`astigmatism = no`	0/12
`astigmatism = yes`	**4/12**
`tear production rate = reduced`	0/12
`tear production rate = normal`	4/12

age=young in 8 ex. and in 2 of them class=hard

- Best test (highest accuracy): `astigmatism = yes`
- Note that there is a tie: both `astigmatism = yes` and `tear production rate = normal` have the same accuracy 4/12; we choose the first one randomly

Current rule

`if astigmatism = yes then recommendation = hard`

Not "perfect" - covers 12 examples but only 4 of them are from class `hard` => refinement is needed

age	spectacle prescription	astigmatism	tear production rate	recommended lenses
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

- Further refinement by adding tests:

  `if astigmatism = yes and ? then recommendation = hard`
- Possible tests:

`age = young`	2/4
`age= pre-presbyoptic`	1/4
`age = presbyoptic`	1/4
`spectacle prescription = myope`	3/6
`spectacle prescription = hypermetrope`	1/6
`tear production rate = reduced`	0/6
`tear production rate = normal`	**4/6**

- Best test: `tear production rate = normal`

Current rule:

`if astigmatism = yes & tear production = normal then recommendation = hard`

Examples covered by the rule

age	spectacle prescription	astigmatism	tear production rate	recommended lenses
young	myope	yes	normal	hard
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	yes	normal	none

The rule is again not "perfect" – 2 examples classified as `none` => further refinement is needed

Further refinement:

`if astigmatism = yes & tear production = normal and ? then recommendation = hard`

Possible tests

`age = young`	2/2
`age= pre-presbyoptic`	1/2
`age = presbyoptic`	1/2
`spectacle prescription = myope`	**3/3**
`spectacle prescription = hypermetrope`	1/3

Best test: tie between the 1st and 4th; choose the one with the greater coverage (4th)

New rule:

`if astigmatism = yes & tear production = normal & spectacle prescription = myope then recommendation = hard`

Does the rule cover all hard examples? No, only 3/4, so we will need another rule

- Delete these 3 examples and start again
- Stop as all examples from class hard are covered

- Follow the same process for the other 2 classes (soft and none)

L3 Linear Regression :

Prediction error/residual

- Prediction error (residual) = observed-predicted value = $= \varepsilon = y_i - \hat{y}_i$

Performance index: sum of squared prediction errors (SSE): $SSE = \sum_i (y_i - \hat{y}_i)^2$

Our goal: select the line which minimizes SSE

- Can be solved using the **method of the least Squares**

The least squares method finds the best fit to the data but doesn't tell us how good this fit it

- E.g. SSE=12; is this large or small?

R^2 measures the *goodness of fit* of the regression line found by the least squares method:

$$R^2 = \frac{SSR}{SST}$$

Values between 0 and 1; the higher the better

- = 1: the regression line fits perfectly the training data
- close to 0: poor fit

SST = SSR + SSE

SSE: sum of squared prediction errors (actual – predict)

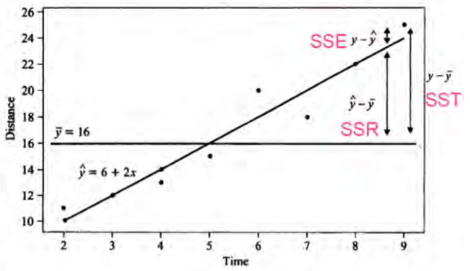
SST: sum of squared total errors (actual – mean)

$$SST = \sum_{i=1}^{n} (y_i - \bar{y})^2$$ = actual value – mean value

$$SST = \sum_{i}^{n} (y_i - \bar{y})^2 = (n-1)\,\mathrm{var}(y)$$ Can be used as a baseline - predicting y without knowing x

SSR: sum of squared regression errors (predict – mean)

$$SSR = \sum_{i}^{n} (\hat{y}_i - \bar{y})^2$$



r - **correlation coefficient**; measures linear relationship between 2 vectors x and y (positive relationship or negative) $r = \pm\sqrt{R^2}$

R^2 – **coefficient of determination**; measures how well the regression line represents the data . in multiple regression, R^2 : multiple coefficient of determination

Mean Absolute Error (MAE): $$MAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

Mean Squared Error (MSE): $$MSE = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$$

Logistic regression (Classification tasks)

The equation of the logistic (sigmoidal) curve is:

$$p = \frac{e^{b_0+b_1x}}{1+e^{b_0+b_1x}}$$

- Two classes :0 and 1
- Fits the data to a sigmoidal curve instead of a straight line (assume the relationship is nonliear)
- Result: value between 0 and 1, probability for class membership : p is probability for class 1 and 1-p is the probability for class 0
- Uses maximum likelihood method to find the parameters b0 and b1

Equations see L3 Slide 31 (pg 179 in Comb_pdf)
Tute: LogisticRegression has a regularization parameter - it is C not alpha - which controls the trade-off between fitting the training data and finding coefficients w close to 0

Overfitting and Regularization
Overfitting:
• Small error on the training set but high error on test set (new examples)
• The classifier has memorized the training examples but has not learned to generalize to new examples!
It occurs when
• we fit a model too closely to the **particularities** of the training set – the resulting model is **too specific**, works well on the training data but doesn't work well on new data
Reasons:
Data: Noise; small training set (not representative)
How algorithm operates: some are more susceptible

generalization performance = accuracy on test set
goal: yield the best test accuracy
Regularization means explicitly restricting a model to avoid Overfitting

Ridge regression
A regularized version of the standard Linear Regression (LR) • Also called **Tikhonov** regularization
the regression coefficients w are chosen so that they not only fit well the training data (as in LR) but also satisfy an additional constraint:
• the magnitude of the coefficients is as small as possible, i.e. **close to 0**
a more restricted model (less complex) is less likely to overfit • Ridge regression uses the so called **L2 regularization** (L2 norm of the weight vector)

- Minimizes the following cost function:

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}_{\text{MSE}} + \underbrace{\alpha\sum_{i=1}^{n} w_i^2}_{\text{regularization term}}$$

Goal: high accuracy on training data (low MSE)

low complexity model – w close to 0

Parameter α controls the trade-off between the performance on training set and model complexity
α closer to 0, model similar to standard LR, more complex

Increasing α makes the **coefficients smaller** (close to 0); this typically **decreases** the performance on the training set but may **improve** the performance on the test set

LASSO = Least Absolute Shrinkage and Selection Operator

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}_{\text{MSE}} + \underbrace{\alpha\sum_{i=1}^{n}||w_i||}_{\text{regularization term (L1 norm)}}$$

Goal: high accuracy on training data (low MSE)

low complexity model

Regression it adds a regularization term to the cost function but it uses the **L1** norm of the regression coeficient vector w
Consequence of using L1 – some w will become exactly 0 => some features will be completely ignored by the model – a form of automatic feature selection • Less features – simpler model, easier to interpret

L4 Naïve Bayes (P206)

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

P(H|E) posteriori probability/ conditional probability
P(H) prior probability

Two assumptions:
1) **Independence** – (the values of the) attributes are conditionally independent of each other, given the class (i.e. for each class value) P(A,B) = P(A)*P(B)
2) **Equally importance** – all attributes are equallyimportant

outlook	temp.	humidity	windy	play
sunny	cool	high	true	?

P(E1|yes)=P(outlook=sunny|yes)=?/9=2/9

P(E2|yes)=P(temp=cool|yes)=3/9

P(E3|yes)=P(humidity=high|yes)=3/9

P(E4|yes)=P(windy=true|yes)=3/9

P(yes)=? 9/14

$$P(yes|E) = \frac{P(E_1|yes)P(E_2|yes)P(E_3|yes)P(E_4|yes)P(yes)}{P(E)}$$

$$P(no|E) = \frac{P(E_1|no)P(E_2|no)P(E_3|no)P(E_4|no)P(no)}{P(E)}$$

$$P(yes|E) = \frac{\frac{2}{9}\frac{3}{9}\frac{3}{9}\frac{3}{9}\frac{9}{14}}{P(E)} = \frac{0.0053}{P(E)}$$

$$P(no|E) = \frac{\frac{3}{5}\frac{1}{5}\frac{4}{5}\frac{3}{5}\frac{5}{14}}{P(E)} = \frac{0.0206}{P(E)}$$

Since P(no|E) > P(yes|E), Naïve Bayes predicts play=no for the new day

The "zero-frequency" problem
What if an attribute value does not occur with every class value? E.g outlook=sunny had never occurred together with play=yes
Remedy: add 1 to the nominator and m to the denominator (m - number of attribute values = 3 for outlook) – (**Laplace correction or smoothing**)
a generalization of the Laplace correction called **m-estimate**

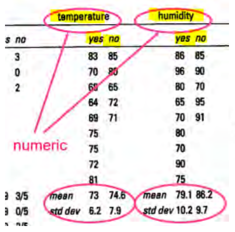
Missing values
1) missing value in the new example • do not include this attribute
2) During training: • do not include the missing values in the counts

calculate the probabilities based on the actual number of training examples without missing values for each attribute

For Numeric Attributes

assume that the numeric attributes follow a normal (or Gaussian) distribution and use probability density function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad \mu = \frac{\sum_{i=1}^{n} x_i}{n} \qquad \sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n-1}}$$



s no	temperature yes	temperature no	humidity yes	humidity no
3	83	85	86	85
0	70	80	96	90
2	68	65	80	70
	64	72	65	95
	69	71	70	91
	75		80	
	75		70	
	72		90	
	81		75	
3/5	mean 73	74.6	mean 79.1	86.2
0/5	std dev 6.2	7.9	std dev 10.2	9.7

calculate mean and std dev for each numeric attribute and apply f(x)

$$f(temperature = 66 \mid yes) = \frac{1}{6.2\sqrt{2\pi}} e^{-\frac{(66-73)^2}{2*6.2^2}} = 0.034$$

μ for temp. for play=yes

σ for temp. for play=yes

Excel: Normdist(66,73,6.2,false) = 0.034

Problems:

1) **Correlated** attributes reduce the power of Naïve Bayes - violation of the independence assumption
   Solution: apply feature selection beforehand to identify and discard correlated (redundant) attributes
2) many features are **not normally distributed**
   Solution: Discretize the data first, i.e. numerical -> nominal attributes
   Use other probability density functions, e.g. Poisson, binomial, gamma

Evaluating Machine Learning Algorithms

Evaluation Procedures

Holdout method - typically 2/3 and 1/3

Validation set - The test data can not be used for hyperparameter tuning

1) Training set - to build the classifier 2) Validation set - to tune its hyperparameters 3) Test set - to evaluate accuracy

Stratification

Ensures that each class is represented with approximately equal proportions in both data sets (training and testing)

e.g. if the class proportion in the whole dataset is 60% class1 and 40% class2, this ratio is maintained in the training and test split

Repeated holdout method

repeating the random split into training and test set several times and calculating average accuracy

e.g. repeating 10 times: in each of the 10 runs, a certain proportion (e.g. 2/3) is randomly selected for training (possibly with stratification) and the reminder is used for testing

• the 10 accuracies are averaged to produce an overall **average accuracy**

Cross-validation 10-fold cross-validation – typically used

Stratified 10-fold cross-validation – this is a standard method for evaluation used in ML • each subset is stratified

Step 1: Split data into 10 sets set1,.., set10 of approximately equal size

Step 2: A classifier is built 10 times. Each time the testing is on 1 set (blue) and the training is on the remaining 9 sets together (white)

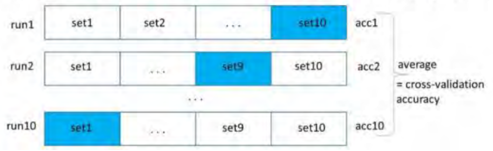
Run1: train on set1+…set9, test on set10 and calculate accuracy (acc1)

Run2: train on set1+…set8+set10, test on set9 and calculate accuracy (acc2)

….

Run10: train on set2+…set10, test on set1 and calculate accuracy (acc10)

Step 3: Calculate the cross validation accuracy = average (acc1, acc2,…acc10)



Leave-one-out cross-validation

Advantages: Makes the best use of data

Deterministic procedure

Disadvantage:

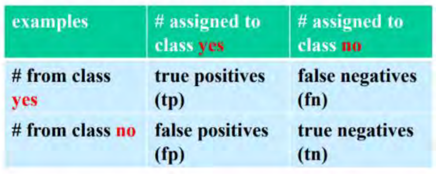
High computational cost, especially for large datasets

Grid search with cross-validation for parameter tuning

Performance Measures TP|FN|FP|TN

2 class problem: yes and no

4 different outcomes - confusion matrix:



examples	# assigned to class yes	# assigned to class no
# from class yes	true positives (tp)	false negatives (fn)
# from class no	false positives (fp)	true negatives (tn)

accuracy in terms of tp, fn, fp and tn? accuracy= (tp+tn)/(tp+fn+fp+tn)

The confusion matrix is not a performance measure, it allows us to calculate performance measures

```
 a  b  c   <-- classified as
50  0  0 |  a = Iris-setosa
 0 44  6 |  b = Iris-versicolor
 0  3 47 |  c = Iris-virginica
```

In addition to accuracy, other performance measures are precision (P), recall (R) and their combination - F1 score classification

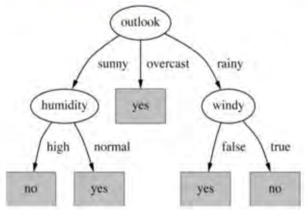
$$P = \frac{tp}{tp+fp} \qquad R = \frac{tp}{tp+fn} \qquad F1 = \frac{2PR}{P+R}$$

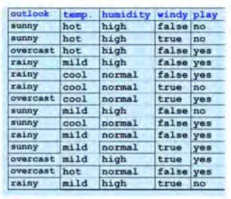
Accuracy = (50+44+47)/(50+0+0+0+44+6+0+3+47)=94%

Ideally, we have high precision and high recall

L5 Decision Tree (P264)

- each *non-leaf node* the corresponds to a test for the values of an attribute
- each *branch* corresponds to an attribute value
- each *leaf node* assigns a class





outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

(Not all attributes are included in decision tree, it has inbuilt feature selection)

Strategy: top-down learning using recursive divide-and-conquer process:

• First: Select the best attribute for root node and create branch for each possible attribute value

• Then: Split examples into subsets, one for each branch extending from the node

• Finally: Repeat recursively for each branch, using only the examples that reach the branch

How do we find the best attribute?
A leaf node with **only 1 class (yes or no)** will not have to be split further and the recursive process will terminate
• We would like this to happen as soon as possible as we seek small trees •a measure of purity of each node

Entropy
The measure of purity that we will use is called **information gain** based on another measure **entropy**
Given a set of examples with their class, entropy measures the **homogeneity (purity)** of this set with respect to the class **smaller entropy, greater the purity**

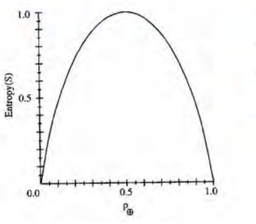
- Entropy H(S) of data set S:
  $H(S) = I(S) = -\sum_i P_i . \log_2 P_i$
  P_i - proportion of examples that belong to class *i*
- Different notation used in textbooks, we will use H(S) and I(S)
- For our example: weather data - 9 yes and 5 no examples:

$$H(S) = -P_{yes}\log_2 P_{yes} - P_{no}\log_2 P_{no} = I(P_{yes}, P_{no}) = I(\frac{9}{14},\frac{5}{14}) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940\ bits$$

- The entropy is measured in bits
- When calculating entropy, we will assumed that $\log_2 0 = 0$

2 classes: yes and no
on x: p, the proportion of positive examples
(the proportion of negative examples will be 1-p)
on y: the entropy H(S)

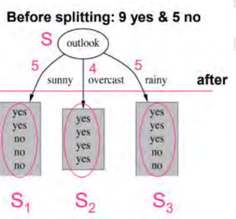
$$H(S) = I(p,(1-p)) = \\ = -p\log_2 p - (1-p)\log_2(1-p)$$



- H(S) ∈ [0,1]
- H(S)=0 => all elements of S belong to the same class (max purity, min value of entropy)
- H(S)=1 => equal number of yes & no (min purity, max value of entropy)

Information gain measures the **reduction in entropy** caused by using an attribute to partition the set of training examples • The best attribute is the one with the **highest information gain** (i.e. with the biggest reduction in entropy)

Calculating information gain



1) Calculate the entropy of the set of examples before split.

$$T1 = H(S) = I(\frac{9}{14},\frac{5}{14}) = 0.940\ bits$$

2) Calculate weighted entropy per each branch.

$$T2 = H(S \mid outlook) = \frac{5}{14}\cdot H(S_1) + \frac{4}{14}\cdot H(S_2) + \frac{5}{14}\cdot H(S_3)$$

$$H(S \mid outlook = sunny) = I(\frac{2}{5},\frac{3}{5}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971\ bits$$

$$H(S \mid outlook = overcast) = I(\frac{4}{4},\frac{0}{4}) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0\ bits$$

$$H(S \mid outlook = rainy) = I(\frac{3}{5},\frac{2}{5}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971\ bits$$

$$H(S \mid outlook) = \frac{5}{14}\cdot 0.971 + \frac{4}{14}\cdot 0 + \frac{5}{14}\cdot 0.971 = 0.693\ bits$$

$$Gain(S|outlook) = H(S) - H(S|outlook) = 0.940 - 0.693 = 0.247\ \text{bits}$$

3) Calculate the information gain for other features

$$Gain(S|temperature) = 0.029\ bits$$
$$Gain(S|humidity) = 0.152\ bits$$
$$Gain(S|windy) = 0.048\ bits$$

we select outlook as it has the highest information gain

Pruning decision trees
Overfitting – high accuracy on the training data but low accuracy on new data

When does overfitting occurs in decision trees?
Training data is too small -> not enough representative examples to build a model that can generalize well on new data
Noise in the training data, e.g. incorrectly labelled examples -> the decision tree learns them by adding new braches and making the tree overly specific

Solution : Use **tree pruning** to avoid overfitting

Two main strategies
• **Pre-pruning** - stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
• **Post-pruning** (is preferred in practice) – fully grow the tree, allowing it to perfectly cover the training data, and then prune it

Different post-pruning methods, e.g.: • sub-tree replacement • sub-tree raising • converting the tree to rules and then pruning them

How much to prune?
Use a **validation set to decide**

Pruning by sub-tree replacement – idea (P288/ L5a-p25)
Bottom-up – from the bottom of the tree to the root

Discretising numeric attributes (binary split e.g temp<45)

- Values of *temperature*:

64	65	68	69	70	71	72	73	74	75	80	81	83	85
yes	no	yes	yes	yes	no	no	no	yes	yes	no	yes	yes	no

- 7 possible splits; let's consider the split between 70 and 71
- Calculate Information gain for:
  - temperature < 70.5 : 4 yes & 1 no
  - temperature =>70.5 : 4 yes & 5 no

$$H(S) = -\frac{8}{14}\log_2\frac{8}{14} - \frac{6}{14}\log_2\frac{6}{14} = 0.985\ bits$$

$$H(S_{temp<70.5}) = -\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5} = 0.722\ bits$$

$$H(S_{temp=>70.5}) = -\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} = 0.991\ bits$$

$$H(S \mid temp70.5) = \frac{5}{14}0.722 + \frac{9}{14}0.991 = 0.895\ bits$$

$$Gain(S \mid temp70.5) = 0.985 - 0.895 = 0.09\ bits$$

outlook	temp.	humidity	windy	play
sunny	85	high	false	no
sunny	80	high	true	no
overcast	83	high	false	yes
rainy	70	high	false	yes
rainy	68	normal	false	yes
rainy	65	normal	true	no
overcast	64	normal	true	yes
sunny	73	high	false	no
sunny	69	normal	false	yes
rainy	74	normal	false	yes
sunny	75	normal	true	yes
overcast	72	high	true	yes
overcast	81	normal	false	yes
rainy	71	high	true	no

.edu.au COMP5318 ML&DM, week 5a, 2021

Alternatives to information gain
If an attribute is highly-branching (**with a large number of values**), information gain will select it! Example: imagine using ID code as one of the attributes : All single instance subsets have entropy=0

Gain ratio is a modification of information gain that reduces its bias towards highly branching attributes • It takes into account the **number of branches** when choosing an attribute and **penalizes highly-branching attributes**

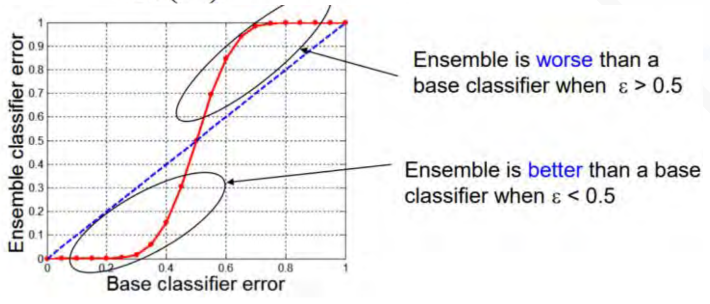
DT Discussion
Advantages:
easy to visualize and understand by nonexperts and clients
Interpretability increases the trust in using the machine learning model in practice
Variations:
purity can be measured in different ways, e.g. CART uses Gini Index not entropy

Ensemble Methods
When do ensemble methods work? (better than single)
• The base classifiers should be good enough, i.e. better than a random guessing (ε <0.5 for binary classifiers)
• The base classifiers are independent of each other

Example: 25 binary classifiers. Each base classifier has an error rate ε =0.35 on the test set (i.e. accuracy=0.65). To predict the class of a new example, the predictions of the base classifiers are combined by majority vote.

Error rate of the ensemble

$$e_{ensemble} = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$



Methods for constructing ensembles
Manipulating the training data - (e.g. Bagging and Boosting)
Manipulating the attributes (e.g Random Forest)
Manipulating the class labels
Manipulating the learning algorithm

Bagging (bootstrap aggregation)
Bootstrap sample D' from D: contains also n examples, randomly chosen from D with replacement (i.e. some examples from D will appear more than once in D', some will not appear at all)
• On average, 63% of the examples in D will also appear in D' as it can be shown that the probability to choose an example is $(1\text{-}1/n)^n$

Steps:
• Create M bootstrap samples
• Use each sample to build a classifier
• To classify a new example: get the predictions of each classifier and combine them with a majority vote
  • i.e. the individual classifiers receive equal weights
Especially effective for **unstable classifiers** (decision trees, neural networks)

Boosting (most widely used)
Idea: Make the classifiers complement each other
How: The next classifier should be created using examples that were difficult for the previous classifiers

AdaBoost (weighed training set) – P314 (L5b p19)
• Each training example has an **associated weight** ($\geq$0)
The higher the weight, the more difficult the example was to classify by the previous classifiers
• Examples with higher weight will have a higher chance to be selected in the training set for the next classifier

Weak learner
is a classifier whose classification performance is slightly better than random guessing (i.e. 50% for binary classification)

Gradient Boosting
while AdaBoost updates the weights of the examples at each iteration, Gradient Boosting adds a new model that **minimizes the error of the previous model**

Create model 1: DT1 fit on training data (X,y), store model
To create model 2:
- Evaluate DT1 on training data, calculate error:
  y2 = y (actual value) - predicted value by DT1
- Create model 2: DT2 fit on (X,y2), store model

To create model 3:
- Evaluate DT2 on training data, calculate error:
  y3 = y2 - predicted value by DT2
- Create model 3: DT3 fit on (X,y3), store model

Now we have 3 decision trees. To make a prediction for a new example: sum the predictions of DT1, DT2 and DT3

Bagging and Boosting - comparison
• **Similarities**
• Use voting (for classification) and averaging (for prediction) to combine the outputs of the individual learners
• Combine classifiers of the same type, typically trees – e.g. decision stumps or decision trees
• **Differences**
  • *Creating base classifiers*:
    • Bagging – separately
    • Boosting – iteratively – the new ones are encouraged to become experts for the misclassified examples by the previous base learners (complementary expertise)
  • *Combination method*
    • Bagging – equal weighs to all base learners
    • Boosting – different weights - based on performance on training data

Random Forest
training data with K features, create an ensemble of M classifiers each using a smaller number of features L (L<K)
Steps:
1) Create feature subsets by **random selection** from the original feature set => creating multiple versions of the training data, each containing only the selected features
2) Build a classifier for each version of the training data
3) Combine predictions with **majority vote**
Combines decision trees • Uses 1) **bagging** + 2) **subset of features** (during decision tree building, when selecting the most important attribute) – typically start by using the square of number of features, then try a few settings

Parameters:
n - number of training examples, m – number of all features, k – number of features to be used by each ensemble member (k<m), M – number of ensemble members (bootstrap samples)

Comments on Random Forests
Performance depends on
• **Accuracy** of the individual trees (strength of the trees)
• **Correlation** between the trees
Ideally: accurate individual trees but less correlated

• Bagging and random feature selection are used to **generate diversit**y and **reduce the correlation** between the trees
• As the number of features k increases, both the strength and correlation increase
• Random Forest typically outperforms a single DT
• Robust to overfitting
• Fast as only a subset of the features are considered

T5:
Pre-pruning stops growing the tree before it perfectly fits the training data. **Restricting the tree depth** is an example of this approach but there are other approaches, e.g. based on **validation set performance**
Post-pruning involves fully growing the tree, allowing it to fit the training data, and then pruning the tree. The main pruning methods are **sub-tree replacement**, **sub-tree raising** and **rule pruning**.

Another adv of pruning: **improve the interpretability**, as tree will be smaller – easier to visualise and understand

Compare decision trees with k-nearest neighbor and linear regression. What advantages do they offer?
The main advantage is that the resulting model (the produced decision tree) can be **easily visualized and understood by non-experts and clients**. This increases the trust in using the machine learning model in practice. Decision trees can also form complex **non-linear decision boundaries**, an advantage over linear regression models.

Bagging:
bag_clf = BaggingClassifier(
DecisionTreeClassifier(random_state=42),
n_estimators=500, max_samples=100, bootstrap=True, random_state=42)

n_estimators=500
500 decision trees
max_samples=100
100 examples randomly sampled from the training data

Random Forest:
max_features - the number of features to consider when looking for the best split;
the default is **max_features=sqrt(n_features)**

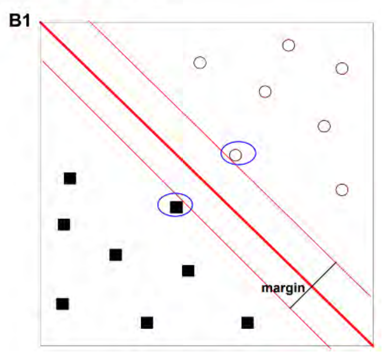
As we increase the number of features, the decision trees part of the ensemble become **more accurate** but also **more similar to each other**, which **increases the overfitting** and in turn **reduces the accuracy** of the ensemble.

Disdavantages of Random Forest compared to a single decision tree
- **Loss of interpretability** - not possible to interpret hundreds of decision trees, and the trees in Random Forest also tend to be bigger as there is no pruning (in the original version of the algorithm)
- **Computationally expensive** on large datsets

Adaboost
ada_clf = AdaBoostClassifier(
DecisionTreeClassifier(**max_depth=1**), **n_estimators=200**, **learning_rate=0.5**, random_state=42)

n_estimators=200 200 models
learning_rate=0.5 how quickly we change the misclassified instance weights from step to step

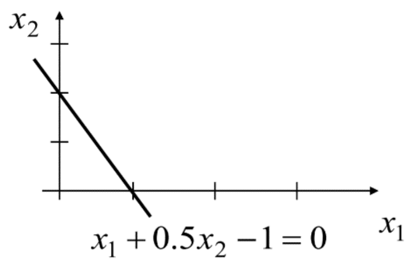
Gradient Boosting
gb_clf = GradientBoostingClassifier(max_depth=1, n_estimators=200, learning_rate=0.2, random_state=42)

L6 SVM & PCA(p343)
A decision boundary B1:


Support vectors are the examples (data points) that lie closest to the decision boundary; they are circled
Margin – the separation between the boundary and the closest examples

Which hyperplane should we select?
The hyperplane with the biggest margin is called the **maximum margin hyperplane**
It is the hyperplane with the **highest possible distance to the training examples**
• SVM selects the maximum margin hyperplane

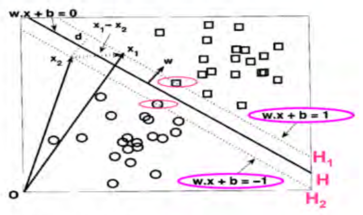
Linear decision boundary



A decision boundary of a linear classifier is w · x + b = 0
If we know the decision boundary, we can easily classify a new example x by calculating f = wx+b and determining the sign
if is above the decision boundary w · x + b>0
if is below the decision boundary w · x + b<0
= sign (w · x + b)

SVM - problem statement

Our separating hyperplane is H
H is in the middle of 2 other hyperplanes, H1 and H2, defined as:
$H_1 : \mathbf{w} \cdot \mathbf{x} + b = 1$
$H2 : \mathbf{w} \cdot \mathbf{x} + b = -1$
The points laying on H1 and H2 are the support vectors
d is the margin of H
It can be shown that: $d = \frac{2}{\|\mathbf{w}\|}$



To <u>maximize</u> the margin d, we need to <u>minimize</u> ||w||
This is equivalent to <u>minimizing</u> the quadratic function: $\frac{1}{2}\|\mathbf{w}\|^2$

Given: a set of labelled training examples
Learn: the maximum margin hyperplane such as all training examples are classified correctly
This could be formulated as a constraint optimization problem:

- Given N training examples $(\mathbf{x}_i, y_i), i = 1,..,N$
  $\mathbf{x}_i = (x_{i1},...,x_{im})^T, y_i = \{-1,1\}$
  training vector class
- Minimize $\frac{1}{2}\|\mathbf{w}\|^2$ ⟵ Maximazing the margin
- Subject to the linear constraint $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i$
  Another way of expressing correct classification of all training examples i=1..N

This is an optimization problem that can be solved using Quadratic Programming (QP) and the Lagrange multiplier method
Firstly, the problem is transformed into an equivalent form using Lagrange multipliers λ :

$$\max \mathbf{w}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

Dot product of pairs of training vectors
Class value of the training vectors

$$subject\ to\ \lambda_i \geq 0, \sum_{i=1}^{N} \lambda_i y_i = 0$$

The values of λs are found using QP
The solution (i.e. the optimal decision boundary) is given by:

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

Max w is a linear combination (coefficient λ * target value*training vector) of the training examples
many of the λs are 0 -> linear combination of a small number of training examples
• The training examples xi with non-zero λi are the **support vectors** and they are the examples closest to the decision boundary w
• => the optimal decision boundary w is a linear combination of support vectors

Classifying new examples

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \quad \text{maximum margin hyperplane}$$

- To classify a new example z:

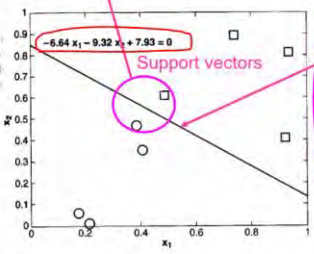
$$f = \mathbf{w} \cdot \mathbf{z} + b = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{z} + b$$

Dot product of the new vector and the support vectors

$$sign(f)$$

i.e. the new example belongs to class 1, if f>0 or class -1 if f<0

	features x_1	x_2	class y	Lagrange Multiplier
x1	0.3858	0.4687	1	65.5261
x2	0.4871	0.611	−1	65.5261
	0.9218	0.4103	−1	0
	0.7382	0.8936	−1	0
	0.1763	0.0579	1	0
	0.4057	0.3529	1	0
	0.9355	0.8132	−1	0
	0.2146	0.0099	1	0



- 8 2-dim. training examples; 2 classes: -1,1
- After solving the problem with QP we find the λs
- Only 2 λs are non-zero (x1 & x2) and they correspond to the support vectors
- Using the λs , the weights (defining the decision boundary are):

$$w_1 = \sum_{i=1}^{2} \lambda_i y_i \mathbf{x}_{i1} = 65.5261(1*0.3858 -1*0.4871) = -6.64$$
$$w_2 = \sum_{i=1}^{2} \lambda_i y_i \mathbf{x}_{i2} = 65.5261(1*0.4687 -1*0.611) = -9.32$$

$b = 7.93$ //there is a formula for b (not shown)

- Classifying new examples:
  - above the decision boundary: class 1
  - below: class -1

W1 = 65.5261(1*0.3858 + -1*0.4871) (1& -1 Class y value)

SVM with soft-margin

• We can modify our method to allow some misclassifications, i.e. by considering the **trade-off** between the **margin width** and the **number of misclassifications**
• As a result, the modified method will construct linear boundary even if the data is not linearly separable
Solution: - **additional parameter C**
C is a hyper-parameter that allows for **a trade-off between maximizing the margin and minimizing the training error**
• **Large C**: **more emphasis on minimizing the training error** than maximizing the margin

Non-linear SVM

•Transform the data from its original feature space to a new space use a linear boundary to separate the data
• If the transformation is **non-linear** and to a **higher dimensional** space, it is **more likely** than a linear decision boundary can be found in it
• The learned linear decision boundary in the new feature space is **mapped back** to the original feature space, **resulting in a non-linear** decision boundary in the original space

transformation from old to new space:

$$\phi = (x_1, x_2) \rightarrow (x_1{}^2 - x_1, x_2{}^2 - x_2)$$

kernel trick
Method for computing the dot product of a pair of vectors in the new space without first computing the transformation of each vector from the original to the new space

We will compute the dot product of the **original features** and use it in a function (called kernel function) to determine the dot product of the **transformed features**

The kernel function specifies the **relationship** between the dot products in the original and transformed space

Example:

2 dim original and 3 dim new space: $\Phi:(x_1,x_2)=(x_1^2,\sqrt{2}x_1x_2,x_2^2)$

u, v – vectors in the original space (2-dim)
Φ(u), Φ(v) – transformed vectors u and v in the new space (3-dim)

$\mathbf{u}\overset{\Phi}{\rightarrow}\Phi(\mathbf{u}),\ \mathbf{v}\overset{\Phi}{\rightarrow}\Phi(\mathbf{v})$

Let's calculate the dot product of Φ(u) and Φ(v)

$$\Phi(\mathbf{u})\cdot\Phi(\mathbf{v})=(u_1^2,\sqrt{2}u_1u_2,u_2^2)\cdot(v_1^2,\sqrt{2}v_1v_2,v_2^2)=$$
$$=u_1^2v_1^2+2u_1u_2v_1v_2+u_2^2v_2^2=(u_1v_1)^2+(u_2v_2)^2+2u_1u_2v_1v_2=$$
$$=(u_1v_1+u_2v_2)^2=(\mathbf{u}\cdot\mathbf{v})^2$$

The dot product in the new space can be expressed via the dot product in the original space!
This relationship is specified by the kernel function: $\Phi(\mathbf{u})\cdot\Phi(\mathbf{v})=(\mathbf{u}\cdot\mathbf{v})^2$

$$\underline{K(\mathbf{u},\mathbf{v})=\Phi(\mathbf{u})\cdot\Phi(\mathbf{v})=(\mathbf{u}\cdot\mathbf{v})^2}$$

Kernel functions allow the dot product in the new space to be computed without first computing Φ for each input vector

Relationship only exists for some types of functions
Functions K for which this is true (i.e. such Φ exist) need to satisfy the **Mercer's Theorem**, i.e. this **restricts the class of functions K we can use**

SVM training and classification using kernel functions

- Training:

$$\max \mathbf{w}(\lambda)=\sum_{i=1}^{N}\lambda_i-\frac{1}{2}\sum_{i,j=1}^{N}\lambda_i\lambda_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$$

$$subject\ to\ \lambda_i\geq 0,\sum_{i=1}^{N}\lambda_i y_i=0$$

- Optimal hyperplane in the new space: $\mathbf{w}=\sum_{i=1}^{N}\lambda_i y_i\Phi(\mathbf{x}_i)$

- Classifying new example z:

$$f=\mathbf{w}\cdot\mathbf{z}+b=\sum_{i=1}^{N}\lambda_i y_i K(\mathbf{x}_i,\mathbf{z})+b$$

Dimensionality Reduction PCA (P376)
Problems with high dimensional data:
Slower Training/unreliable classification/overfitting/not possible to interpret/hard to visualize/not all features are important
Dimensionality reduction removes **redundant** and **highly correlated** features and reduces **noise** in the data

PCA (also called a **feature projection** method)
PCA main idea
Given: N examples with dimensionality **m** (i.e. m features)
Find: **m** new axes Z1 ,…, Zm orthogonal to each other such that Var(Z1) > Var(Z2)…. > Var(Zm) • Z1,…, Zm are called principal components
The principal components are vectors that define a new coordinate system
They are **ordered** based on **how much variance** they capture
• The first axis goes in the direction of the **highest variance** in the data
• The second axis is orthogonal to the first one and goes in the direction of the second highest variance
• The third one is orthogonal to both the first and second and goes in the direction of the third highest variance, and so on

How to reduce data dimensionality?
Select k largest principal components Z1,Z2….Zk and project our data points on them (k<m)

How many principal components (dimensions) to select?
Method 1: Set min % of variance that should be preserved, e.g. 95%
Method 2: (Elbow method) an elbow in the curve where the variance stops growing fast

How to find the principal components?
Using a standard matrix factorization method, called **Singular Value Decomposition (SVD)**

Theorem: Any *n* x *m* matrix X (n ≥m) can be written as the product of 3 matrices

$$X=U\times\Lambda\times V^T$$

U - *n* x *m* orthogonal matrix
$\mathbf{V}^T$ - the transpose of an *m* x *m* orthogonal matrix
Λ - *m* x *m* diagonal matrix containing the singular values (positive or zero elements)

V defines the new set of axes (**principal components**)
• Provides important information about the variance in data – the 1st axis goes in the direction with highest variance, 2nd – 2nd highest variance and so on
• X is the **original data**
• U is the **transformed data**, i.e. the i-th row of U contains the coordinates of the i-th row of X in the new coordinate system

X can be re-written as:

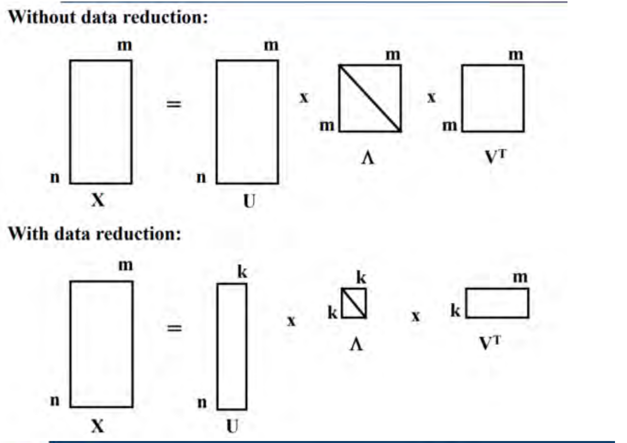
$$X=\lambda_1u_1v_1^T+\lambda_2u_2v_2^T+\cdots\lambda_mu_mv_m^T$$

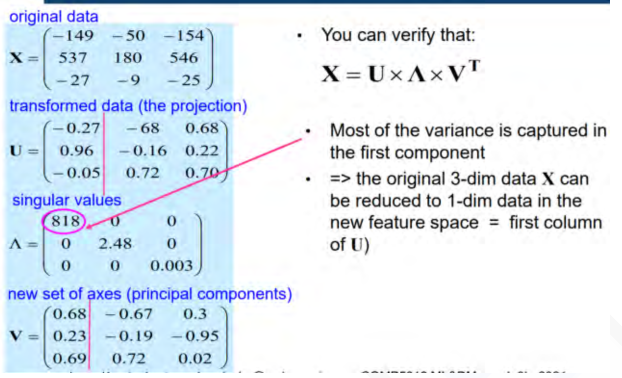
where λ are sorted in decreasing order
Data reduction comes from taking only the first *k* components (*k*<*m*)

$$X_{reduced}=\lambda_1u_1v_1^T+\lambda_2u_2v_2^T+\cdots\lambda_ku_kv_k^T$$

=> The size of the data can be reduced by eliminating the weaker components (the ones with low variance)

Graphical representation of SVD (P391 L6b – p16)





original data

$$\mathbf{X} = \begin{pmatrix} -149 & -50 & -154 \\ 537 & 180 & 546 \\ -27 & -9 & -25 \end{pmatrix}$$

transformed data (the projection)

$$\mathbf{U} = \begin{pmatrix} -0.27 & -68 & 0.68 \\ 0.96 & -0.16 & 0.22 \\ -0.05 & 0.72 & 0.70 \end{pmatrix}$$

singular values

$$\Lambda = \begin{pmatrix} 818 & 0 & 0 \\ 0 & 2.48 & 0 \\ 0 & 0 & 0.003 \end{pmatrix}$$

new set of axes (principal components)

$$\mathbf{V} = \begin{pmatrix} 0.68 & -0.67 & 0.3 \\ 0.23 & -0.19 & -0.95 \\ 0.69 & 0.72 & 0.02 \end{pmatrix}$$

- You can verify that:

$$\mathbf{X} = \mathbf{U} \times \mathbf{\Lambda} \times \mathbf{V}^T$$

- Most of the variance is captured in the first component
- => the original 3-dim data **X** can be reduced to 1-dim data in the new feature space = first column of **U**)

SVD for compression

Compression ratio

$$r = \frac{k(1+n+m)}{n \times m}$$

Compression ratio = after compression/ before compression

For n >> m >k, this ratio is approximately k/m

e.g. if m = 365 and k = 10 => r =0.28 or 28%

PCA for feature extraction in images – face recognition

Image Compression Examples

L6 Tutorial:

use the **SVC** class to create linear and non-linear SVM classifiers. SVC stands for "Support Vector Classifier"

We need to set the **kernel parametet** to "linear"; the default is "rbf", corresponding to Radial-Basis Function (RBF) kernel, i.e. a non-linear SVM.

SVM classifiers are very sensitive to the values of the parameters.

gamma and C parameters (the most important parameters)

✓ The **parameter C** controls the trade-off between the performance on training set and model complexity

small C means a very restricted mode: a smaller C should be used if the model overfits and a larger if it underfits

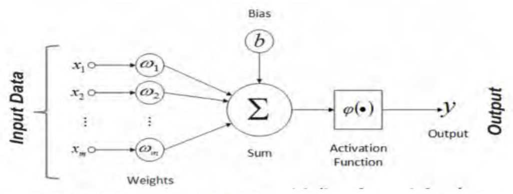
✓ **gamma** controls the **width of the Gaussian kernel** - smaller gamma means larger width and vice versa.

gamma acts as a regularization parameter - if the SVM model is **overfitting**, gamma should be reduced; conversely, if it is underfitting - gamma should be increased

Regarding the type of kernel - the rule of thumb is to **try first a linear kernel**. In addition to SVC(kernel="linear"), there is another option: using the class LinearSVC. LinearSVC is much faster that SVC(kernel="linear"), especially if the training set is large (has many features and many examples). Next, SVM with RBF kernel should be tried as it typically works well, and then other types of kernels.

L7 Introduction to Artificial Neural Networks (p414)

Perceptron Learning Rule

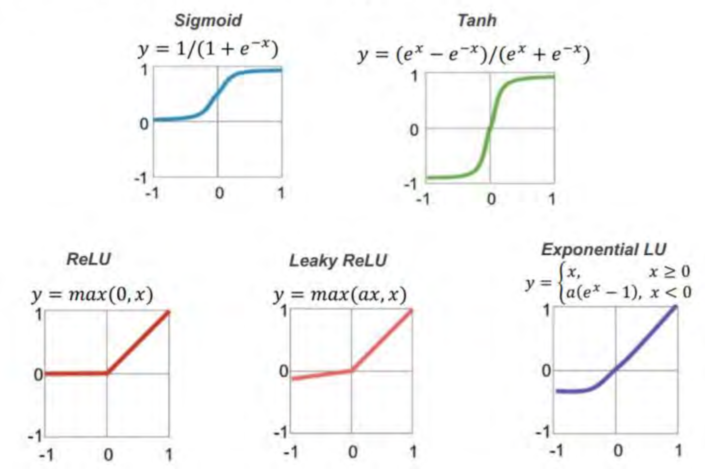


1. *Initialise the weights* $(w_0, w_1, \dots, w_m)$
2. *For all training example* (x_m, y_m)
   - *Compute* $f(\sum_{i=0}^{m} w_i x_i + b)$
   - *Update the parameters (w, b)*
   - *(we will discuss about this in the back-propagation section)*
3. *Until stopping condition is met*

Activation function:

aims to add a non-linear property to the function (neural network)

Activation Function – different types



Back-propagation is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration). Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

1. Initialise the weights (w0,w1...Wm)
2. For all training example (Xm,Ym)
• Compute all in hidden layer, output layer
• Update the parameters (w, b)
3. Until stopping condition is met

$$w^{new} = w - \gamma \nabla C(x)$$

Cost Function – Method 1: MSE (Mean Squared Error)
Mean Square Error (MSE) is the most commonly used regression loss function. MSE is the average of squared distances between our target variable and predicted values

$$MSE = \frac{1}{n}\sum_i(\hat{y}_i - y_i)^2 \qquad ACE = -\frac{1}{n}\sum_i y_i log(\hat{y}_i)$$

Only if the y is encoded by one-hot vector

Cost Function – Method 2: ACE (Averaged Cross Entropy Error)
ACE = Cross-entropy loss, or log loss, measures the performance of a classification model whose **output is a probability value between 0 and 1**. Cross-entropy loss increases as the predicted probability diverges from the actual label.

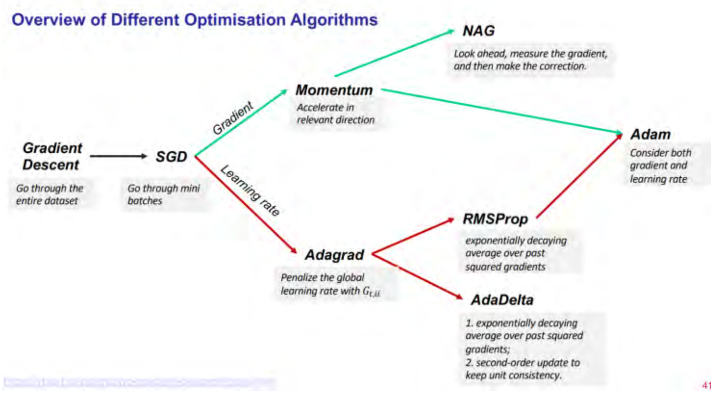
Gradient Descent
In neural networks, the key technique to arrive at the optimal weights is the Gradient Descent algorithm. This algorithm relies on a hyperparameter called the **learning rate** which allows one to moderate the **rate of weight change**, such that the cost function is minimized.

Vanilla gradient descent, aka batch gradient descent, computes the gradient of the cost function w.r.t. to the parameters θ for the **entire** training dataset.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

it is extremely slow for calculating all dataset in every epoch.

Stochastic Gradient Descent (SGD)
performs a parameter update for each training example X^i and label y^i . (picks a random instance in the training set at every step and computes the gradients based only on that single instance.) SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster than the GD.



Overview of Different Optimisation Algorithms
Gradient Descent
Go through the entire dataset
SGD
Go through mini batches
Gradient
Learning rate
Momentum
Accelerate in relevant direction
NAG
Look ahead, measure the gradient, and then make the correction.
Adam
Consider both gradient and learning rate
Adagrad
Penalize the global learning rate with $G_{t,ii}$
RMSProp
exponentially decaying average over past squared gradients
AdaDelta
1. exponentially decaying average over past squared gradients;
2. second-order update to keep unit consistency.
41

Data Augmentation
Adding noise to the input: a special kind of augmentation.
Weight Decay
Limiting the growth of the weights in the network. • A term is added to the original loss function, penalizing large weights
Dropout
The weights are scaled-down by a factor of p (e.g. 0.5).

Batch Normalization
Covariate Shift is undesirable, because the later layers have to keep adapting to the change of the type of distribution.

BN reduces effects of exploding and vanishing gradients, because every becomes roughly normal distributed. Without BN, low activations of one layer can lead to lower activations in the next layer, and then even lower ones in the next layer and so on.

Tutorial
per_clf = Perceptron(max_iter=2000, tol=1e-3, random_state=42)
tol=1e-3 : once the change in loss function goes below le-3, it will stop

Dense layer
Fully connected layer

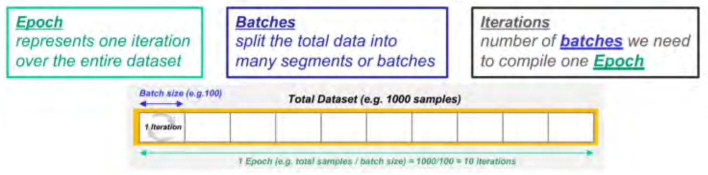
Softmax activation function with 10 output
Transform 10 numbers into probability (0,1), sum =1; take the max of probability, which is our prediction

20epochs: every image will be looked at 20 times
Argmax: gives the location of the cell which has the highest value

L8 CNN & RNN (P498)
Weight Initialisation
Modern deep learning libraries, such as Keras, offer a host of network initialization methods, all are variations of initializing the weights with small random numbers.

Activation Function	Initialisation	Code
Sigmoid	Xavier	np.random.randn(n_input,n_output) / sqrt(n_input)
ReLU	He	np.random.randn(n_input,n_output) / sqrt(n_input / 2)

Epoch, Batch Size, and Iteration



Epoch
represents one iteration over the entire dataset
Batches
split the total data into many segments or batches
Iterations
number of batches we need to compile one Epoch
Batch size (e.g.100)
Total Dataset (e.g. 1000 samples)
1 Iteration
1 Epoch (e.g. total samples / batch size) = 1000/100 = 10 iterations

Fully Connected Neural Networks : Basic neural networks are vulnerable to the change of position, size, angle of images.

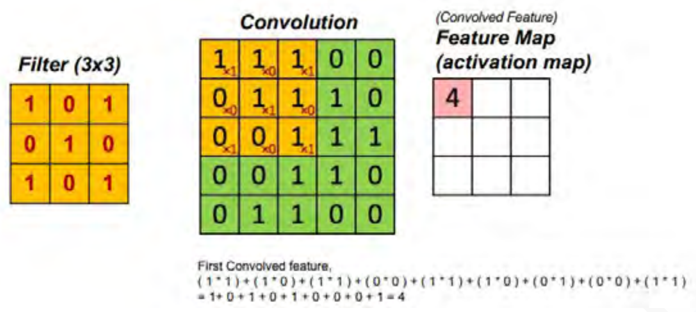
Convolutional Neural Networks (CNN/ ConvNets)
CNN architectures make the **explicit assumption** that the inputs are **images**, which allows us to encode **certain properties** into the architecture
From **Feature Extraction** (Convolutional layer and pooling layer) to **Classification** (fully connected layer)

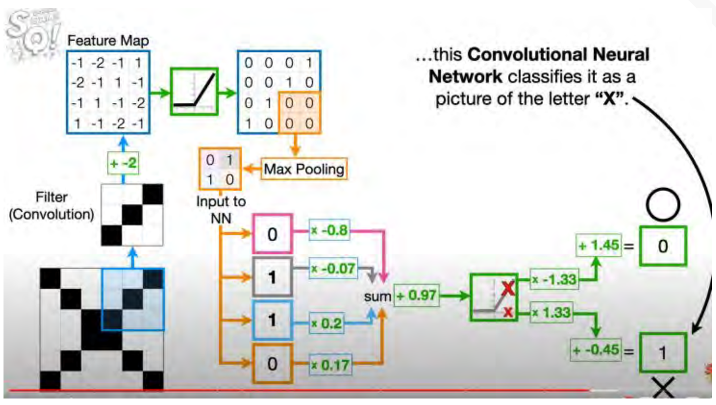
INPUT – CONV – RELU – POOL flatten– **FC**
The CONV-RELU and POOL can be repeated (depends on your model)

CNNs are very **similar** to **fully connected Neural Networks**:
• made up of neurons that have learnable **weights** and **biases**.
• each neuron receives some **inputs**, performs a **dot product** (optionally follows it with a **non-linearity**)
• express a single differentiable score function (from the raw image pixels on one end to class scores at the other)
• have a **loss function** (e.g. Softmax) on the last (fully-connected) layer Hence, all the tricks for learning fully connected Neural Networks still apply.

CONV (Convolutional) layer
Compute the **output of neurons** that are connected to **local regions** in the input, each computing a dot product between their **weights (from the learnable filters)** and a **small region** they are connected to in the input volume.
• This may result in volume such as [32 x 32 x 12] if we decided to use 12 filters

ConvNet is a sequence of Convolution Layers, interspersed with activation functions

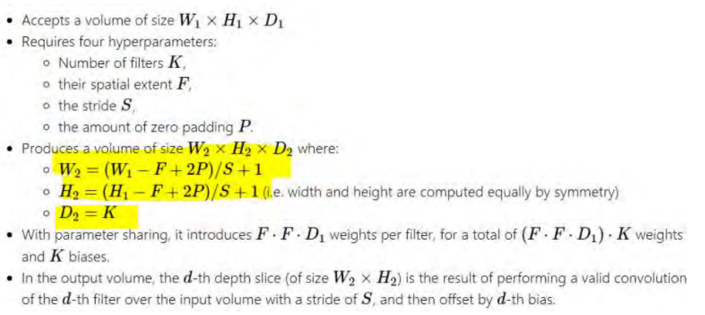




Feature Map
Filter (Convolution)
Max Pooling
Input to NN
sum
...this Convolutional Neural Network classifies it as a picture of the letter "X".

CONV (Convolutional) layer – Padding
It will be convenient to pad the input volume with **zeros around the border**. The size of this zero padding is a **hyperparameter**. The nice feature of zero padding is that it will **allow us to control the spatial size of the output volumes.**

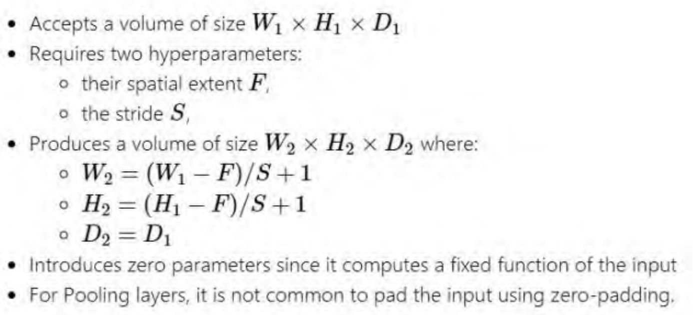
CONV (Convolutional) layer – Strid
Stride controls how the filter convolves around the input volume. • When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters **jump 2 pixels at a time** as we slide them around. This will produce smaller output volumes spatially.



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

RELU layer
apply an elementwise activation function, such as the **max(0,x)** thresholding at zero. This leaves the size of the volume unchanged ([32x32x12])

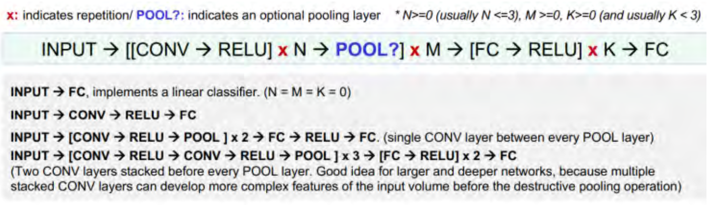
POOL (Pooling) layer – max pooling or average pooling
perform a down-sampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12]
It is common to periodically insert a Pooling layer in-between successive Conv layers.
• progressively **reduce the spatial size** of the representation to reduce the **amount of parameters and computation** in the network, and hence to also control overfitting.



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

FLATTEN (Flattening) Layer
In between the convolutional layer and the fully connected layer, there is a 'Flatten' layer. Flattening transforms a multi-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier

FC (Full Connected) layer
Compute the class scores, each of the 2 categories correspond to a class score, such as among the 2 categories of our dataset. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume



x: indicates repetition/ POOL?: indicates an optional pooling layer   * N>=0 (usually N <=3), M >=0, K>=0 (and usually K < 3)

INPUT → [[CONV → RELU] x N → POOL?] x M → [FC → RELU] x K → FC

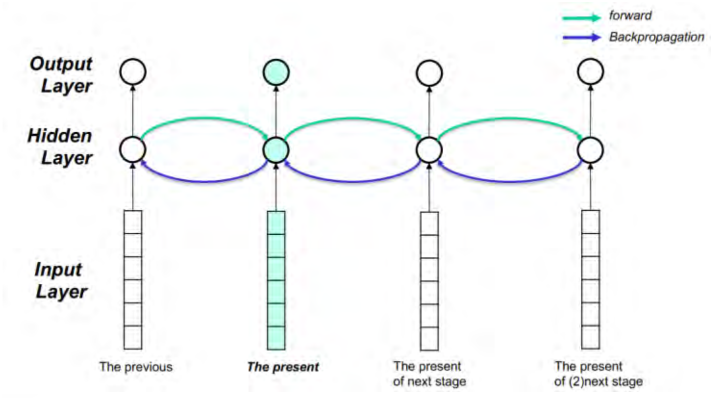
INPUT → FC, implements a linear classifier. (N = M = K = 0)
INPUT → CONV → RELU → FC
INPUT → [CONV → RELU → POOL] x 2 → FC → RELU → FC. (single CONV layer between every POOL layer)
INPUT → [CONV → RELU → CONV → RELU → POOL] x 3 → [FC → RELU] x 2 → FC
(Two CONV layers stacked before every POOL layer. Good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation)

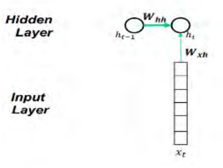
– LeNet-5s [CONV-POOL-CONV-POOL-FC-FC]
– AlexNet, VGG, GooLeNet

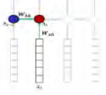
Recurrent Neural Network

Natural Language Processing with Neural Network + Memory = Sequence Modelling

Neural Network + Memory = Recurrent Neural Network





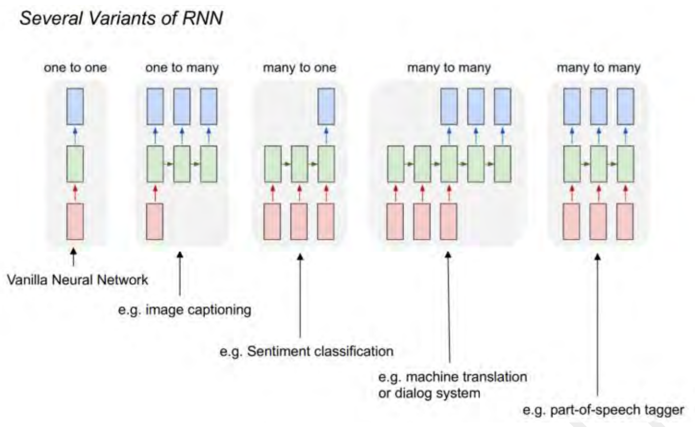


$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

New hidden state A function with parameters W Previous state input

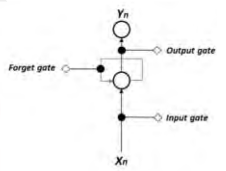
$$\boxed{h_t} = \boxed{tanh}(\underline{W_{hh}}\,\underline{h_{t-1}} + \underline{W_{xh}}\,\underline{x_t} + \underline{b_h})$$

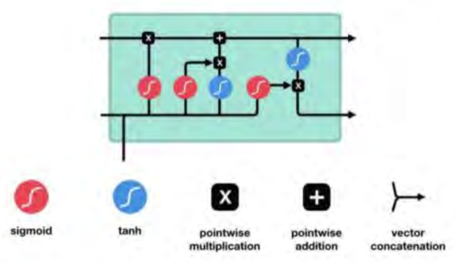
New hidden state A function with parameters W Previous state input



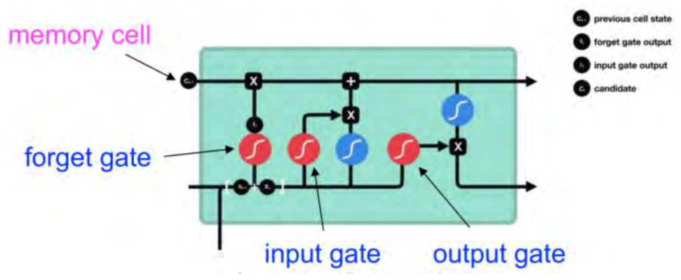
Vanilla RNN cannot effectively handle the long sequence, because of the vanishing and exploding gradient issues

LSTM (Long Short-Term Memory)





- 4 times more parameters than RNN
- Distinguish the important information through **gating**
- Widely used and SOTA in many sequence learning problems



– **Forget Gate**

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

Decides what information should be thrown away or kept Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The **closer to 0 means to forget, and the closer to 1 means to keep**.

– **Input Gate**

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

1. Pass the previous hidden state and current input into a sigmoid function
2. Pass the hidden state and current input into the tanh function to squish values between -1 and 1 to **help regulate the network**
3. Multiply the tanh output with the sigmoid output *sigmoid output will decide which information is important to keep from the tanh output

– **Cell States**

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

the cell state gets pointwise multiplied by the forget vector • take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant • That gives us our new cell state

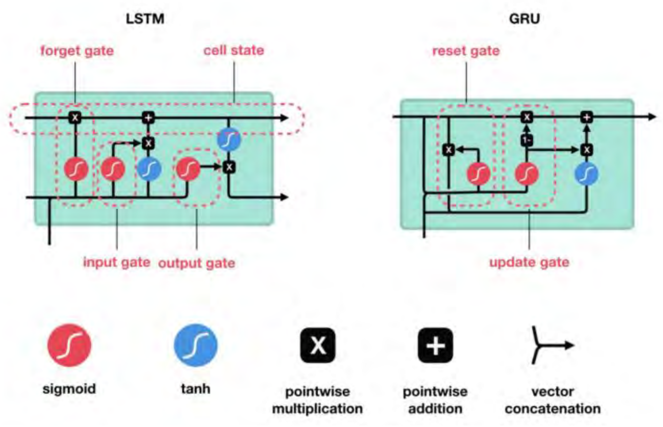
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

– **Output Gate**

$$h_t = o_t * \tanh(C_t)$$

decides what the next hidden state should be. • pass the previous hidden state and the current input into a sigmoid function • pass the newly modified cell state to the tanh function • multiply the tanh output with the sigmoid output to decide what information the hidden state should carry

Gated Recurrent Unit



- GRU first computes an **update gate** based on current input **word vector** and **hidden state**
- Compute reset gate similarly but with different weights
  - If reset gate unit is ~0, then this ignores previous memory and only stores the new word information
- Final memory at time step combines current and previous time steps

L9 Clustering (p606)

Clustering (a.k.a. Unsupervised learning)

The process of grouping the data into clusters so that the data objects (examples) are:

• similar to one another within the same cluster

• dissimilar to the objects in other clusters

Application

Image Discretization

Image Segmentation

Measuring Similarity/Distance

The goal of clustering is “putting nearby points (small distance) into the group”

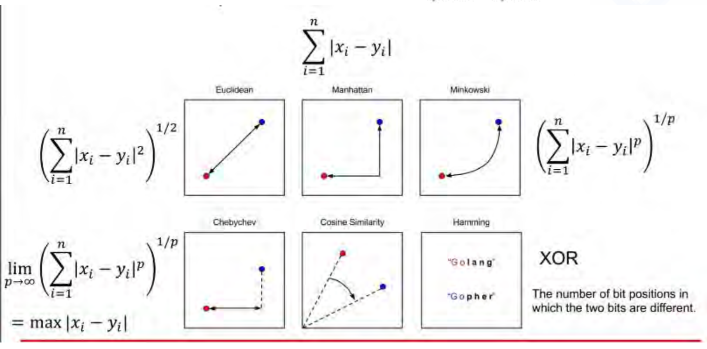
How to measure the distance? • Euclidean distance • Manhattan distance • Cosine distance

Euclidean Distance $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Manhattan Distance d =|x2-x1|+|y2-y1|

Cosine Distance $\frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$


$\sum_{i=1}^{n} |x_i - y_i|$
Euclidean
Manhattan
Minkowski
$\left(\sum_{i=1}^{n} |x_i - y_i|^2\right)^{1/2}$
$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$
Chebychev
Cosine Similarity
Hamming
$\lim_{p\to\infty}\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p} = \max |x_i - y_i|$
"Golang"
"Gopher"
XOR
The number of bit positions in which the two bits are different.

Partitioning Clustering

Partitioning clustering are clustering methods that require the analyst to **specify the number of clusters** to be generated. The best-known family of partitioning clustering algorithms is **k-means clustering**.

k-means clustering Procedure

(p622, L9 p 23)

Complexity is O(n * K * I * d)

- n = number of points, K = number of clusters, I = number of iterations, d = number of attributes

O(n) represents the complexity of a function that increases linearly and in direct proportion to the number of inputs.

K-means convergence (Stopping) Criterion

•no (or minimum) re-assignments of data points to different clusters, or

• no (or minimum) change of centroids, or

• minimum decrease in the sum of squared error (SSE)

$$SSE = \sum_{j=1}^{K} \sum_{x \in C_j} d(x, m_j)^2$$

- C_j is the jth cluster,
- m_j is the centroid of cluster C_j (the mean vector of all the data points in Cj)
- $d(x, m_j)$ is the (Euclidian) distance between data point x and centroid m_j.

Choosing Initial Centroids

perform multiple runs, each with a different set of randomly chosen initial centroids, and then select the set of clusters with the minimum SSE.

K-means++

```
1:  For the first centroid, pick one of the points at random.
2:  for i=1 to number of trials do
3:  Compute the distance, d(x), of each point to its closest centroid.
4:  Assign each point a probability proportional to each point's d(x)2.
5:  Pick new centroid from the remaining points using the weighted probabilities.
6:  end for
```

Handling Empty Clusters

•Choose the point that is **farthest away** from any current centroid. If nothing else, this eliminates the point that currently contributes most to the total squared error.

* A K-means++ approach could be used as well.

• Choose the **replacement centroid** at random from the cluster that has the highest SSE.

* • This will typically split the cluster and reduce the overall SSE of the clustering. If there are several empty clusters, the above can be repeated several times.

Outliers

when outliers are present, the resulting cluster centroids (prototypes) are typically not as representative as they otherwise would be and thus, the SSE will be higher.

use approaches that remove outliers before clustering.

Updating Centroids Incrementally

Bisecting K-means

K-means Strengths and Weaknesses

Strengths : K-means is simple and can be used for a wide variety of data types. quite efficient, even though multiple runs are often performed. Some variants, including bisecting K-means, are even more efficient, and are less susceptible to initialization problems.

Weaknesses: not suitable for all types of data cannot handle nonglobular clusters or clusters of different sizes and densities, although it can typically find pure subclusters if a large enough number of clusters is specified. also has trouble clustering data that contains outliers. **Solution**: Outlier detection and removal can help significantly in such situations.

Gaussian Mixture Models (EM Clustering)

•Assume that the data generating process is a mixture of Multivariate Normals

• Assume π_k is the weight of the k-th Multivariate Normals in the mixture π_k in[0,1], sum = 1

• Assume that all observations from cluster k are drawn randomly from a MVN(μ_k, Σ_k) distribution

Procedures:

Start by fixing k at some value (how many dists you assume

• Calculate the probability that each point belongs to each distribution

• Use these probabilities to compute a new estimate for the parameters

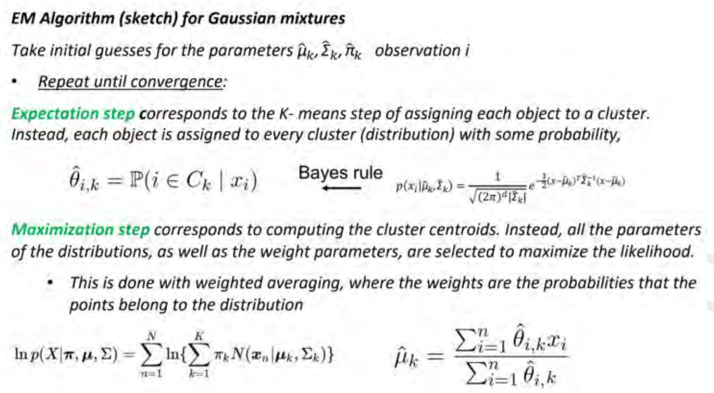
EM approach

1. Guess some cluster centers
2. Repeat until converged
   1. E-Step: assign points to the nearest cluster center
   2. M-Step: set the cluster centers to the mean

the "E-step" or "Expectation step" is so-named because it involves **updating our expectation** of which cluster each point belongs to.

The "M-step" or "Maximization step" is so named because it involves **maximizing some fitness** function that defines the location of the cluster centers—in this case, that maximization is accomplished by taking a **simple mean of the data in each cluster**.

Expectation step & Maximization step



EM Algorithm (sketch) for Gaussian mixtures

Take initial guesses for the parameters $\hat{\mu}_k, \hat{\Sigma}_k, \hat{\pi}_k$ observation i

- *Repeat until convergence:*

Expectation step corresponds to the K- means step of assigning each object to a cluster. Instead, each object is assigned to every cluster (distribution) with some probability,

$$\hat{\theta}_{i,k} = \mathbb{P}(i \in C_k \mid x_i) \quad \xleftarrow{\text{Bayes rule}} \quad p(x_i|\hat{\mu}_k, \hat{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^d|\hat{\Sigma}_k|}} e^{-\frac{1}{2}(x-\hat{\mu}_k)^T\hat{\Sigma}_k^{-1}(x-\hat{\mu}_k)}$$

Maximization step corresponds to computing the cluster centroids. Instead, all the parameters of the distributions, as well as the weight parameters, are selected to maximize the likelihood.

- *This is done with weighted averaging, where the weights are the probabilities that the points belong to the distribution*

$$\ln p(X|\boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k N(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \Sigma_k)\} \qquad \hat{\mu}_k = \frac{\sum_{i=1}^{n} \hat{\theta}_{i,k} x_i}{\sum_{i=1}^{n} \hat{\theta}_{i,k}}$$

The mickey shape cluster : • GMM's do better on this example because they essentially allow for a **data-adaptive notion of distance** when assigning points to centroids

• i.e., In the original data, we have 2 clumps with small variance, and one clump with large variance

• K-means can't capture this added information

Hierarchical Clustering

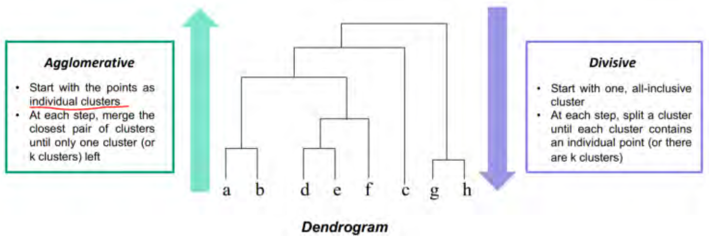
Why use Hierarchical Clustering?

Do **not** have to assume any particular number of clusters

- Any desired number of clusters can be obtained by 'cutting' the dendrogram at the proper level

They may correspond to **meaningful taxonomies**

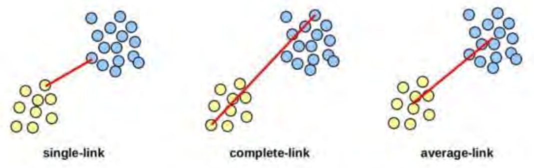
- Example in biological sciences (e.g., animal kingdom, phylogeny reconstruction, …)

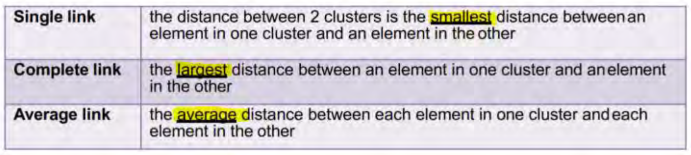


Agglomerative
• Start with the points as individual clusters
• At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
a
b
d
e
f
c
g
h
Dendrogram
Divisive
• Start with one, all-inclusive cluster
• At each step, split a cluster until each cluster contains an individual point (or there are k clusters)

Agglomerative (Bottom-up) Clustering Algorithm

Important! Distance Metrics





Single link	the distance between 2 clusters is the smallest distance between an element in one cluster and an element in the other
Complete link	the largest distance between an element in one cluster and an element in the other
Average link	the average distance between each element in one cluster and each element in the other

Strengths and Weaknesses

Strengths: typically used because the underlying application, e.g., creation of a taxonomy, requires a hierarchy. some studies have suggested that these algorithms can produce betterquality clusters

Weaknesses: are expensive in terms of their computational and storage requirements. The fact that all merges are final can also cause trouble for noisy, high-dimensional data, such as document data.

Solution: In turn, these two problems can be addressed to some degree by first partially clustering the data using another technique, such as K-means.

Tutorial:

two **assumptions** are the basis of the k-means model

1)The "cluster center" is the arithmetic mean of all the points belonging to the cluster.

2)Each point is closer to its own cluster center than to other cluster centers.

Issues using the **expectation–maximization** algorithm

a) The number of clusters, it cannot learn the number of clusters from the data

b) K-means is limited to linear cluster boundaries

DBSCAN

Partitioning (e.g. K-means) are designed to find spherical-shaped clusters. It have difficulty finding clusters of arbitrary shape, such as S shape and oval clusters.

Features and Characteristics of Density-based Clustering

Discover clusters of arbitrary shape • Handle noise • One scan (only examine the local region to justify density) • Need **density parameters** as termination condition

Density = number of points within a specified radius (Eps)

Two parameters:

• Epsilon (Eps - Ɛ): Maximum radius of the neighbourhood

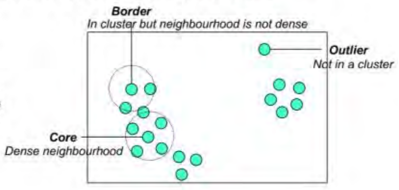
• Minimum Points (MinPts): Minimum number of points in the Eps-neighourhood of a point

If the radius is too large, then all points are core points
• If the radius is too small, then all points are outliers

The center-based approach to density allows us to classify a point as being

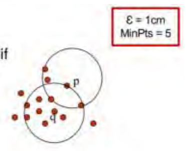
- A **core point** if it has at least a specified number of points (MinPts) within Eps
  - These are points that are at the interior of a cluster
  - Counts the point itself
- A **border point** is not a core point, but is in the neighborhood of a **core point**
- A **noise/outlier point** is any point that is not a core point or a border point

Any two core points that are close enough—within a distance *Eps* of one another—are put in the same cluster.
Any border point that is close enough to a core point is put in the same cluster as the core point. (Ties need to be resolved if a border point is close to core points from different clusters.)
Noise points are discarded.



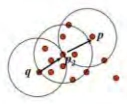
Directly density-reachable:

- A point p is **directly density-reachable** from a point q with Eps (ε), MinPts if
  - p belongs to $N_{Eps}(q)$
  - **core point** condition: $|N_{Eps}(q)| \geq MinPts$



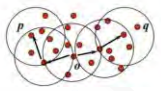
Density-reachable:

- A point p is **density-reachable** from a point q with Eps (ε), MinPts if there is a chain of points $p_1, \ldots, p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i



Density-connected:

- A point p is **density-connected** from a point q with Eps (ε), MinPts if there is a point o such that both p and q are density-reachable from o with Eps (ε), MinPts



Algorithm
Arbitrarily select a point p
• Retrieve all points **density-reachable** from p with Eps (ε), MinPts If p is a core point, a cluster is formed If p is a border point, no points are density-reachable from p, and DBSCAN visits the next point of the database
• Continue the process until all of the points have been processed

Computational complexity

- If a spatial index is used (e.g., kd-trees), the computational complexity of DBSCAN is O(nlogn), where n is the number of database objects
- Otherwise, the complexity is $O(n^2)$

Method for selecting DBSCAN parameters

- *Decide how many points you want in a dense region: MinPts.*
  (Suppose we want core points to have at least k ε-neighbors)
- *Determine the distance from each point to its k-th nearest neighbor, called the **kdist***
- *For points that belong to some cluster, the value of **kdist** will be small*
  [if k is not larger than the cluster size]
- *For points that are not in a cluster, such as noise points, the **kdist** will be relatively large*

Grid-based Clustering Method
• Efficiency and scalability: # of cells < # of data points
• Uniformity: Uniform but hard to handle highly irregular data distributions
• Locality: Limited by pre-defined cell sizes, borders, and the density threshold
• Curse of dimensionality: Hard to cluster high-dimensional data

STING
CLIQUE

Clustering Evaluation: External
1. Cluster homogeneity • The purer, the better
2. Cluster completeness • Assign objects belonging to the same category in the ground truth to the same cluster
3. Rag bag better than alien • Putting a heterogeneous object into a pure cluster should be penalized more than putting it into a rag bag (i.e., “miscellaneous” or “other” category)
4. Small cluster preservation • Splitting a small category into pieces is more harmful than splitting a large category into pieces

1. Matching-based measures • Purity, maximum matching, F-measure
2. Entropy-Based Measures • Conditional entropy • Normalized mutual information (NMI)
3. Pairwise measures • Four possibilities: True positive (TP), FN, FP, TN • Jaccard coefficient, Rand statistic, Fowlkes-Mallow measure

Silhouette coefficient as an internal measure

Tutorial:
Gaussian Mixture Model
assumption : there is a hidden model driving the distribution of the samples
each sample is drawn from one of a nite number of Gaussian distributions: k being the number of Gaussians.

Expectation step: statistically we dene an expected value of the likelihood function of , with respect to the conditional distribution of Z given X. **Maximisation** step: maximise the likelihood function from the E step

A Gaussian Mixture Model can tackle this kind of data, since each hidden variable has not just a mean (the centre) but a variance in each direction. The E step is also more sophisticated as it assigns a weighted conditional probability of each datapoint belonging to each cluster (the responsibilities). The M step still maximises the likelihood by adjusting the parameters of the contributing Gaussians.

DBSCAN addresses below issues:
minimal requirements of domain knowledge to determine the input parameters,
discovery of clusters with arbitrary shape,
good eciency on large databases.

CLIQUE
two parameters to choose:
intervals is the **number** of divisions of the space in each dimension,
and threshold is the **minimum number of points** that a subspace cell should contain to consider its points as non-outliers.

Reinforcement Learning
Reinforcement learning is a computational approach to learning from interaction.

policy π* that maximizes the sum of rewards.

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right] \quad \text{with } s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

Bellman equation

Q-learning
We start with a Q table which stores a value for each observation_space, action space pair. Q will represent the Quality of the action at that observation, with respect to the rewarded end goal.
During training quality values will be learned for each action in each state

three hyperparameters:
$\boldsymbol{\alpha}$ a learning rate
$\boldsymbol{\gamma}$ a discount factor
$\boldsymbol{\varepsilon}$ a weighting between exploration and exploitation