Neural Methods for NLP

Master LiTL --- 2021-2022

chloe.braud@irit.fr

https://github.com/chloebt/m2-litl-students

Goals

- Understand what are Neural Networks and why everybody is so crazy about them
- Being able to train and evaluate a deep learning model
    - understand the hyper-parameters, being able to optimize a DL model
    - understand the varied architectures, their underlying motivations, their use
    - understand the input: what are embeddings? Focus on word embeddings, contextual and non-contextual
    - how to build a model for a specific application: classification, sequence labelling, generation
- Having an idea of the limitations and current challenges

Practical sessions / Assignments:

- Library: PyTorch
- Environment: Google Collab (you can also download the notebook and use Jupyter...)

Bibliography and resources

Personal note: I have a PhD in Sciences du Langage. I’m now a researcher in Computer Science. I’m in between Human and Computer science, I’m interested by both aspects. Note that I’m not going to be very technical in the details. There are many many good resources for more details.

- *Neural Network Methods for NLP*, Y. Goldberg
- Online courses:
  - Neural Nets for NLP, G. Neubig,
  - Stanford courses with C. Manning (official website: https://web.stanford.edu/class/cs224n/)
- J. Eisenstein course: https://github.com/jacobeisenstein/gt-nlp-class/blob/master/notes/eisenstein-nlp-notes.pdf

I used other resources to build this course, I’ll try to give all the sources used.

(Tentative) Schedule

- S47 - C1: 23.11 13h30-15h30 (2H) → ML reminder + TP1
- S48 - C2: 30.11 13h30-15h30 (2H) → Intro DL + TP2
- S49 - C3: 07.12 13h30-16h30 (**3H**) → Training a NN + TP3
- S50 - C4: 14.12 13h30-16h30 (**3H**) → Embeddings + TP4
- S51 - 21.12: break
- S52 - 28.12: break
- S1 - C5: 04.01 10h-12h (2H) → [Start projects]
- S2 - C6: 14.01 10h-12h (2H) → CNN
- S3 - C7: 18.01 13h-16h (**3H**) → RNN + TP5
- S4 - C8: 25.01 13h-16h (**3H**) → NLP applications and NN + TP6
- S5 - C9: 01.02 10h-12h (2H) → [Work on projects]
- S6 - C10: 15.02 10h-12h (2H) → Current challenges

14.02: Assignments due (code + report)

Neural Methods for NLP

Course 1: Machine Learning (reminder)

Master LiTL --- 2021-2022

chloe.braud@irit.fr

https://github.com/chloebt/m2-litl-students

Neural methods for NLP

- 1980's: Symbolic NLP
  - rule-based approach, hand-written rules
  - advantages: based on linguistics expertise, very precise
  - inconvenients: lack of coverage, time consuming
- 1990's: 'Statistical' NLP
  - learn rules automatically = (mostly linear) functions, with high-dimensional, sparse feature vectors
  - large annotated corpora
  - handcrafted features
  - rather fast to train, still good baselines
- ≃ 2010: 'Neural' NLP
  - combine linear and non-linear functions, over dense inputs
  - (very) large annotated corpora and very large unannotated corpora
  - improved performance (in general), no feature engineering
  - harder to interpret ("black box")

Neural methods for NLP

- 1980’s: Symbolic NLP
    - rule-based approach, hand-written rules
    - advantages: based on linguistics expertise, very precise
    - inconvenients: lack of coverage, time consuming
- **1990’s: ‘Statistical’ NLP**
    - learn rules automatically = (mostly linear) functions, with high-dimensional, sparse feature vectors
    - large annotated corpora
    - handcrafted features
    - rather fast to train, still good baselines
- ≃ 2010: ‘Neural’ NLP
    - combine linear and non-linear functions, over dense inputs
    - (very) large annotated corpora and very large unannotated corpora
    - improved performance (in general), no feature engineering
    - harder to interpret (“black box”)

Machine Learning for NLP

Applications:

- NLP applications: spam filtering, spell checking, machine translation, summarization, web search, recommendation systems, sentiment analysis, hate speech detection…
- NLP tasks: sentence splitting, tokenization, POS tagging, NER, syntactic parsing, semantic parsing, discourse parsing, event identification, detecting language change, representation learning, speech recognition...

Data investigation:

- Looking at how works your model could help understanding your data/problem:
    - e.g. **Age or Gender bias in models**: *Gender Bias in Part-of-Speech Tagging and Dependency Parsing Data*, A. Garimella, C. Banea, D. Hovy, & R. Mihalcea. ACL 2019
- (Linguistic) Hypothesis checking
    - e.g. Scientific fraud: **specific writing style?** *Is writing style predictive of scientific fraud?*, C. Braud and A. Søgaard. EMNLP 2017
- For 'fun': e.g. see T. Van de Cruys' book of **poetry generated** via ML

Content

Statistical Learning

1. Learning problems
2. Workflow and terminology
3. Linear classification
4. Representation function
5. Linear Algebra?

Practical session 1: implement a ML model with Scikit + Linear algebra?

Content

ML ≈ DL

1. **Learning problems**
2. **Workflow and terminology**
3. Linear classification
4. Representation function
5. Linear Algebra?

Practical session 0: implement a ML model with Scikit (see TP3 < Tim), Linear algebra?

Content

ML ≈ DL

ML ≠ DL

1. Learning problems
2. Workflow and terminology
3. **Linear classification**
4. **Representation function**
5. Linear Algebra?

Practical session 0: implement a ML model with Scikit (see TP3 < Tim), Linear algebra?

Learning problems and scenarios

Most common learning problem in NLP: **classification**

Most common scenario: **supervised** learning

→ using pre-trained word embeddings is in fact doing semi-supervised learning

The different tasks

- **Classification:** predict a categorical label for each item
  - single label: each instance is assigned a single label
    - binary: 2 labels, e.g. an email is either a spam or not
    - multi-class: > 2 labels, e.g. sentiment is either positive, negative or neutral
  - multi-label: each instance is assigned multiple labels, e.g. *The Lord of the Ring* is classified as: Adventure, Fantasy, Drama
- **Sequence labeling / structured prediction**: predict a categorical label for each member of a sequence
  - e.g. POS tagging, NER...
  - can be seen as performing independent classification tasks on each item
  - but performance are improved when taking into account the dependence between the elements
- **Regression**: Predict a **real value** for each item
  - e.g.: prediction of stock values, variations of economic variables, house prices..
  - rarer for NLP, but e.g. data with depression “scores” (DAIC)

The different tasks

- **Clustering**: Partition items into homogeneous regions
  - kind of classification but without classes known a priori
  - can be useful if you don't have manual labels or want to explore your data
  - often used for very large data sets
    - e.g.: in social network analysis, attempt to identify “communities” within large groups of people.
- **Ranking**: Order items according to some criterion (e.g. Web search)
- Dimensionality reduction or manifold learning: Transform an initial representation of items into a lower-dimensional representation (for pre- processing or visualisation)

The learning scenarios

Depend on the **annotations** you have:

Supervised learning:

- we have a set of **labeled** examples as training data
- most common for classification and regression

The learning scenarios

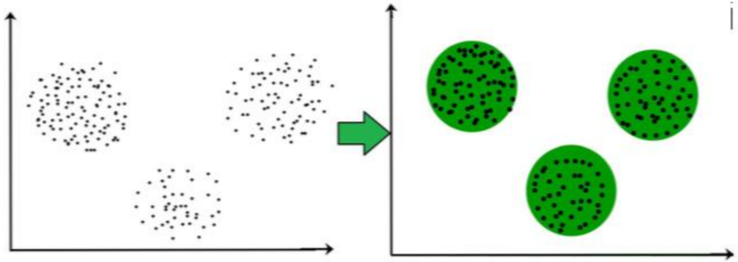
Depend on the **annotations** you have:

Supervised learning:

- we have a set of **labeled** examples as training data
- most common for classification and regression

Unsupervised learning:

- we only have **unlabeled** training data
    - e.g.: Clustering and dimensionality reduction
    - often hard to evaluate



The learning scenarios

Semi-supervised learning: the training sample consists of both **labeled and unlabeled** data

The learning scenarios

Semi-supervised learning: the training sample consists of both **labeled and unlabeled** data

```
easier to obtain!
```

The learning scenarios

Semi-supervised learning: the training sample consists of both **labeled and unlabeled** data

easier to obtain!

Very hard in practice, but many variations:

- **labeled + automatically labeled** data
  - e.g. sentiment analysis with smileys as (noisy) labels
- **labeled + external resource** giving constraints
  - e.g. POS tagging with a dictionary
- **labeled + labeled** data for another task
  - multi-task learning
- **labeled + unlabeled**: pre-trained word embeddings

Especially used for **transfer learning** / domain adaptation:

- e.g. building a model for a new language or for a new genre of texts

Supervised classification

Supervised classification:

- the most common scenario for NLP (with supervised structured prediction)
- **supervised**: input = **labeled** data points
- **classification**: assign a category/class to each item, e.g.
  - is a word a VERB or a NOUN?
  - Is a document talking about Sport or Politics or Economy?

Binary vs Multi-class:

2 classes (e.g. positive/negative, comedy/drama) *vs* more than 2 classes (e.g. positive/negative/neutral, any genre)

Distinction that has an impact

- on the algorithm: various strategies to deal with MC problems
- on evaluation: various metrics
- but rather transparent with scikit: algorithms/functions can be used for both binary and MC problems

Machine Learning: workflow and terminology

The different steps when doing machine learning:

- (1) preparing data,
- (2) learning and tuning,
- (3) predicting and evaluating

Terminology:

- input
- model and parameters
- train/dev/test sets

Machine Learning

Start with:

- a set of labelled data = data points + (gold) labels
- a function that could be used to compute a label for a data point

Learning a model:

- Goal: try to get the best function, i.e. that finds the right/gold label
- Process: iterate over the examples, and adjust the parameters of the function to avoid errors

Evaluating the model:

- Goal: evaluate the performance of the learned model
- Process: once the model is learned / trained, make predictions over unseen data and compute some performance metrics

Machine Learning

Start with:

- a set of labelled data = data points + (gold) labels → **supervised setting**
- a function that could be used to compute a label for a data point → **classification**

Learning a model:

- Goal: try to get the best function, i.e. that finds the right/gold label
- Process: iterate over the examples, and adjust the parameters of the function to avoid errors

Evaluating the model:

- Goal: evaluate the performance of the learned model
- Process: once the model is learned / trained, make predictions over unseen data and compute some performance metrics

Machine Learning

Start with:

- a set of labelled data = **data points + (gold) labels** → **supervised setting**
- a **function** that could be used to compute a label for a data point → **classification**

Learning a model:

- Goal: try to get the best function, i.e. that finds the right/gold label most often
- Process: iterate over the examples, and adjust the **parameters** of the function to avoid errors

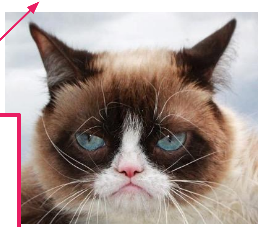
Evaluating the model:

- Goal: evaluate the **performance** of the learned model
- Process: once the model is learned / trained, make **predictions** over unseen data and compute some performance metrics

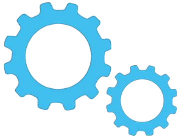
Supervised classification

Data preparation | **Training** | **Evaluating**

Input Data / Training set
(Labeled examples)

Data points x	**gold Labels y**
	Cat
	Sandwich

Represented by some features



learning some function f with parameters

Test set

output

Cat

Use f to predict a label

20/20
Très bien !

Compute score using some performance metrics

Workflow: (1) Data preparation

- **Define the problem**: task? labels?
- Collect (labeled) data: datasets available online, scikit toy datasets, scrap data…
- Randomly partition your data, i.e. shuffle then split:
    - Train / dev / test: e.g. 80-10-10 (or use pre-defined split), in gal train > test
    - Train / test + cross-fold on training set for tuning
- Data description: **you need to know your data!**
    - Number of training/evaluation examples
    - Class distribution: number of examples per class
    - Vocabulary, language, genre, etc…
- **Feature extraction/engineering**: critical step, reflects prior knowledge
    - Possibly linguistic pre-processing: POS tagging, parsing, NER, etc...
    - Vectorization, normalization

Workflow: (1) Data preparation

- **Define the problem**: task? labels?
- Collect (labeled) data: datasets available online, scikit toy datasets, scrap data…
- Randomly partition your data, i.e. shuffle then split:
  - Train / dev / test: e.g. 80-10-10 (or use pre-defined split), in gal train > test
  - Train / test + cross-fold on training set for tuning
- Data description: **you need to know your data!**
  - Number of training/evaluation examples
  - Class distribution: number of examples per class
  - Vocabulary, language, genre, etc…
- **Feature extraction/engineering**: critical step, reflects prior knowledge
  - Possibly linguistic pre-processing: POS tagging, parsing, NER, etc...
  - Vectorization, normalization

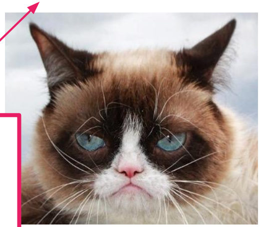
Supervised classification

Data preparation

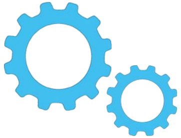
Training

Evaluating

Input Data / Training set
(Labeled examples)

Data points x	**gold Labels y**
	Cat
	Sandwich

Represented by some features



learning some function f with parameters

Test set

Use f to predict a label

output

Cat

20/20
Très bien !

Compute score using some performance metrics

Workflow: (2) Learning + Tuning

- **Choose a learning algorithm**: crucial, especially if training is long
    - Advice: try first with a fast algorithm
- Train: at each training step,
    - update values for the parameters
    - the values for the hyper-parameters are fixed
- Tune:
    - identify the tunable hyper-parameters
    - search the best values for the hyper-parameters

Learning algorithms for classification

- Naive Bayes
- **Linear classifiers:**
    - perceptron
    - passive-aggressive
    - Logistic Regression aka MaxEnt
    - linear SVM
- Non linear SVM
- Neural networks

See [the doc on supervised learning]

See [the tutorial: working with text]

Learning algorithms for classification

- Naive Bayes
- **Linear classifiers**:
  - perceptron
  - passive-aggressive
  - Logistic Regression aka MaxEnt
  - linear SVM
- Non linear SVM
- Neural networks

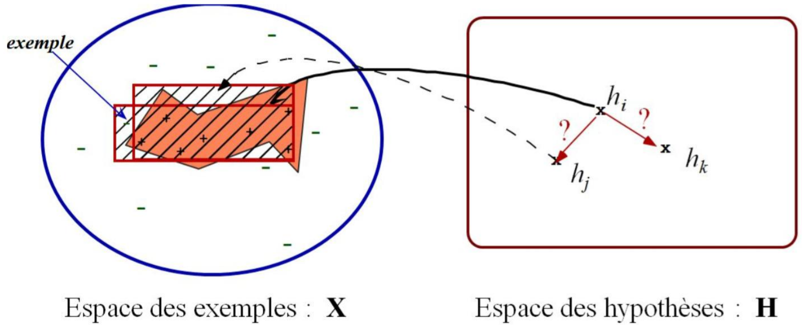
See the doc on supervised learning

See the tutorial: working with text

Workflow: (2) Learning + Tuning

- Choose a learning algorithm: crucial, especially if training is long
    - Advice: try first with a fast algorithm
- **Train**: at each training step,
    - update values for the **parameters**
    - the values for the hyper-parameters are fixed
- Tune:
    - identify the tunable hyper-parameters
    - search the best values for the hyper-parameters

Model and learning function



Remember that we are learning a function:

- **Target function**: the true function ***f*** we want to learn / approximate.
- **Hypothesis** (sometimes called *model*): a function ***h*** that we hope is similar to the target.

Apprentissage supervisé :

- À partir de l'échantillon d'apprentissage $S = \{(x_i , y_i)\}_{1,m}$
- on cherche une loi de dépendance sous-jacente

Par exemple une fonction ***h*** aussi proche possible de ***f*** (fonction cible) avec *f* telle que $y_i = f(x_i)$

Learning a model

Remember that we want to avoid errors, find the best hypothesis:

- **Loss function**: measures the difference, or loss, between a predicted label and a true label.
    - $L: Y \times \hat{Y} \rightarrow \mathbb{R}$.
    - zero-one loss, squared loss, hinge loss...

The learning algorithm tries to get the smaller possible loss on the examples it is given:
- i.e. if the predicted label is wrong, it modifies the values of the weights assigned to the features

The loss should decrease with learning!

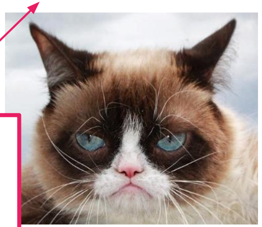
Workflow: (2) Learning + Tuning

- Choose a learning algorithm: crucial, especially if training is long
    - Advice: try first with a fast algorithm
- Train: at each training step,
    - update values for the parameters
    - the values for the hyper-parameters are fixed
- **Tune**:
    - identify the tunable **hyper-parameters**
    - search the best values for the hyper-parameters

Supervised classification

Data preparation

Input Data / Training set
(Labeled examples)

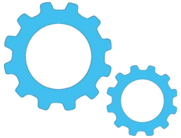
Data points x	**gold Labels y**
	Cat
	Sandwich

Represented by some features

Training

Tuning

setting hyper-parameters



learning some function f with parameters

Evaluating

Test set

output

Cat

Use f to predict a label

20/20
Très bien !

Compute score using some performance metrics

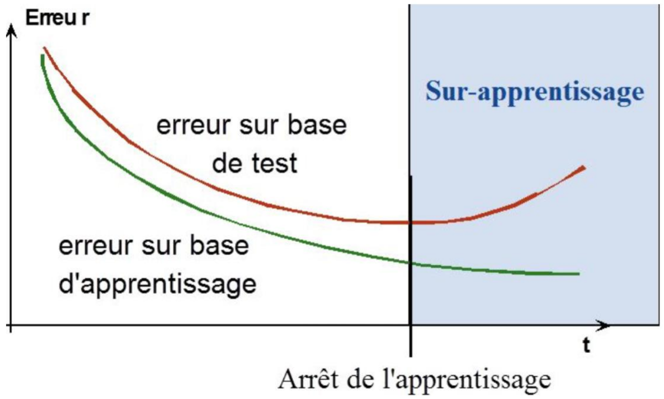
Tuning / Optimization

- We **learn the parameters** of the model (or weights, ***w*** or ***θ***)
- We need to **set values for the hyper-parameters** associated to the learner
    - tuning = searching for the best values for hyper-parameters e.g. smoothing, regularization strength etc

Tuning:

- identify the hyper-parameters of your model (SciKit: *estimator.get_params()*)
- choose the *right* performance metrics to optimize
- choose the right procedure → always set apart a test set:
    - use a **validation / development set**:
        - define a set of possible values for each hyper-parameter
        - train a model for each subset of values and evaluate on the dev set
        - compare the results: keep the model giving the best score on dev
        - evaluate (only) this model on the test set
    - *n*-**fold cross-validation**: esp. when small amount of data
        - very easy with scikit: **grid-search cross-validation**

Learning is not memorizing



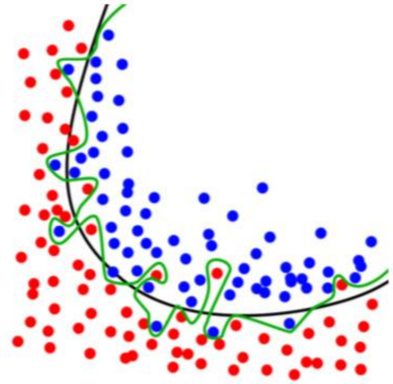
Consistent model:

- **no error on train** set
- but **poor performance on test**,

i.e. memorize the data, unable to generalize

Overfitting

- Very complex decision surface: no generalization to unseen data
- Less complex: might generalize better in spite of some errors



"The **green line represents an overfitted model** and the **black line represents a regularized model**. While the green line best follows the data, it is too dependent on the training data" (Mohri)

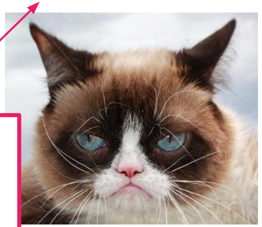
→ Solution = **regularization**: constraining a model to make it simpler

= add a regularization term to minimize the complexity of the model, i.e. an **hyper-parameter** corr. to the strength of the regularization

Supervised classification

Data preparation

Input Data / Training set
(Labeled examples)

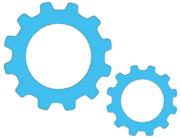
Data points x	**gold Labels y**
	Cat
	Sandwich

Represented by some features

Training

setting hyper-parameters

Tuning



learning some function f with parameters

Evaluating

Test set

output

Cat

Use f to predict a label

20/20
Très bien !

Compute score using some performance metrics

(3) Prediction + Evaluation

Using the final values for parameters and hyper-parameters, **evaluate on test**

- Use your model to **make predictions** on the **unseen** test data (y_{pred})
- **Compute a score** by comparing y_{true} and y_{pred} for each example in the test
- **Compare** to other systems: baselines, state-of-the-art...

It is important to:

- Keep track of the values used (final and tested) for the hyper-parameters for reproducibility!
- Choose a / several relevant **evaluation metrics**
- Propose relevant baselines

Classification metrics

For classification, we mostly use:

- Global scores: accuracy, averaged F1
- Per class scores: precision, recall, F1

Accuracy is the most common metrics:

- fraction of correctly predicted samples
    - e.g. 90 well predicted over 100 examples: accuracy = 90%
- issue esp. with imbalanced data,
    - e.g. Cancer detection: 90 non cancer, 10 cancer is 90% a good score for predicting cancer?
    - we want to predict well the positive class

Evaluating a model

Report one or **several metrics**:

- depend on the setting (binary or multi-class) and task
- classification: Accuracy, Macro/weighted F1, prec/rec/F1 per class

Compare to other systems:

- baselines: simplest feature/algo, dummy classifier (most frequent class)
- state-of-the-art: systems from the literature, reported or reproduced, compare different algorithms
- compare different feature sets
- compare different datasets: prove the robustness of your method over different genres, languages…

Try to **understand your model**:

- Scikit: classifiers have a *coef_* parameters that allows to inspect the weights associated to each feature
    - **eli5**: a library to debug ML models, compatible with Scikit, see the doc
- try to relate observed behaviour to a priori knowledge, esp. linguistic

Input: examples, features and labels

Examples or samples / instances / data points = items of data used for learning or evaluating (*m* examples)

Features = set of attributes associated to an example (*n* features)

A **set of examples**: a **matrix** $\boldsymbol{X}$ of size $m \times n$

- 1 example = 1 row, i.e. a vector $\mathbf{x}$ of size n

Labels = values assigned to examples; for classification:

- General label set: Y of p classes
- Labels for all examples = a list of size m, e.g. y_{true}
- 1 label for 1 example = 1 value y in the list
- **labeled example** = a pair ($\mathbf{x}$, y)

Dataset for ML = $\mathbf{X}$ and $\mathbf{y}_{\mathbf{true}}$

Model, Parameters and hyper-parameters

Model: what we learn is a model of our data

- We sometimes call model the weights learned, i.e. the importance that the model associates with each feature
    - e.g. Sentiment analysis: 'love':+10, 'hate':-42, ‘green’:0…
- Weights are saved in a vector **w** (or θ) of size n(+1)
- Each of these weights is a **parameter**/coefficient of the model

Hyper-parameters (or free parameters, part of the model):

- there could be parameters dependent on the learning algorithm used
- setting the values for these hyper-parameters is called **tuning** the model

Train / Dev / Test sets

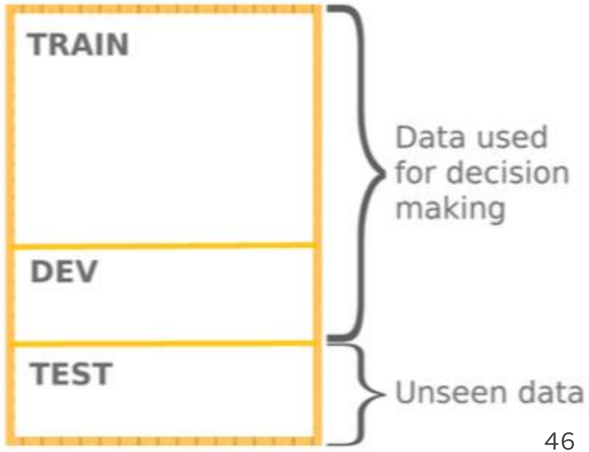
Training sample/set: examples used to train a learning algorithm, to **learn/fit** a model

Development/validation set: examples used to **tune** the hyper-parameters of a learning algorithm

Test/evaluation set: examples used to **evaluate** the performance of a learning algorithm

- The test set is separate from the train/dev sets
- The test set is not made available in the learning stage

Searching for the best model on the test set =



Linear classification

- Binary classification: linear functions, weight matrix and bias
- Reminder: Logistic Regression = linear scores + logistic function
- Loss function

ML $\longrightarrow$ DL: Change here = power of non linearity ; LR performed by each neuron

Why Linear classifiers?

Remember that: ML is about finding a function ***h*** that best approximate the target function ***f***

- Searching over the set of all possible functions is very hard
- We thus restrict ourselves over specific families of functions, the ***hypothesis class*** e.g. the space of all linear functions with d_{in} inputs and d_{out} outputs
    - inject the learner with inductive bias: a set of assumptions about the desired solution
    - facilitate procedures for searching solutions
- The hypothesis class also determines what can and cannot be represented by the learner!

Linear Classifiers

Hypothesis class = high-dimensional linear functions, of the form:

$$f(\mathbf{x}) = \mathbf{W}.\mathbf{x} + \mathbf{b}$$

with $\mathbf{x} \in R^{d\_in}$, $W \in R^{d\_inxd\_out}$, $\mathbf{b} \in R^{d\_out}$

- The task of searching over the space of functions = searching over the space of parameters, i.e. finding the best Θ = **W**, **b**.
- Sometimes, to make the parameterization explicit, we write: f(**x** ; **W**, **b**)
- In binary classification, **w** is a vector

Recall on linear algebra: $\mathbf{W.x} = \Sigma_j\, w_j\, x_j = w_0.x_0 + w_1.x_1 + ... + w_n.x_n\ (+b)$

With *n* features, we have:

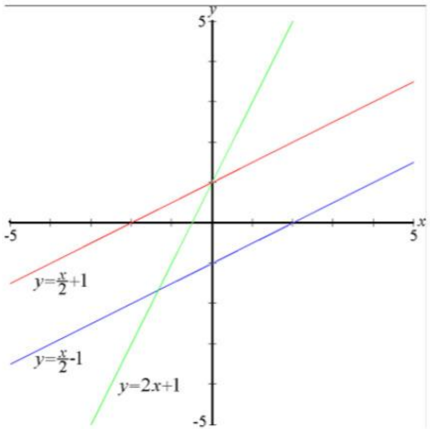
- a data point: $\boldsymbol{x} =< x_0, x_1, ..., x_n >$
- the weights: $\boldsymbol{w} =< w_0, w_1, ..., w_n >$

Linear Classifiers

The decision boundary is a linear function of the input: in the binary case, it's a line (1 dimension / feature), a plane (2 d) or an hyperplane (n+1 d) separating the two classes

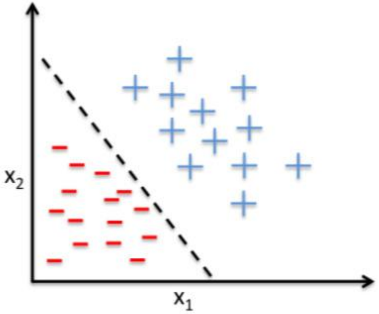
Decision boundary:

$y = w_0 . x_0 + b$: a line



Prediction:

if $\hat{y} = \mathbf{w} \cdot \mathbf{x} > 0$, predict positive class.

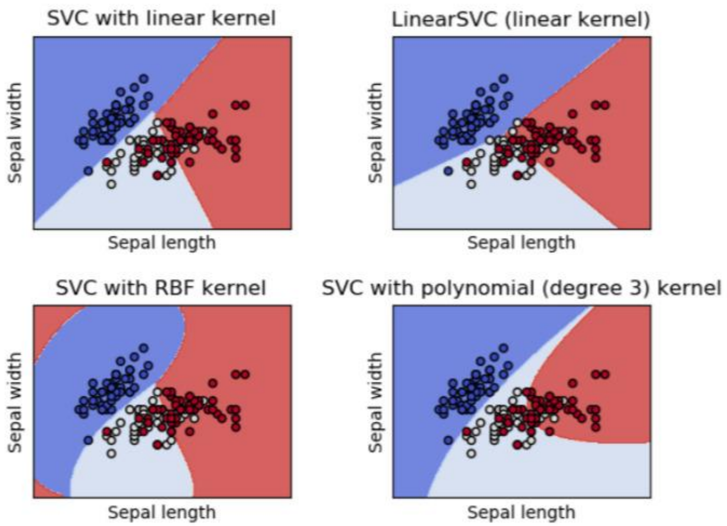


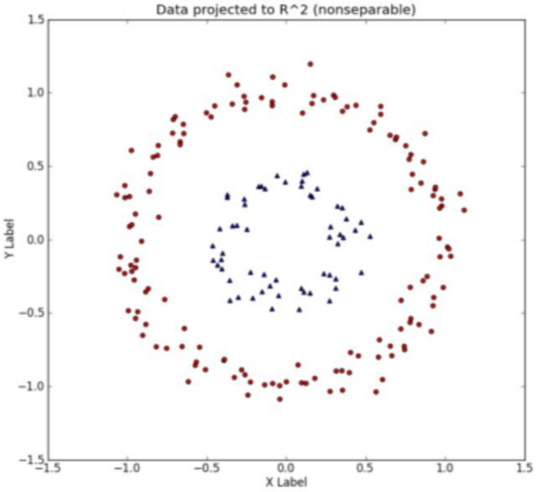
Example of a linear decision boundary for binary classification.

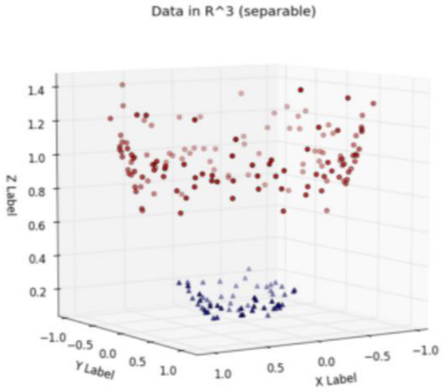
Introducing non-linearity

SVM with non-linear kernel

- mapping of the original input feature space to a **higher-dimensional** feature space,
- with the hope that data may be **linearly separable** in this new space





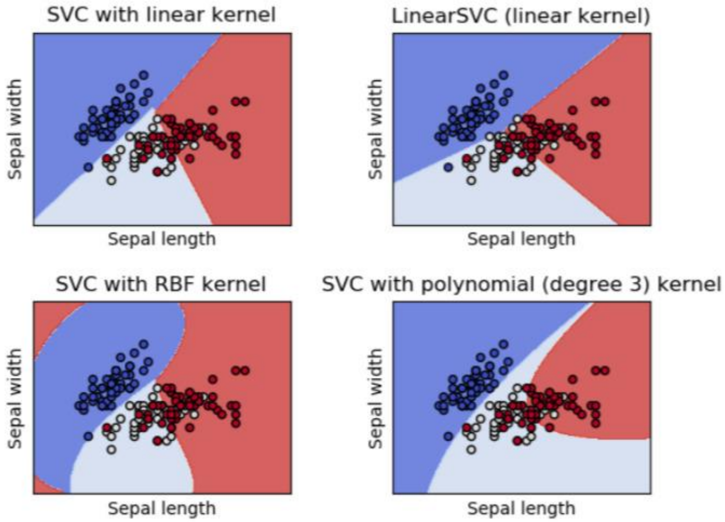


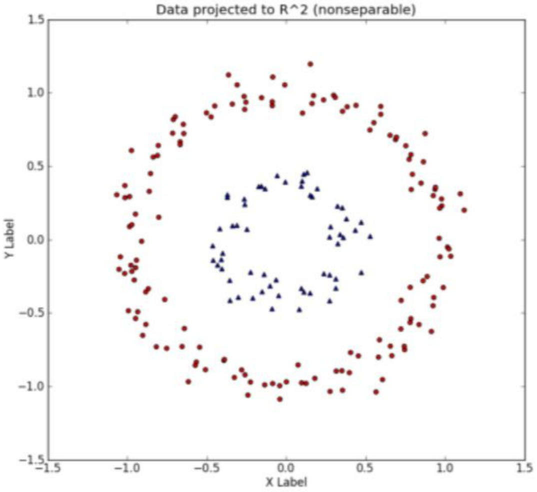
Introducing non-linearity

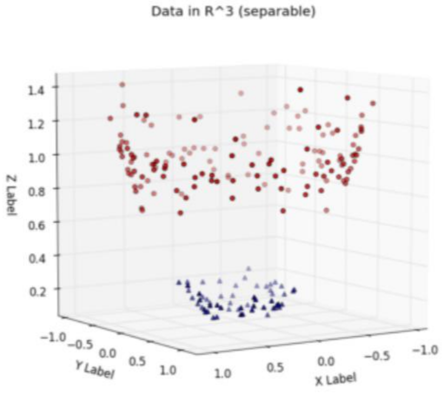
SVM with non-linear kernel

Neural Network: keep non-linearity and transformation of the input space.

- mapping of the original input feature space to a **higher-dimensional** feature space,
- with the hope that data may be **linearly separable** in this new space







Loss functions: training as optimization

The goal of the algorithm is to return a function **f()** that accurately maps input examples to their desired labels

- i.e. a function such that the predictions **ŷ = f(x)** over the training set are accurate
- the ***loss function*** is used to quantify the **loss suffered when predicting ŷ while the true label is y**
    - **L(ŷ,y)**: assigns a numerical score (a scalar) to a predicted output **ŷ** given the true expected output **y**
- Should be bounded from below: minimum attained only for cases where the prediction is correct
- The parameters **W** and **b** are set to minimize **L** (usually, the sum of the losses over the training examples)

$$\mathbf{L(\Theta) = 1/n \sum_{i=1..n} L(f(xi\ ;\ \Theta),\ yi)}$$

Thus training correspond to find this minimum:

$$\mathbf{\hat{O} = argmin_{\Theta}\ L(\Theta) = argmin_{\Theta}\ 1/n \sum_{i=1..n} L(f(xi\ ;\ \Theta),\ yi)}$$

Loss functions: training as optimization

The goal of the algorithm is to return a function **f()** that accurately maps input examples to their desired labels

- i.e. a function such that the predictions **ŷ = f(x)** over the training set are accurate
- the ***loss function*** is used to quantify the **loss suffered when predicting ŷ while the true label is y**
    - **L(ŷ,y)**: assigns a numerical score (a scalar) to a predicted output **ŷ** given the true expected output **y**
- Should be bounded from below: minimum attained only for cases where the prediction is correct
- The **parameters W and b are set to minimize L** (usually, the sum of the losses over the training examples)

$$\mathbf{L(\Theta) = 1/n \sum_{i=1..n} L(f(xi\ ;\ \Theta),\ yi)}$$

Thus training correspond to find this minimum:

$$\mathbf{\hat{O} = argmin_{\Theta}\ L(\Theta) = argmin_{\Theta}\ 1/n \sum_{i=1..n} L(f(xi\ ;\ \Theta),\ yi)}$$

Minimizing errors

Loss functions: training as optimization

The goal of the algorithm is to return a function **f()** that accurately maps input examples to their desired labels

- i.e. a function ŷ such that the predictions **ŷ = f(x)** over the training set are accurate
- the ***loss function*** is used to quantify the loss suffered chen predicting **ŷ** while the true label is **y**
    - **L(ŷ,y)**: assigns a numerical score (a scalar) to a predicted output **ŷ** given the true expected output **y**
- Should be bounded from below: minimum attained only for cases where the prediction is correct
- The **parameters W and b are set to minimize L** (usually, the sum of the losses over the training examples)

$$\mathbf{L(\Theta) = 1/n \sum_{i=1..n} L(f(xi\ ;\ \Theta), yi)}$$

Thus training correspond to find this minimum:

Regularization

$$\mathbf{\hat{O} = argmin_{\Theta}\ L(\Theta) = argmin_{\Theta}\ 1/n \sum_{i=1..n} L(f(xi\ ;\ \Theta), yi) + \lambda\ R(\Theta)}$$

Minimizing errors

Loss functions: training as optimization

Same for NN: We'll go back to loss functions later

The goal of the algorithm is to return a function **f()** that accurately maps input examples to their desired labels

- i.e. a function ŷ such that the predictions **ŷ = f(x)** over the training set are accurate
- the ***loss function*** is used to quantify the loss suffered chen predicting **ŷ** while the true label is **y**
  - **L(ŷ,y)**: assigns a numerical score (a scalar) to a predicted output **ŷ** given the true expected output **y**
- Should be bounded from below: minimum attained only for cases where the prediction is correct
- The **parameters W and b are set to minimize L** (usually, the sum of the losses over the training examples)

$$L(\Theta) = 1/n \sum_{i=1..n} L(f(xi\, ;\, \Theta), yi)$$

Thus training correspond to find this minimum:

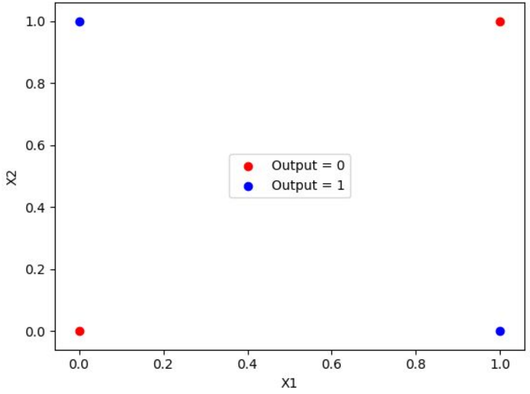
Regularization

$$\hat{O} = \text{argmin}_{\Theta}\, L(\Theta) = \text{argmin}_{\Theta}\, 1/n \sum_{i=1..n} L(f(xi\, ;\, \Theta), yi) + \lambda\, R(\Theta)$$

Minimizing errors

Summary

- Linear functions: a great class of hypothesis for ML, worked for decades
- Non-linearity: seems useful, since many problems are non linear, e.g. XOR problem
- Learning is about solving an optimization problem, i.e. minimizing a function called the loss (while keeping the complexity of the model ‘reasonable’).



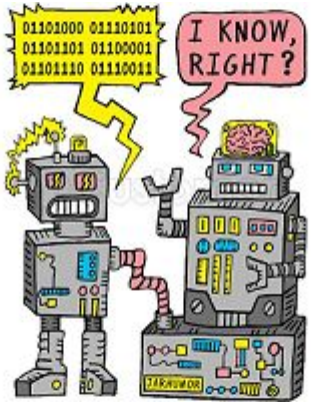
Representation function

- “Feature engineering”:
  - choose features, e.g. words, POS, NE, gaze, meta-data …
  - represent information (vectorizing, normalizing): bow, n-grams ; TF-IDF, …

ML $\longrightarrow$ DL: change here = NN seen as representation learners

ML $\longrightarrow$ DL: **sparse vs dense inputs**

Feature representation



Main issue:

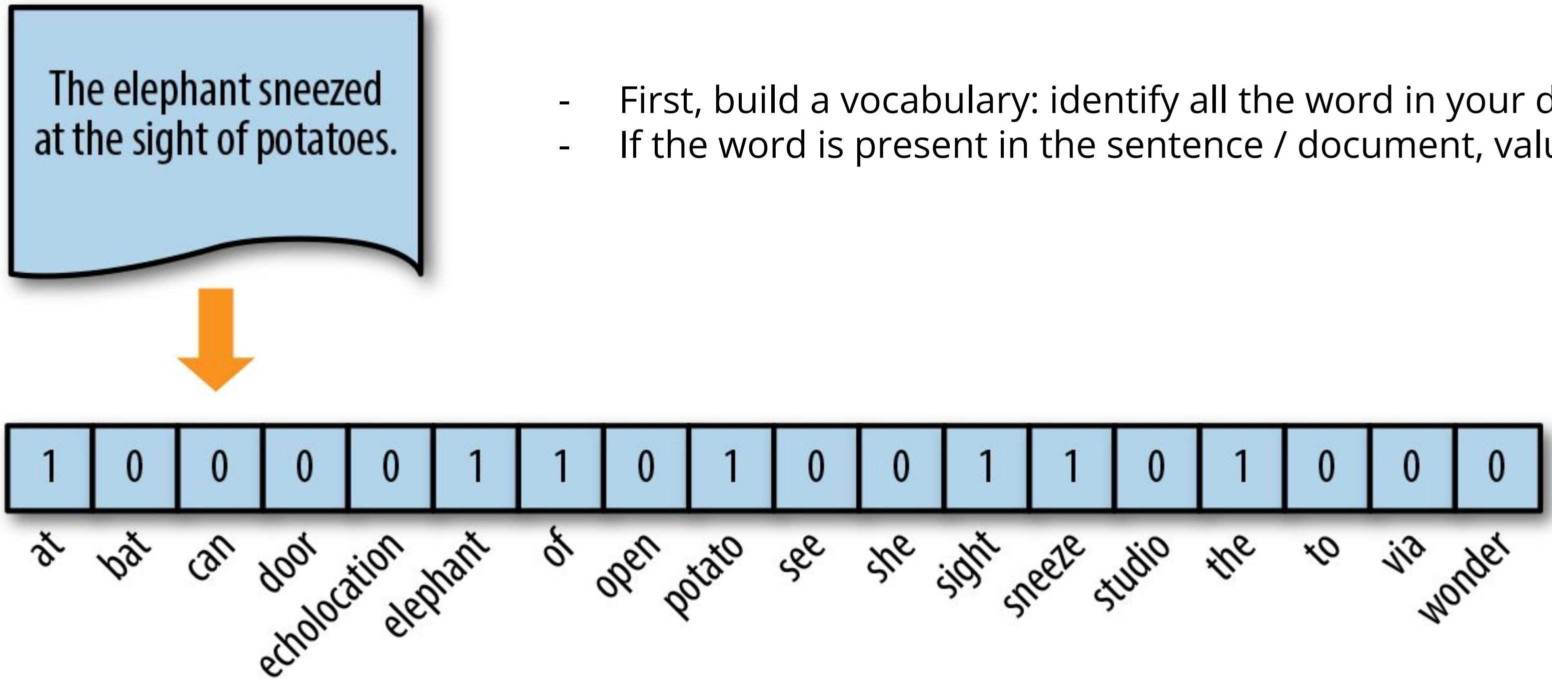
- how to represent text?

e.g. how to transform a sentence into a vector of numerical values?

Bag-of-Words (BOW):

- one vector where **each dimension is a word in our vocabulary**
- if the word / feature is present in the document, associate a specific value

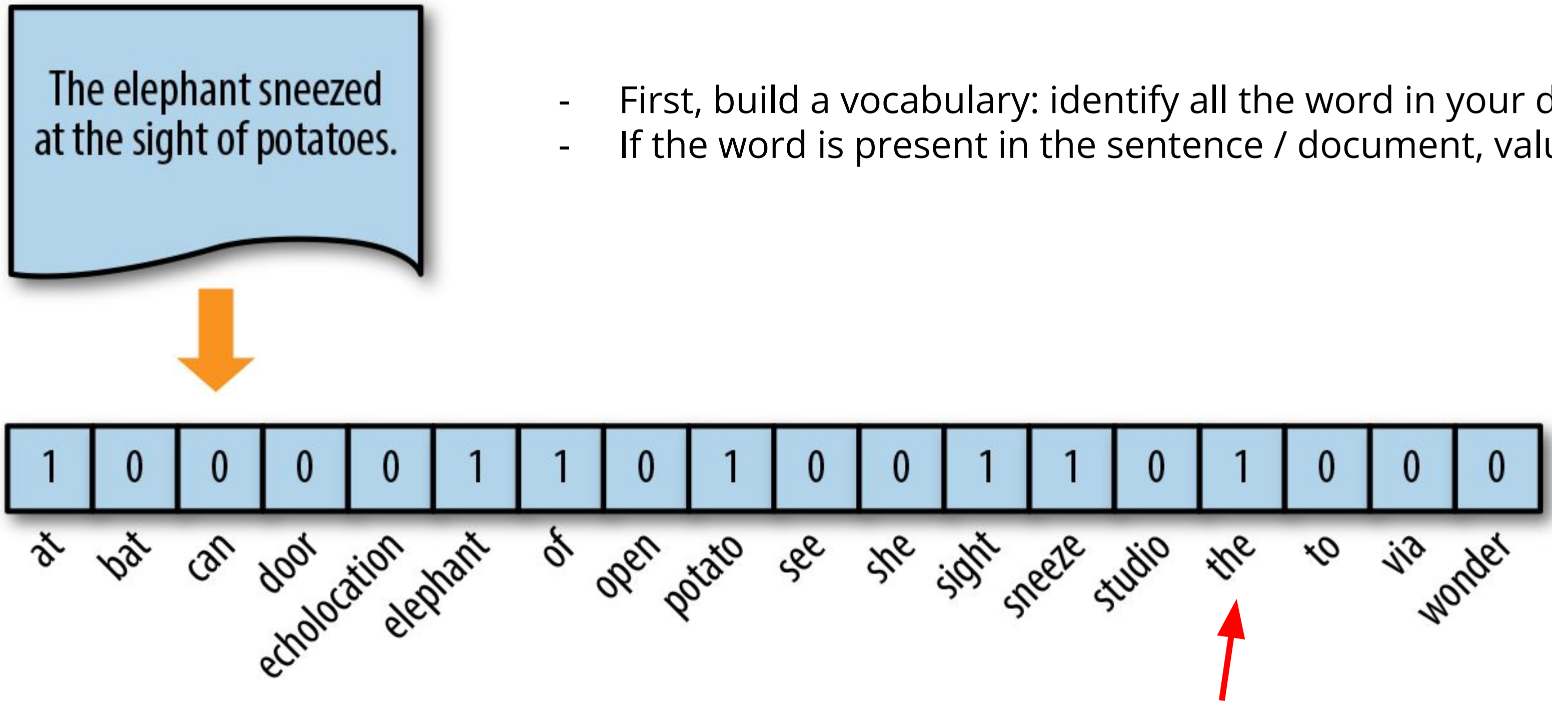
BOW: One-hot encoding



- First, build a vocabulary: identify all the word in your data
- If the word is present in the sentence / document, value = 1

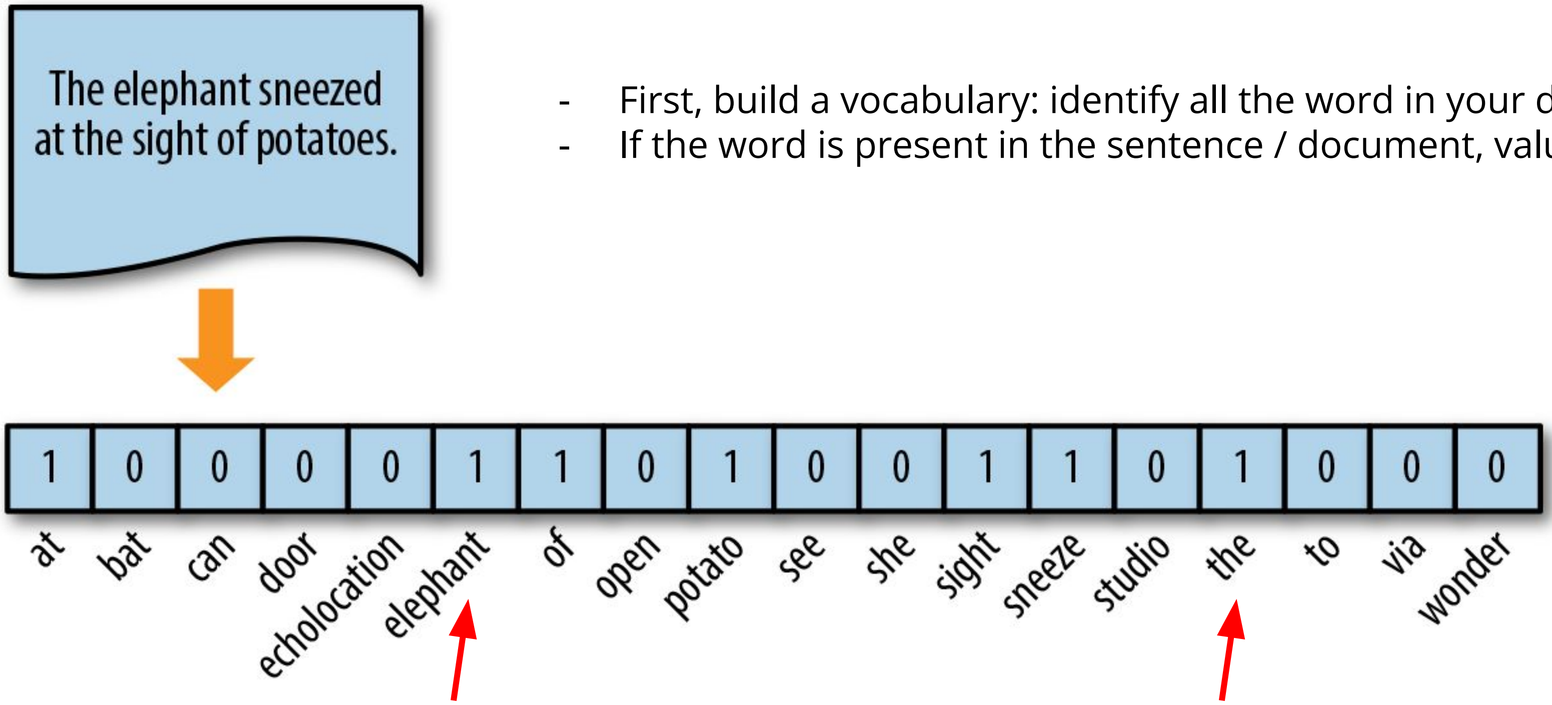
18 words / dimensions

BOW: One-hot encoding



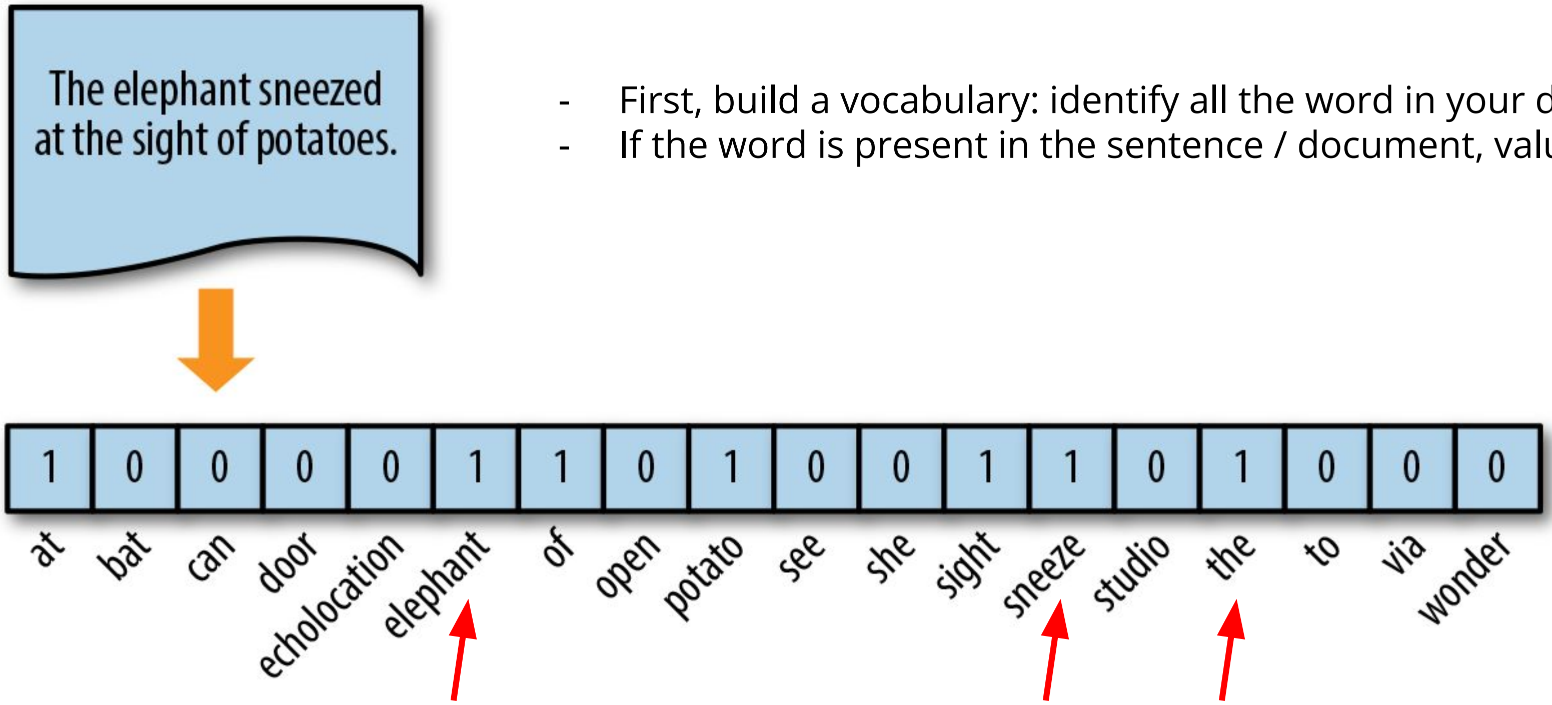
- First, build a vocabulary: identify all the word in your data
- If the word is present in the sentence / document, value = 1

BOW: One-hot encoding



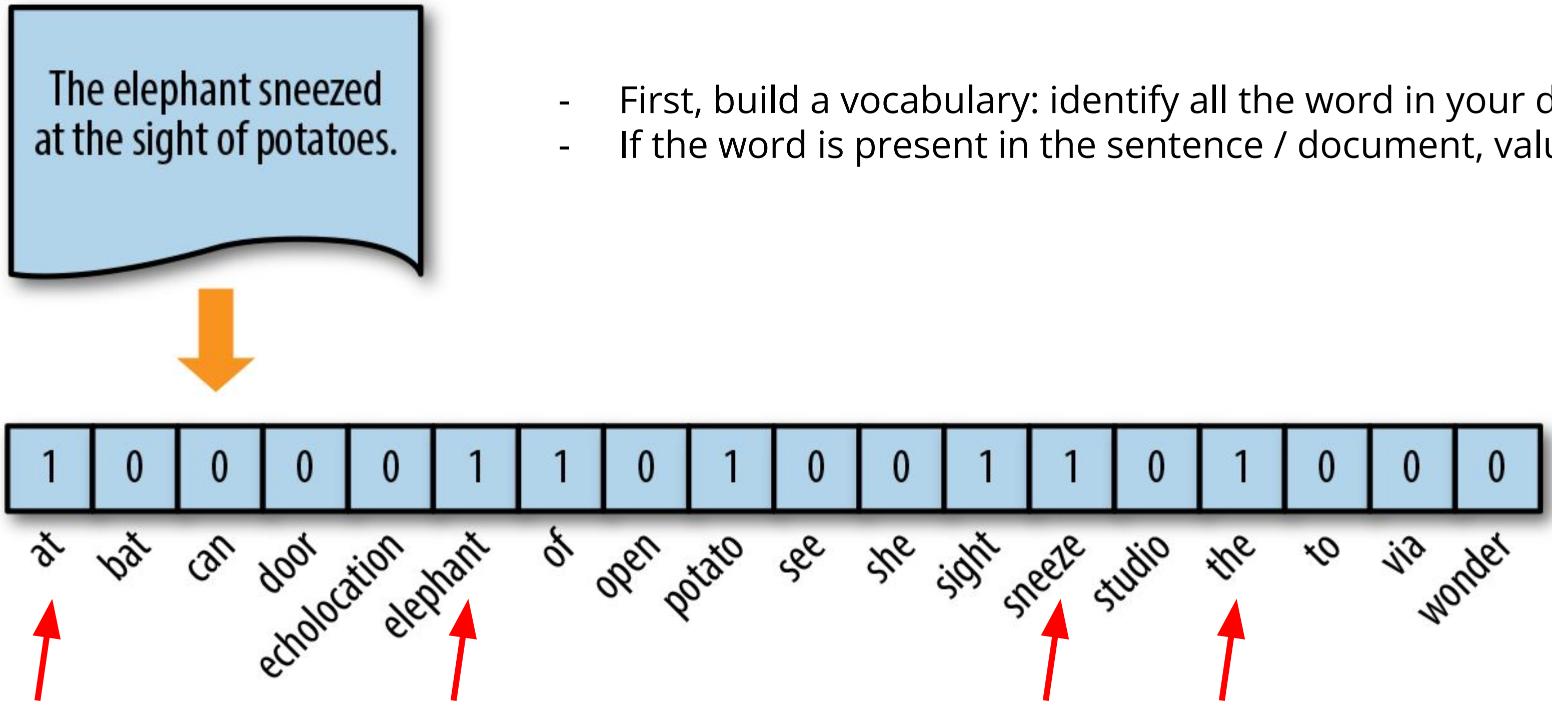
- First, build a vocabulary: identify all the word in your data
- If the word is present in the sentence / document, value = 1

BOW: One-hot encoding



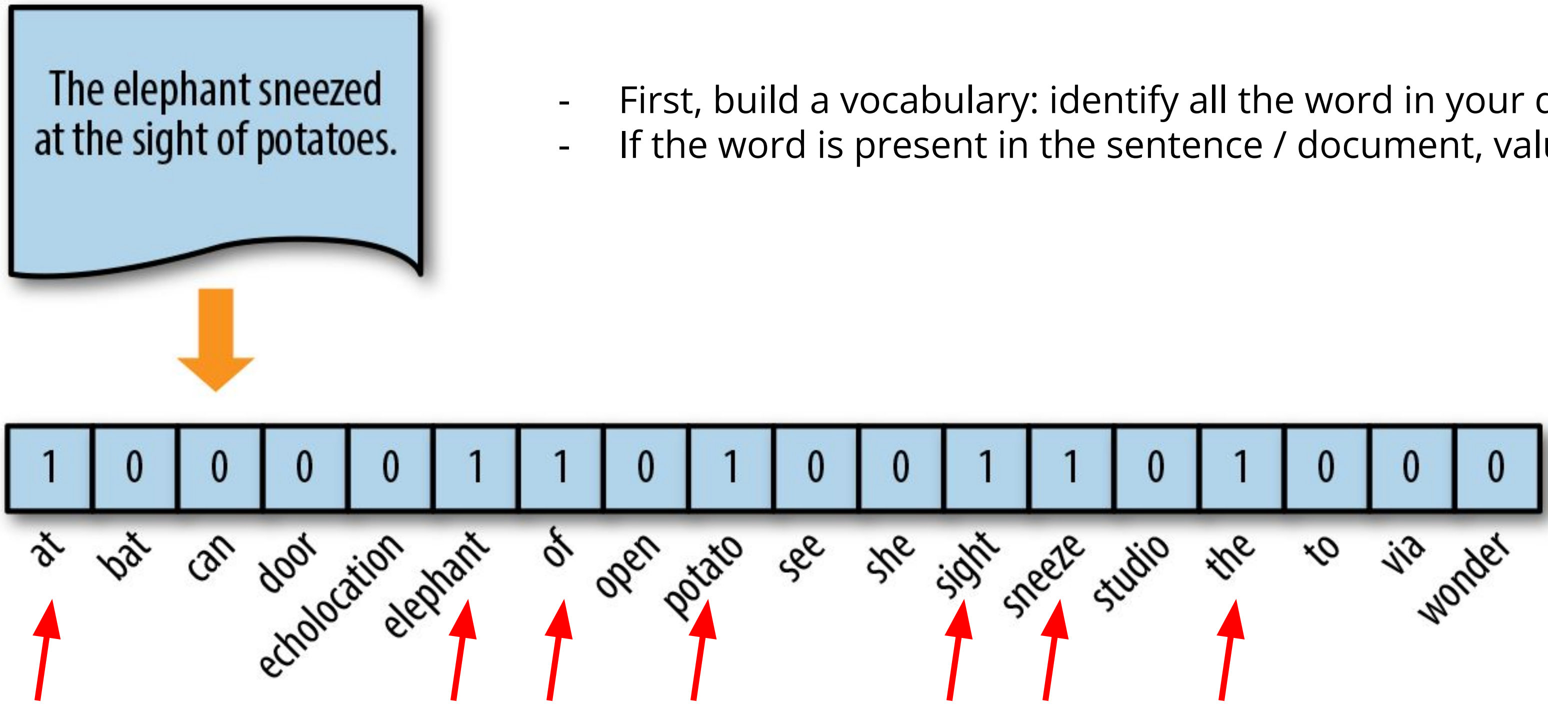
- First, build a vocabulary: identify all the word in your data
- If the word is present in the sentence / document, value = 1

BOW: One-hot encoding



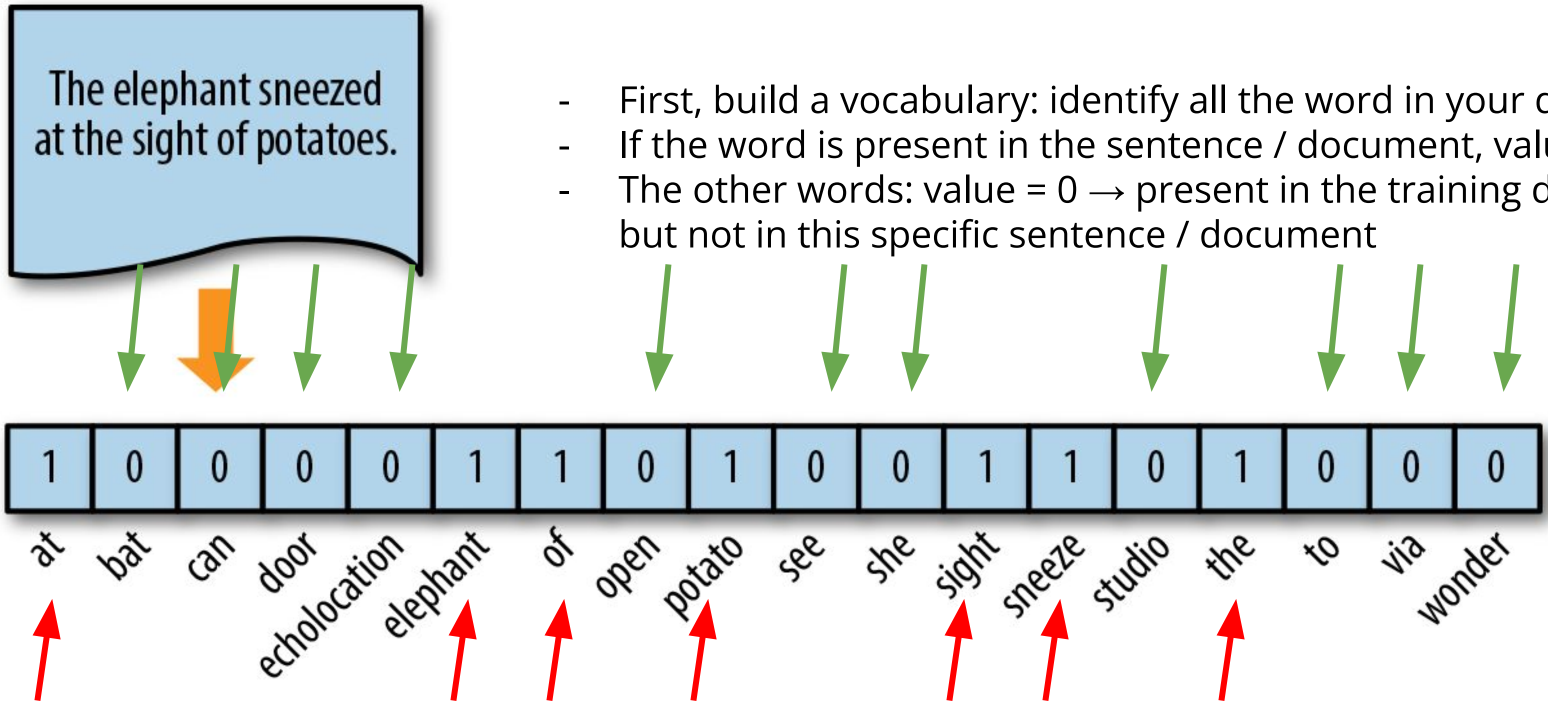
- First, build a vocabulary: identify all the word in your data
- If the word is present in the sentence / document, value = 1

BOW: One-hot encoding



- First, build a vocabulary: identify all the word in your data
- If the word is present in the sentence / document, value = 1

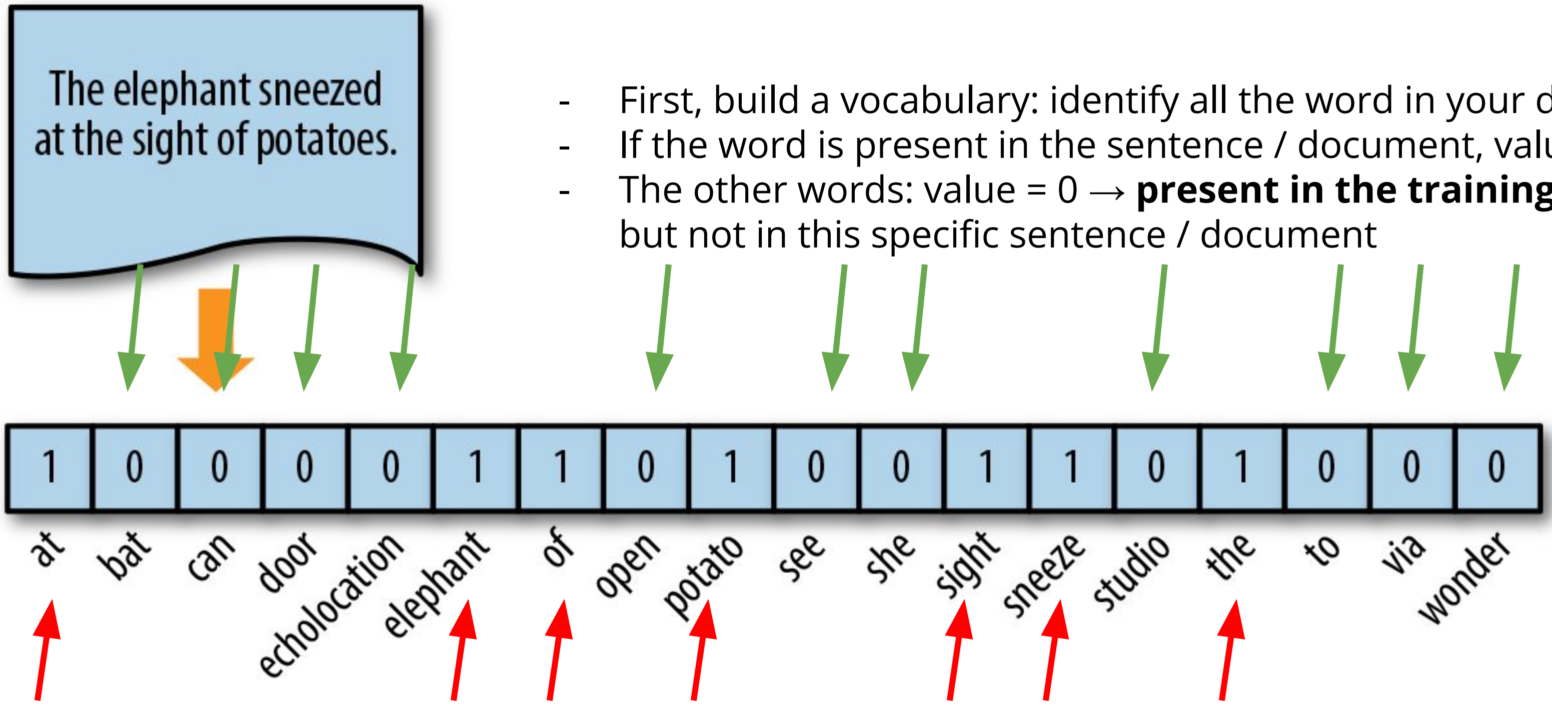
BOW: One-hot encoding



- First, build a vocabulary: identify all the word in your data
- If the word is present in the sentence / document, value = 1
- The other words: value = 0 → present in the training data, but not in this specific sentence / document

BOW: One-hot encoding

- First, build a vocabulary: identify all the word in your data
- If the word is present in the sentence / document, value = 1
- The other words: value = 0 → **present in the training data**, but not in this specific sentence / document



Bag-of-Words

Easy to use: now the computer can “read” your sentence

The elephant sneezed at the sight of potatoes.

<1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0>

Varied flavors:

- **Binary**
- Raw frequencies: some words are repeated = more important
- Normalizing with TF-IDF: take into account the distribution of the words in the entire corpus
    - “the”: very frequent but not very crucial
    - “magnificent”: rare, but crucial

Bag-of-Words

Easy to use: now the computer can “read” your sentence

The elephant sneezed at the sight of potatoes.

<1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 2, 0, 0, 0>

‘the’

Varied flavors:

- Binary
- **Raw frequencies**: some words are repeated = more important (?)
- Normalizing with TF-IDF: take into account the distribution of the words in the entire corpus
  - “the”: very frequent but not very crucial
  - “magnificent”: rare, but crucial

Bag-of-Words

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

Easy to use: now the computer can “read” your sentence

The elephant sneezed at the sight of potatoes.

<1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 2, 0, 0, 0>

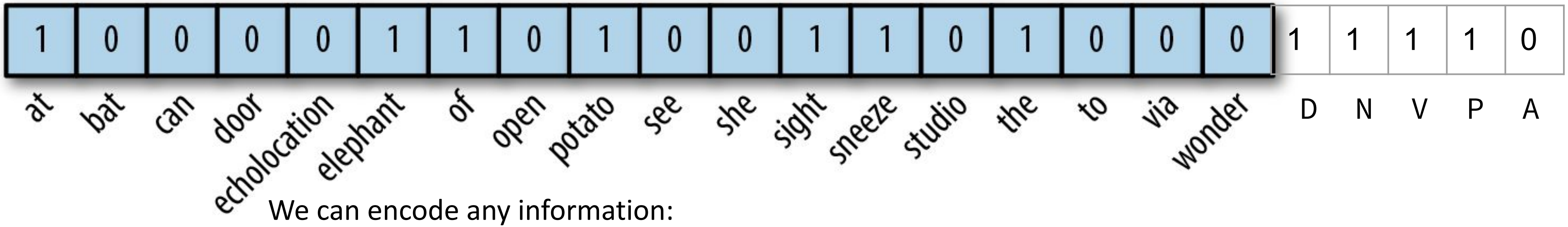
Varied flavors:

- Binary
- Raw frequencies: some words are repeated = more important
- **Normalizing with TF-IDF**: take into account the distribution of the words in the entire corpus
  - “the”: very frequent but not very crucial
  - “magnificent”: rare, but crucial

Bag of any features: one-hot encoding

Can be used to take into account any information, e.g. POS tags:

The/D elephant/N sneezed/V at/P the/D sight/N of/P potatoes/N



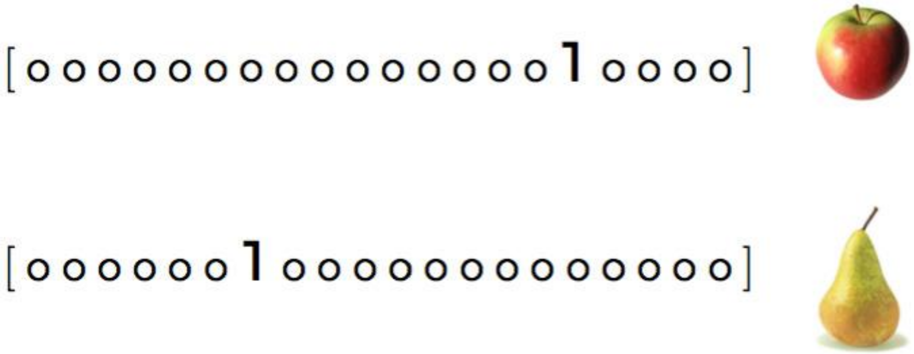
We can encode any information:

- presence of a syntactic relation
- presence of a Named Entity / numbers /dates / amounts
- word associated to a sense if disambiguated
- words in the next sentence
- semantic classes…

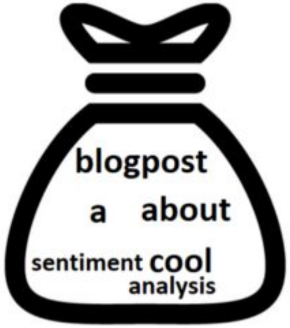
Also extra-linguistic features : gender of the writer, number of likes ...

One-hot representation

- Defining features has to be done **manually**: require expertise and tests
- A word is represented with a **one-hot vector**: easy to implement

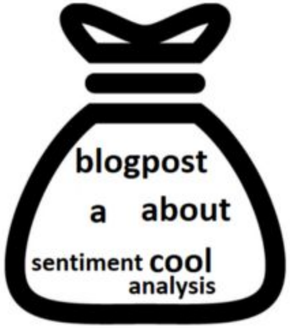


Problems and extensions



1. **Very high dimensional**:
   - 18 dimensions for the previous sentence but could be 100k dimensions!
     - Curse of dimensionality: makes learning hard, prone to overfitting
   - Solutions:
     - ignoring specific words, e.g. stop words
     - keeping only the most frequent / highest TF-IDF
     - grouping words: semantic categories, clusters (Brown)

Problems and extensions



1. Very high dimensional:
   - 18 dimensions for the previous sentence but could be 100k dimensions!
     - Curse of dimensionality: makes learning hard, prone to overfitting
   - Solutions:
     - ignoring specific words, e.g. stop words
     - keeping only the most frequent / highest TF-IDF
     - grouping words: semantic categories, clusters (Brown)
2. Bag-of-Words representation **ignores word ordering and context**
   - crucial:
     - *"I don't know why **I like this movie**."* vs *"**I don't like this movie** and I know why."*
   - solutions: n-grams, i.e. use combination of multiple words
     - e.g. trigrams such as "do not like", "like this movie"
     - but even more dimensions!

TP1: Sentiment analysis with Scikit

In the practical session, we will implement a system for sentiment classification of movie reviews.

- pre-process data (BoW, n-grams)
- train and evaluate a model
- compare different algorithms
- investigate model decisions

Sources

- *Foundations of Machine Learning*, Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar, MIT Press
- Comparing SVM and NN:
  - **Short answer:** On small data sets, SVM might be preferred. https://stats.stackexchange.com/questions/510052/are-neural-networks-better-than-svms
  - https://www.baeldung.com/cs/svm-vs-neural-network
- https://dair.ai/notebooks/nlp/2020/03/19/nlp_basics_tokenization_segmentation.html
- https://www.infoq.com/presentations/nlp-practitioners/
- https://github.com/sebastianruder/NLP-progress