ML Week 0x03 Feature Extraction

Categorical variables

- *One of K* or *one-hot* encoding — one binary feature per possible value
- The importance of not encoding order where none exists
- Text as explanatory variable $\implies$ encode as feature vectors
- Bag of words
  - Corpus (collection of documents)
  - Vocabulary (set of unique words in document)
  - Words = dimensions
  - Order of words doesn't matter
  - Order in vectors encodes words
  - Binary: present or not
  - `CountVectorizer`
    * by default converts to lowercase
    * tokens
    * stop words (mot vide) — words in most documents don't convey much information
    * stemming — rule-based, drop suffixes
      (racinisation ou désuffixation : transformer des flexions en leur radical or racine)
    * lemmatization – find root form of word
      (lemmatisation : transformer en lemme (forme canonique))
- TF - IDF
  - `norm`: L1, L2, none (in equation: max)
  - `use_idf`: enable IDF, default=True
  - `smooth_idf`: use $n_t + 1$ in IDF, default=True
  - `sublinear_tf`: Apply sublinear scaling, replacing TF_{td} with $1 + \log(TF_{td})$.
- Exercise: `HashingVectorizer`
  - Problems:
    1. Two passes to create structure: learn vocabulary (tokens), then create feature vectors

2. Vocabulary (dict) stored in memory

- Instead, `HashingVectorizer`
  * Bounded memory (no dict), even low memory (sparse scipy matrix)
  * Stateless, so can be used online (streaming) and parallel
  * Fast to serialize/unserialize
  * n_features defaults to 2^{20}
  * Note negative values. Increment takes sign of hash value, so possibility of cancellation.
  * **But** can't compute inverse transform, so hard to know which features are most important
  * Collisions can happen, but rare for 2^{20}
  * No IDF weighting, since IDF is stateful

OCR

- sklearn.digits
  * over 1700 hand-written digits (0–9)
  * 8×8 four bit pixels
  * white is most intense and represented by 0
  * black is least intense and represented by 16
- Feature vectors
  * In general, matrices not sparse
  * $100 \times 100 \implies 1e4$
  * $1920 \times 1080 \implies 2e6$
  * Problems: space, time
  * More problems: sensitive to position, rotation, scale
  * Even more problems: sensitive to illumination
  * We'll come back to this with SVM (machine à vecteurs de support)...
- Corners and edges
  * Basic computer vision techniques
  * Define: feature extraction
  * Define: feature engineering
  * Compression
  * Point matching
  * Edge detectors are mostly rotation invariant
  * So therefore corner detectors are, too
  * But scaling can hide corners

- SIFT
  - Uses scale-space
  - Approximates Laplacian of Guassian with Difference of Gaussian for finding scale space
  - Maybe a bit slower than we'd like
- SURF
  - Speeds up SIFT
  - Approximates Laplacian of Gaussian wth Box Filter for finding scale space
  - **Example**
  - Standardized dataset: zero mean, unit variance (why?)
  - $(x - \mu)/\sigma$