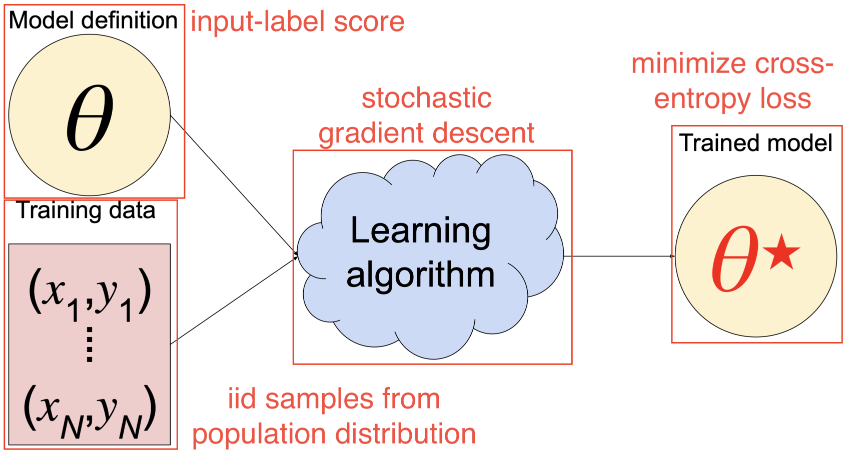
Lectures on Natural Language Processing

3. Linear Classifiers

Karl Stratos

Review: Principles of Machine Learning



No matter what the task is, the principles will (largely) stay the same.

Supervised ML Pipeline

- **Problem definition:** Define a supervised learning problem.
- **Data collection:** Start with training data for which we know the correct outcome.
- **Representation:** Choose how to represent the data.
- **Modeling:** Choose a **hypothesis class** – a set of possible explanations for the connection between input (e.g., image) and output (e.g., class label).
- **Estimation:** Find best hypothesis you can in the chosen class. This is what people usually think of as "learning".
- **Model selection:** If we have different hypothesis classes, we can select one based on some criterion.

Topic Classification with Linear Classifiers

- Problem definition
- Data collection
- Representation
- Modeling
- Estimation

Problem and Data

- **Problem.** Classify a document $x \in \mathcal{X}$ to a topic $y \in \mathcal{Y}$
- **Data.** Many annotated datasets
  - **AG News**: News articles, one of 4 types (*World*, *Sports*, *Business*, or *Sci/Tech*)
  - **DBpedia**: Wikipedia articles, one of 14 types (*Company*, *Artist*, *Film*, *Animal*, etc.)
  - **Yahoo! Answers**: Online questions/answers, one of 10 types (*Health*, *Sports*, *Science & Mathematics*, etc.)
- Related: sentiment classification
  - **IMDB**: Movie reviews, one of $K = 2$ labels (positive or negative)
  - **Yelp**: Restarant reviews, one of $K = 5$ labels (number of stars)
  - **Amazon**: Product reviews, one of $K = 5$ labels (number of stars)

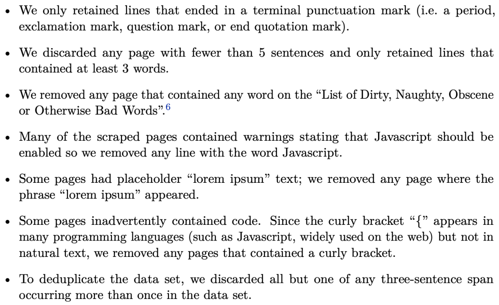
Topic Classification with Linear Classifiers

- Problem definition
- Data collection
- Representation
- Modeling
- Estimation

Text Preprocessing

- Text data can be extremely noisy.
  - Non-language strings: HTML code, URLs
  - Noise in language: typos, ungrammatical sentences, numerical values ("87.175")
- A lot of data cleaning/preprocessing efforts may be necessary.

(Raffel et al. (2020) on cleaning Common Crawl)



- We only retained lines that ended in a terminal punctuation mark (i.e. a period, exclamation mark, question mark, or end quotation mark).
- We discarded any page with fewer than 5 sentences and only retained lines that contained at least 3 words.
- We removed any page that contained any word on the "List of Dirty, Naughty, Obscene or Otherwise Bad Words".[6]
- Many of the scraped pages contained warnings stating that Javascript should be enabled so we removed any line with the word Javascript.
- Some pages had placeholder "lorem ipsum" text; we removed any page where the phrase "lorem ipsum" appeared.
- Some pages inadvertently contained code. Since the curly bracket "{" appears in many programming languages (such as Javascript, widely used on the web) but not in natural text, we removed any pages that contained a curly bracket.
- To deduplicate the data set, we discarded all but one of any three-sentence span occurring more than once in the data set.

- We will assume text is already reasonable clean.

Tokenization

"*the dog didn't see the cat.*"

- **Whitespace**? $[the, dog, didn't, see, the, cat.]$ (length 6)
  - Can't handle non-whitespace splits ("cat.")
- **Rule-based**? $[the, dog, did, n't, see, the, cat, .]$ (length 8)
  - Language specific, needs experts to develop manual rules
- **Bytes**??? $[\backslash\mathrm{xec}, \backslash\mathrm{x95}, \ldots, \backslash\mathrm{xeb}, \backslash\mathrm{x85}, \backslash\mathrm{x95}]$
  - UTF-8: can encode all > 1 million valid character code points in Unicode using ≤ 4 one-byte (8-bit) code units.
  - Language agnostic, but completely unreadable. Also the space of possible token types is too big (2^{32}).

Modern NLP: automatically induce tokenization rules

- Given a budget (e.g., at most 10,000 token types), optimizing some unsupervised objective on large quantities of text
- WordPiece (Wu et al., 2016), byte-pair encoding (BPE) (Sennrich et al., 2016)

Index Mapping

Once text is cleaned and tokenized

- Every piece of text = sequence of integers

$$[\text{the}, \text{dog}, \text{saw}, \text{the}, \text{cat}] \qquad \rightarrow \qquad [7, 10, 3870, 7, 13]$$

- **Vocabulary** $\mathcal{V} = \{1 \ldots V\}$: Set of possible token types
  - Depends on what tokenization scheme we choose
- More jargons in NLP:
  - **Corpus**: Unlabeled text dataset
  - n-**gram**: Sequence of n word types. For $n = 1, 2, 3$, called **unigram**, **bigram**, **trigram**

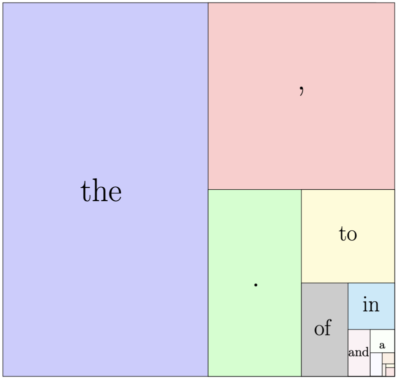
At this point we work with integers only. There is no string.

Zipf's Law

Let $w_1 \ldots w_V \in \mathcal{V}$ be unigrams sorted in decreasing probability. Empircally the following seems to hold for all $1 \leq i < V$:

$$\begin{aligned} &\text{Probability of seeing } w_i \text{ in text} \\ &\approx 2 \times \text{Probability of seeing } w_{i+1} \text{ in text} \end{aligned}$$

First four words: 93% of the unigram probability mass?



Document Representation

- Represent a document as a vector to classify.
- **Bag-of-words (BOW)** representation: $x \in \{0,1\}^V$ indicating the presence of word types

$$\text{document} = [4, 2, 2, 1, 2] \quad \rightarrow \quad x = \begin{bmatrix} 0 \\ \color{red}{1} \\ \color{red}{1} \\ 0 \\ \color{red}{1} \\ 0 \\ 0 \\ 0 \end{bmatrix} \in \{0,1\}^V$$

- What information are we losing in this representation?
- Benefits: Simple, memory-efficient (sparse), good for modeling topics (presence of "keywords")

Topic Classification with Linear Classifiers

- Problem definition
- Data collection
- Representation
- Modeling
- Estimation

Hypothesis Space of Linear Classifiers

- Possible models: all mappings from $\mathbb{R}^V$ to $\{1 \ldots K\}$
- **Linear classifiers**: all $(W, b) \in \mathbb{R}^{V \times K} \times \mathbb{R}^K$ defining

$$x \mapsto \arg\max_{y=1}^{K} [\underbrace{\underbrace{W^\top}_{K \times V} \underbrace{x}_{V \times 1} + \underbrace{b}_{K \times 1}}_{K \times 1}]_y$$

- Can be seen as defining input-label scores

$$\textbf{score}_{W,b}(x, y) := w_y^\top x + b_y =: h_y$$

where $w_y \in \mathbb{R}^V$ is the y-th column of $W = [w_1 \ldots w_K]$

- $h \in \mathbb{R}^K$ called **scores** or **logits** (for x)

Tip: Matrix Multiplication

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} ag + bh \\ cg + dh \\ eg + fh \end{bmatrix}$$

Two practical interpretations of matrix-vector product

1. Dot product between the vector and each row of the matrix
2. Linear combination of the columns of the matrix

(Matrix-matrix product: concatenating matrix-vector products)

Linear Classifier with BOW Representation

$$\textbf{score}_{W,b}(x, y) = \sum_{i=1:\ x_i=1}^{V} [w_y]_i + b_y$$

"Interpretable": For $x = \mathrm{BOW}(\text{"market up market up up"})$,

$$\begin{aligned}\textbf{score}_{W,b}(x, \mathrm{business}) &= [w_{\mathrm{business}}]_{\text{"market"}} + [w_{\mathrm{business}}]_{\text{"up"}} + b_{\mathrm{business}} \\ &\gg [w_{\mathrm{sports}}]_{\text{"market"}} + [w_{\mathrm{sports}}]_{\text{"up"}} + b_{\mathrm{sports}} \\ &= \textbf{score}_{W,b}(x, \mathrm{sports})\end{aligned}$$

Topic Classification with Linear Classifiers

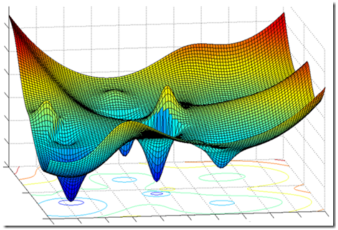
- Problem definition
- Data collection
- Representation
- Modeling
- Estimation

Review: Empirical Cross-Entropy Loss

Training data: $(x_1, y_1) \ldots (x_N, y_N) \overset{\text{iid}}{\sim} \mathbf{pop}_{XY}$

$$\widehat{J}_N(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log p_\theta(y_i | x_i)$$

- Goal: find θ with small $\widehat{J}_N(\theta)$
- Universal approach: **gradient descent**
- We will express $\widehat{J}_N$ as an explicit (differentiable) function of parameters W, b to derive the gradients.



Empirical Cross-Entropy Loss for Linear Classifiers

$$-\log p_\theta(y|x) = -\log \frac{\exp(\textbf{score}_{W,b}(x,y))}{\sum_{y'=1}^{K} \exp(\textbf{score}_{W,b}(x,y'))}$$
$$= \log \left(\sum_{y'=1}^{K} \exp(\textbf{score}_{W,b}(x,y')) \right) - \textbf{score}_{W,b}(x,y)$$

Plugging in the model definition $\textbf{score}_{W,b}(x,y) = w_y^\top x + b_y$

$$\widehat{J}_N(W,b) = \frac{1}{N} \sum_{i=1}^{N} \log \left(\sum_{y=1}^{K} \exp(w_y^\top x_i + b_y) \right) - w_{y_i}^\top x_i - b_{y_i}$$

- Differentiable, also convex: thus gradient descent will find a global optimum (with the right step size)
- When is this minimized?

What Can Happen With Finite Training Data?

$N = 2$ examples

$$\begin{aligned} \mathcal{V} &= \{1:\texttt{stock}, 2:\texttt{game}, 3:\texttt{monday}, 4:\texttt{friday}\} \\ \mathcal{L} &= \{\texttt{business}, \texttt{sports}\} \\ x_1 &= (1,0,1,0) \qquad y_1 = \texttt{business} \\ x_2 &= (0,1,0,1) \qquad y_2 = \texttt{sports} \end{aligned}$$

Verify that the following model achieves zero training loss:

$$\begin{aligned} w^*_{\texttt{business}} &= (0,0,999,0) \qquad & b^*_{\texttt{business}} &= 0 \\ w^*_{\texttt{sports}} &= (0,0,0,999) \qquad & b^*_{\texttt{sports}} &= 0 \end{aligned}$$

Is this a good model?

Generalization Issues

Test example ("game monday")

$$x_{\text{new}} = (0, 1, 1, 0) \qquad y_{\text{new}} = \texttt{sports}$$

Model output

$$p_{W^*,b^*}(\texttt{business}|x_{\text{new}}) = 1 \qquad p_{W^*,b^*}(\texttt{sports}|x_{\text{new}}) = 0$$

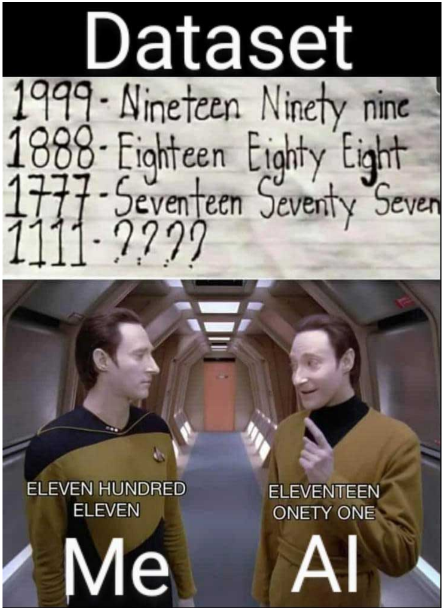
This happened because the model picked up the following non-generalizable rule to fit the training data perfectly

- If "monday", predict business.
- If "friday", predict sports.

In contrast, *we* know that "stock" is predictive of business and "game" is predictive of sports.

- Training data, *because of finiteness*, allowed an unnatural "loophole" that the model can exploit

Overfitting


Dataset
1999 - Nineteen Ninety nine
1888 - Eighteen Eighty Eight
1777 - Seventeen Seventy Seven
1111 - ????
ELEVEN HUNDRED ELEVEN
ELEVENTEEN ONETY ONE
Me
AI

Overfitting: Model succeeds in fitting (finite) training data by exploiting spurious input-label correlations that do not generalize.

Train-Validation-Test Split of Data

Always prepare a 3-way split of a labeled dataset

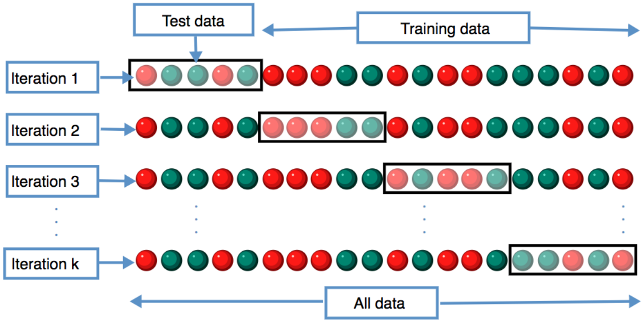
- **Training set**: Use this portion for training a supervised model. Majority of data ($> 90\%$)
- **Validation (aka. development/held-out) set**: Use this portion to check generalization performance of a trained model. About the same size as the test set
- **Test set**: After all experimentations and tuning, apply the trained model to this portion once and report final performance.

If dataset is missing a validation set, make one from random samples in the training set.

- If you don't have enough labeled data, consider k-fold **cross-validation**: a method for "reusing" held-out data

k-fold cross-validation

- Partition training data into k roughly equal parts
- Train on all but i-th part, test on i-th part



Use the average performance to assess a training configuration

- For final evaluation, train on all parts

Regularization Methods

Ways to prevent overfitting.

1. Get more labeled data: best regularization method if possible!

2. **Early stopping**: stop training when validation performance fails to improve for a certain number of times ("patience")

3. Explicit regularization term: for some $\lambda > 0$ (to be tuned on dev set)

$$\min_{\theta \in \mathbb{R}^d} \widehat{J}_N(\theta) + \lambda \underbrace{\sum_{i=1}^{d} \theta_i^2}_{||\theta||_2^2} \quad \text{or} \quad \min_{\theta \in \mathbb{R}^d} \widehat{J}_N(\theta) + \lambda \underbrace{\sum_{i=1}^{d} |\theta_i|}_{||\theta||_1}$$

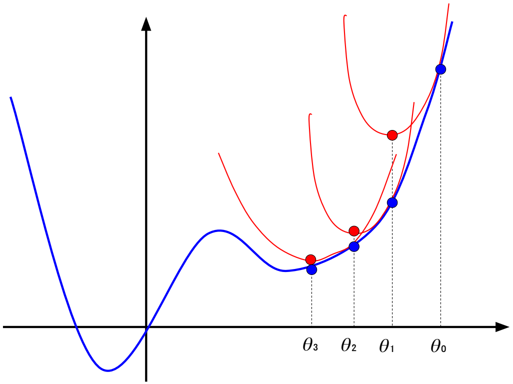
4. Other techniques (e.g., dropout, label smoothing), will come back to these

Review: Gradient Descent

Start from some $\theta_0 \in \mathbb{R}^d$, repeatedly minimize local approx. of $f(\theta)$ around θ_t by

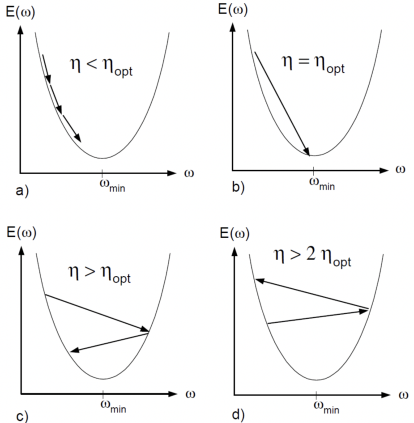
$$\theta_{t+1} = \theta_t - \underbrace{\eta_t}_{\text{“step size” or “learning rate”}} \nabla f(\theta_t)$$

until $\nabla f(\theta_t) \approx 0_d$



Gradient Descent Convergence

- Generally, for convex functions, gradient descent will converge to global optimum
  - Stop by (a combination of): max number of iterations, plateau in validation error, and other criteria
- The learning rate η may be very important to ensure rapid convergence (or convergence at all)



(LeCun et al, 1996)

Stochastic Gradient Descent (SGD)

- **Input**: N examples (x_i, y_i) defining per-example losses $\widehat{J}_i : \mathbb{R}^d \to \mathbb{R}$
- **Objective**: Average loss $\widehat{J}_N(\theta) = (1/N) \sum_{i=1}^N \widehat{J}_i(\theta)$
- **Hyperparameters**: Number of epochs E, (mini-)batch size B, learning rate schedule η_t
- **Algorithm**: Initialize $\theta_0 \in \mathbb{R}^d$. For E epochs:
    1. Shuffle $(1 \dots N)$ and partition into B-sized batches $\mathcal{I}_1 \dots \mathcal{I}_M$ $(M = N/B)$.
    2. For $j = 1 \dots M$, take a gradient step

$$\theta_{t+1} \leftarrow \theta_t - \eta_t \left(\frac{1}{B} \sum_{i \in \mathcal{I}_j} \nabla \widehat{J}_i(\theta_t) \right)$$

- **Memory footprint**: $O(d)$ to store θ and gradients

Gradient for Linear Classifiers

$$\widehat{J}_N(W,b) = \frac{1}{N}\sum_{i=1}^{N} \log\left(\sum_{y=1}^{K} \exp(w_y^\top x_i + b_y)\right) - w_{y_i}^\top x_i - b_{y_i}$$

Can show $\widehat{J}_N$ is convex, so gradient descent will converge to a global minimum. Exercise: derive the gradients

$$\nabla_{w_y}\widehat{J}_N(W,b) = \frac{1}{N}\sum_{i=1}^{N} \left(p_{W,b}(y|x_i) - \mathbb{1}(y = y_i)\right) x_i \in \mathbb{R}^V$$

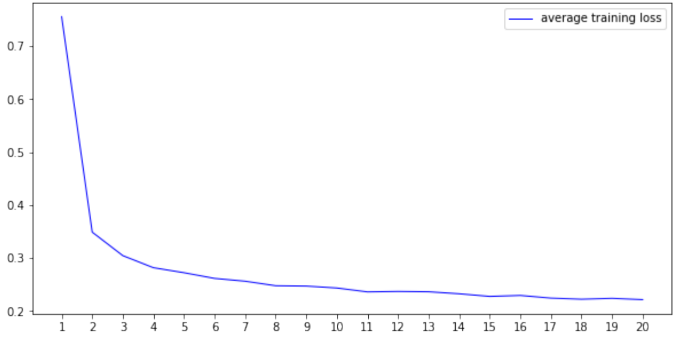
$$\nabla_{b_y}\widehat{J}_N(W,b) = \frac{1}{N}\sum_{i=1}^{N} (p_{W,b}(y|x_i) - \mathbb{1}(y = y_i)) \in \mathbb{R}$$

Intuitive: adjust the difference between model prediction and ground truth

Rule 1: Always Monitor Training Loss

$\widehat{J}_N(\theta)$ should decrease each epoch on average: this is what we are optimizing!

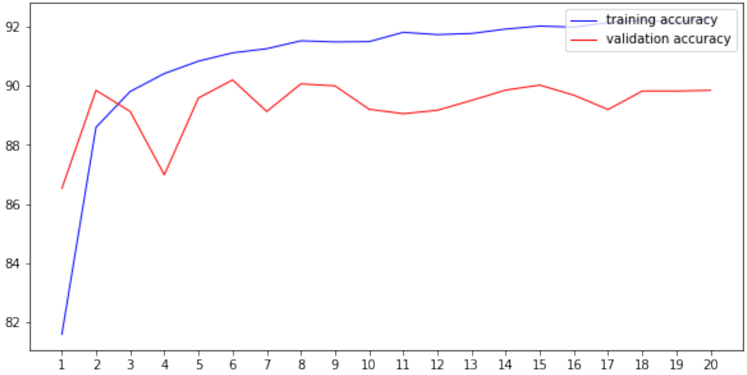
- Easy to track: accumulate losses over batches and divide by N



Rule 2: Always Monitor Validation Performance

Check validation performance (at least) every epoch and do early stopping to prevent overfitting.

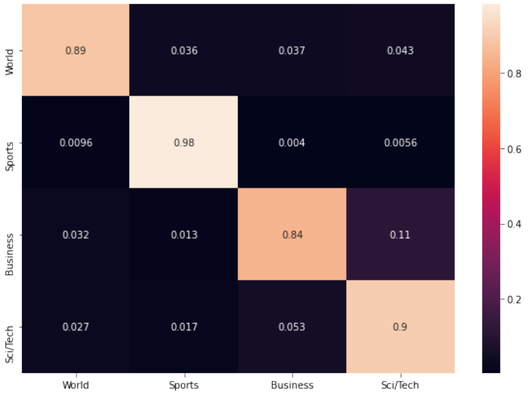
- Performance for topic classification is simple accuracy (# correct / # test examples), but it can be more complicated for other tasks (structured prediction?)



Rule 3: Look at Errors

Once performance seems okay by quantitative metrics, do some *qualitative* analysis of errors to get an actual understanding of challenges in the task and how to improve

- Confusion matrix works for simple classification, but may need to be creative to analyze complex problems (translation?)

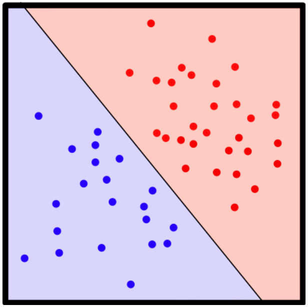


Summary: Topic Classification with Linear Classifiers

- **Problem definition:** topic classification
- **Data collection:** topic-labeled documents (e.g., AG News)
- **Representation:** bag-of-words (BOW)
- **Modeling:** hypothesis class of linear classifiers
- **Estimation:** minimize empirical cross-entropy loss by SGD, with regularization
- **Model selection:** pick the model with best validation performance

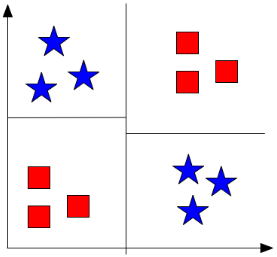
Limitations of a Linear Classifier

Linearly separable



Accuracy 100% ✓

Not linearly separable (e.g., XOR)



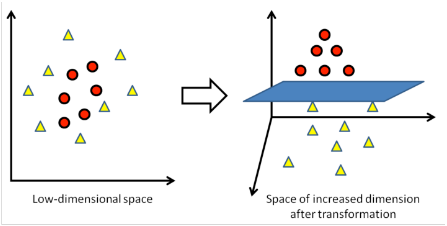
Accuracy $\leq 50\%$ ✗

Solutions

1. **Feature engineering:** Specify better input representation
2. **Feature learning:** Representation $=$ part of model to learn

Feature Engineering

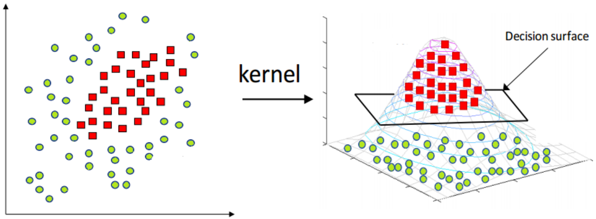
- You can always add new dimensions until data is separable



- Much of past NLP research spent in feature engineering. For instance, for topic classification consider
  - Higher-order features: bag-of-n-grams $x \in \{0,1\}^{V^n}$?
  - Side information (e.g., author identity? date? length?)
- Pros: Requires a deep understanding of the specific problem, interpretable weights, can work well even with small data by specifying right features
- Con: Have to do this for every new problem, no way to know how much engineering is enough

Aside: Kernel Trick

Technique to *implicitly* enrich input representation, applicable whenever model/learning involves only dot product between inputs (e.g., SVMs)



No need to manually engineer good features, but has other cons (not easily scalable to large data, still have to choose the kernel)

- Largely out of the scope of this course
- Active research on connections between kernel machines and deep learning

Feature Learning as Part of Model Learning

- This is "deep learning".
- Parameterizes an **encoder** $\mathbf{enc}_\theta : \mathcal{X} \to \mathbb{R}^H$, which computes the input representation
- Only score changes: score of $(x, y) \in \mathcal{X} \times \{1 \ldots K\}$ now

  $$\mathbf{score}_\theta(x, y) := w_y^\top \mathbf{enc}_\theta(x) + b_y$$

  θ now includes all parameters associated with the encoder, as well as the linear classifier parameters $(W = [w_1 \ldots w_K], b)$
- Conditional label distribution defined the same way

  $$p_\theta(y|x) = \frac{\exp(\mathbf{score}_\theta(x, y))}{\sum_{y'=1}^K \exp(\mathbf{score}_\theta(x, y'))} \qquad \forall y = 1 \ldots K$$
- Same training scheme: gradient descent on the cross-entropy loss

Example: One Hidden Layer Feedforward Network

- Parameters: $\theta \in \mathbb{R}^{100V+20200+201K}$ referring to
  - **Embedding matrix**: $E = [e_1 \dots e_V] \in \mathbb{R}^{100 \times V}$, $e_i \in \mathbb{R}^{100}$ is a dense, 100-dimensional vector representation of word $i \in \mathcal{V}$
  - **Hidden layer**: $U \in \mathbb{R}^{100 \times 200}$ and $a \in \mathbb{R}^{200}$
  - **Linear layer**: $W = [w_1 \dots w_K] \in \mathbb{R}^{200 \times K}$ and $b \in \mathbb{R}^K$
- Encoder: given an initial BOW representation $x \in \{0,1\}^V$ of a document, let $\textbf{avg}_E(x) := (1/|x|) \sum_{i:x_i=1} e_i$ and compute

$$\textbf{enc}_\theta(x) = \max \left\{ 0, \underbrace{U^\top}_{200 \times 100} \underbrace{\textbf{avg}_E(x)}_{100 \times 1} + \underbrace{a}_{200 \times 1} \right\} \in \mathbb{R}^{200}$$

  where $\max\{0, v\}$ is elementwise.
- $\textbf{score}_\theta(x,y) = w_y^\top \textbf{enc}_\theta(x) + b_y$
- $p_\theta(y|x) \propto \exp(\textbf{score}_\theta(x,y))$

Training

Use SGD to optimize

$$\min_{\theta \in \mathbb{R}^d} \; -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\exp(\textbf{score}_\theta(x_i, y_i))}{\sum_{y=1}^{K} \exp(\textbf{score}_\theta(x_i, y))}$$

Recall: θ denotes parameters of the encoder as well as (W, b)

Important questions

- How should we define the encoder? Does it matter?
- How can we calculate gradients?
- How can we make training efficient with so many parameters?