Features

Sanjay Lall and Stephen Boyd

EE104
Stanford University
Records and embedding
Raw data

- raw data pairs are \((u, v)\), with \(u \in \mathcal{U}, v \in \mathcal{V}\)
- \(\mathcal{U}\) is set of all possible input values
- \(\mathcal{V}\) is set of all possible output values
- each \(u\) is called a record
- typically a record is a tuple, or list, \(u = (u_1, u_2, \ldots, u_r)\)
- each \(u_i\) is a field or component, which has a type, e.g., real number, Boolean, categorical, ordinal, word, text, audio, image, parse tree (more on this later)
- e.g., a record for a house for sale might consist of
  
  (address, photo, description, house/apartment?, lot size, \ldots, \# bedrooms)
Feature map

- learning algorithms are applied to \((x, y)\) pairs,

\[
x = \phi(u), \quad y = \psi(v)
\]

- \(\phi: \mathcal{U} \rightarrow \mathbb{R}^d\) is the \textit{feature map} for \(u\)
- \(\psi: \mathcal{V} \rightarrow \mathbb{R}^m\) is the \textit{feature map} for \(v\)
- feature maps transform \textit{records} into \textit{vectors}
- feature maps usually work on each field separately,

\[
\phi(u_1, \ldots, u_r) = (\phi_1(u_1), \ldots, \phi_r(u_r))
\]
- \(\phi_i\) is an \textit{embedding} of the type of field \(i\) into a vector
Embeddings

- embedding puts the different field types on an equal footing, *i.e.*, vectors

- some embeddings are simple, *e.g.*, 
  - for a number field ($\mathcal{U} = \mathbb{R}$), $\phi_i(u_i) = u_i$
  - for a Boolean field, $\phi_i(u_i) = \begin{cases} 
  1 & u_i = \text{TRUE} \\
  -1 & u_i = \text{FALSE} 
\end{cases}$
  - color to $(R, G, B)$

- others are more sophisticated
  - text to TFIDF histogram
  - word2vec (maps words into vectors)
  - pre-trained neural network for images (maps images into vectors)

(more on these later)
Faithful embeddings

A faithful embedding satisfies

- $\phi(u)$ is near $\phi(\tilde{u})$ when $u$ and $\tilde{u}$ are ‘similar’
- $\phi(u)$ is not near $\phi(\tilde{u})$ when $u$ and $\tilde{u}$ are ‘dissimilar’

- lefthand concept is vector distance
- righthand concept depends on field type, application

- interesting examples: names, professions, companies, countries, languages, ZIP codes, cities, songs, movies
- we will see later how such embeddings can be constructed
Examples

- geolocation data: $\phi(u) = (\text{Lat}, \text{Long})$ in $\mathbb{R}^2$ or embed in $\mathbb{R}^3$ (if data points are spread over planet)

- day of week (each day is ‘similar’ to the day before and day after)
Example: word2vec

- word2vec maps a dictionary of 3 million words (and short phrases) into $\mathbb{R}^{300}$
- developed from a data set from Google News containing 100 billion words
- assigns words that frequently appear near each other in Google News to nearby vectors in $\mathbb{R}^{300}$
Example: word2vec

(showing only $x_1$ and $x_2$, for a selection of words associated with emotion)
Imagenet embedding

- Imagenet is an open image database with 14m labeled images in 1000 classes
- vgg16 maps images $u$ (224 × 224 pixels with R,G,B components) to $x \in \mathbb{R}^{4096}$
- vgg16 was originally developed to classify the image labels
- repurposed as general image feature mapping
- vgg16 has neural network form with 16 layers, with input $u$, output $x$
Images $u^i$ for $i = 1, 2, \ldots, 6$ are embedded to $x^i = \phi(u^i) \in \mathbb{R}^{4096}$

Matrix of pairwise distances $d_{ij} = \|x^i - x^j\|_2$

$$d = \begin{bmatrix}
0 & 109.7 & 97.9 & 96.2 & 107.4 & 103.0 \\
0 & 63.9 & 71.6 & 109.4 & 109.2 \\
0 & 69.4 & 101.5 & 101.4 \\
0 & 96.5 & 96.8 \\
0 & 86.6 \\
0
\end{bmatrix}$$
**Standardized embeddings**

we usually assume that an embedding is *standardized*

- entries of $\phi(u)$ are centered around 0
- entries of $\phi(u)$ have RMS value around 1
- roughly speaking, entries of $\phi(u)$ range over $\pm 1$

- with standardized embeddings, entries of feature map

  $$\phi(u_1, \ldots, u_r) = (\phi_1(u_1), \ldots, \phi_r(u_r))$$

  are all comparable, *i.e.*, centered around zero, standard deviation around one

- $\text{rms}(\phi(u) - \phi(\tilde{u}))$ is reasonable measure of how close records $u$ and $\tilde{u}$ are
**Standardization or \( z \)-scoring**

- Suppose \( \mathcal{U} = \mathbb{R} \) (field type is real numbers)

- For data set \( u^1, \ldots, u^n \in \mathbb{R} \)

\[
\bar{u} = \frac{1}{n} \sum_{i=1}^{n} u^i \quad \text{std}(u) = \left( \frac{1}{n} \sum_{i=1}^{n} (u^i - \bar{u})^2 \right)^{\frac{1}{2}}
\]

- The \( z \)-score or standardization of \( u \) is the embedding

\[
x = \text{zscore}(u) = \frac{1}{\text{std}(u)} (u - \bar{u})
\]

- Ensures that embedding values are centered at zero, with standard deviation one

- \( z \)-scored features are very easy to interpret: \( x = \phi(u) = +1.3 \) means that \( u \) is 1.3 standard deviations above the mean value
Log transform

- old school rule-of-thumb: if field $u$ is positive and ranges over wide scale, embed as $\phi(u) = \log u$ (or $\log(1 + u)$ if $u$ is sometimes zero), then standardize

- examples: web site visits, ad views, company capitalization

- interpretation as faithful embedding:
  - $20$ and $22$ are similar, as are $1000$ and $1100$
  - but $20$ and $120$ are not similar
  - *i.e.*, you care about fractional or relative differences between raw values
  
  (here, log embedding is faithful, affine embedding is not)

- can also apply to output or label field, *i.e.*, $y = \psi(v) = \log v$ if you care about percentage or fractional errors; recover $\hat{v} = \exp(\hat{y})$
Example: House price prediction

- we want to predict house selling price $v$ from record $u = (u_1, u_2)$
  - $u_1 =$ area (sq. ft.)
  - $u_2 =$ # bedrooms
- we care about relative error in price, so we embed $v$ as $\psi(v) = \log v$ (and then standardize)
- we standardize fields $u_1$ and $u_2$

$$x_1 = \frac{u_1 - \mu_1}{\sigma_1}, \quad x_2 = \frac{u_2 - \mu_2}{\sigma_2}$$

- $\mu_1 = \bar{u}_1$ is mean area
- $\mu_2 = \bar{u}_2$ is mean number of bedrooms
- $\sigma_1 = \text{std}(u_1)$ is std. dev. of area
- $\sigma_2 = \text{std}(u_2)$ is std. dev. of # bedrooms

(means and std. dev. are over our data set)
Example: House price linear regression predictor

- predict $y = \log v$ (log of price) from standardized area and # bedrooms
- linear predictor: $\hat{y} = \theta_1 + \theta_2 x_1 + \theta_3 x_2$
- in terms of original raw data:

$$\hat{v} = \exp \left( \theta_1 + \theta_2 \frac{u_1 - \mu_1}{\sigma_1} + \theta_3 \frac{u_2 - \mu_2}{\sigma_2} \right)$$

- exp undoes log embedding of house price
- readily interpretable, e.g., what does $\theta_2 = 0.7$ mean?
Vector embeddings
Vector embeddings for real field

- we can embed a field $u$ into a vector $x = \phi(u) \in \mathbb{R}^k$
- useful even when $\mathcal{U} = \mathbb{R}$ (real field)
- polynomial embedding:
  \[ \phi(u) = (1, u, u^2, \ldots, u^d) \]
- piecewise linear embedding:
  \[ \phi(u) = (1, (u)_-, (u)_+) \]
  where $(u)_- = \min(u, 0)$, $(u)_+ = \max(u, 0)$
- linear predictor with these features yield polynomial and piecewise linear predictors of raw features
Categorical data

- data field is *categorical* if it only takes a finite number of values
- *i.e.,* \( \mathcal{U} \) is a finite set \( \{\alpha_1, \ldots, \alpha_k\} \); \( \alpha_i \) are *category labels*
- we often use category labels \( 1, \ldots, k \), and refer to ‘category \( i \)’
- examples:
  - *true/*false (two values, also called Boolean)
  - *apple, orange, banana* (three values)
  - *monday, …, sunday* (seven values)
  - ZIP code (around 40000 values)
  - countries (around 185 values)
  - languages (several thousand spoken by large numbers of people)
One-hot embedding for categoricals

- $\mathcal{U} = \{1, \ldots, k\}$
- **one-hot embedding**: $\phi(i) = e_i \in \mathbb{R}^k$
- examples:
  - $\phi(\text{apple}) = (1, 0, 0)$, $\phi(\text{orange}) = (0, 1, 0)$, $\phi(\text{banana}) = (0, 0, 1)$
  - $\phi(\text{true}) = (1, 0)$, $\phi(\text{false}) = (0, 1)$  (another embedding of Boolean, into $\mathbb{R}^2$)
  - $\phi(\text{Mandarin}) = e_1$, $\phi(\text{English}) = e_2$, $\phi(\text{Hindi}) = e_3$, $\ldots$, $\phi(\text{Azeri}) = e_{55}$, $\ldots$
- standardizing these features handles *unbalanced* data
Reduced one-hot embedding for categoricals

- $\mathcal{U} = \{1, \ldots, k\}$
- one-hot embedding maps $\mathcal{U}$ to $\mathbb{R}^k$; reduced one-hot embedding maps $\mathcal{U}$ to $\mathbb{R}^{k-1}$
- choose one value, say $i = k$, as the default or nominal value
- $\phi(k) = 0 \in \mathbb{R}^{k-1}$, i.e., map the default value to (vector) 0
- $\phi(i) = e_i \in \mathbb{R}^{k-1}$, $i = 1, \ldots, k-1$
- example: $\mathcal{U} = \{\text{True}, \text{False}\}$ with False as default

\[
\phi(\text{TRUE}) = 1, \quad \phi(\text{FALSE}) = 0
\]

(a common embedding of Booleans into $\mathbb{R}$)
Ordinal data

- ordinal data is categorical, with an order
- example: \textit{Likert scale}, with values

\begin{center}
\text{STRONGLY DISAGREE, DISAGREE, NEUTRAL, AGREE, STRONGLY AGREE}
\end{center}

- can embed into $\mathbb{R}$ with values $-2, -1, 0, 1, 2$
- or treat as categorical, with one-hot embedding into $\mathbb{R}^5$
- example: number of bedrooms in house
  - can be treated as a real number
  - or as an ordinal with (say) values $1, \ldots, 6$
Feature engineering
Feature engineering

- basic idea:
  - start with some features
  - then process or transform them to produce new (‘engineered’) features
  - use these new features in your predictor

- was it a good idea? did it improve your predictor?
  - train your model with original features and validate performance
  - train your model with new features and validate performance
  - if performance with new features is better, your feature engineering was successful
Types of feature transforms

- **modify individual features**: replace original feature $x_i$ with modified or transformed feature $x_i^{\text{new}}$
  - simple example: standardize, $x_i^{\text{new}} = (x_i - \mu_i)/\sigma_i$

- **create multiple features from each original feature**
  - simple example: powers, replace $x_i$ with $(x_i, x_i^2, \ldots, x_i^q)$

- **create new features from multiple original features**
  - simple example: product, $x_i^{\text{new}} = x_k x_l$
**Gamma-transform**

- **γ-transform**: \( x_{i}^{\text{new}} = \text{sign}(x_{i})|x_{i}|^{\gamma_{i}}, \gamma_{i} > 0 \)
Clipping

- **winsorize** or **clip** or **saturate** with lower and upper clip levels $l_i, u_i$

\[
x_i^{\text{new}} = \begin{cases} 
    u_i & x_i > u_i \\
    x_i & l_i \leq x_i \leq u_i \\
    l_i & x_i < l_i 
\end{cases}
\]
replace $x_i$ with $(x_i, x_i^2, \ldots, x_i^q)$
Split into positive and negative parts

- replace $x_i$ with $((x_i)_+, (x_i)_-)$
- or, split into negative, middle, and high values: replace $x_i$ with
  
  $((x_i + 1)_-, \text{sat}(x_i), (x_i - 1)_+)$

$\text{sat}(a) = \min(1, \max(x, -1))$ is the saturation function
Creating new features from multiple original features

- can be used to model *interactions* among features

- examples: for $i < j$
  - maximum: $\max(x_i, x_j)$
  - product: $x_i x_j$

- example: all monomials up to degree 3 of $(x_1, x_2)$:

  \[
  (x_1, x_2, x_1^2, x_1 x_2, x_2^2, x_1^3, x_1^2 x_2, x_1 x_2^2, x_2^3)
  \]

  linear model with these features gives arbitrary degree 3 polynomial of $(x_1, x_2)$
Interpreting products of features as interactions

- Suppose $x_i$ are Boolean, with values 0, 1, for $i = 1, \ldots, d$, e.g., representing patient symptoms.
- Create new 'interaction' features $x_i x_j$, for $i < j$, of which there are $d(d - 1)/2$.
- Linear regression model (for $d = 3$) is

$$
\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_{12} x_1 x_2 + \theta_{13} x_1 x_3 + \theta_{23} x_2 x_3
$$

- $\theta_1$ is the amount our prediction goes up when $x_1 = 1$.
- $\theta_3$ is the amount our prediction goes up when $x_3 = 1$.
- $\theta_{13}$ is the amount our prediction goes up when $x_1$ and $x_3$ are both 1 (in addition to $\theta_1 + \theta_3$).
- E.g., with $\theta_{13}$ large, the simultaneous presence of symptoms 1 and 3 makes our estimate go up a lot.
Quantizing

- specify *bin boundaries* $b_1 < b_2 < \cdots < b_k$

- partitions into *bins* or *buckets* $(-\infty, b_1]$, $(b_1, b_2]$, $\ldots$ $(b_{k-1}, b_k]$, $(b_k, \infty)$

- common choice of bin boundaries: quantiles of $x_i$, e.g., deciles

- replace $x_i$ with

$$
\begin{align*}
& e_1 & & x_i \leq b_1 \\
& e_2 & & b_1 < x_i \leq b_2 \\
& \vdots & & \vdots \\
& e_k & & b_{k-1} < x_i \leq b_k \\
& e_{k+1} & & b_k < x_i
\end{align*}
$$

*i.e.*, $x_i$ maps to $e_l$, if $x_i$ is in bin $l$
Feature engineering pipeline

- feature transformations can be done multiple times
- start by embedding original record $u$ into vector feature $x^0 \in \mathbb{R}^{d^0}$ using $\tilde{\phi}$, $x^0 = \tilde{\phi}(u)$
- superscript 0 in $x^0$ and $d^0$ means starting point for feature engineering
- transform $x^0$ using a feature engineering transform $T^1$, to get $x^1 = T^1(x^0) \in \mathbb{R}^{d^1}$
- superscript 1 in $x^1$ and $d^1$ means ‘first step’ of feature engineering
- repeat $M$ times to get final embedding $x = x^M = \phi^M(x^{M-1})$
- final feature map is a composition:
  \[ \phi = T^M \circ T^{M-1} \circ \cdots \circ T^1 \circ \tilde{\phi} \]
- called feature engineering pipeline
Automatic feature generation
Hand crafted versus automatic features

- Features and feature engineering described above generally done by hand, using experience
- Can also develop feature mappings automatically, directly from some data
- Examples: word2vec, vgg16 were developed automatically (from very large data sets)
- We’ll later see some of these methods (PCA, neural nets, …)
Summary
raw features are mapped to vectors for subsequent processing

feature maps can range from simple to complex

use validation to choose among different candidate feature maps

sometimes the original feature map is followed by subsequent transformations, called feature engineering

we’ll see later how feature mappings can be derived from data, as opposed to by hand