MATERIAL FOR EMBEDDINGS
MACHINE LEARNING E20

Mathias Ravn Tversted
Department of Computer Science
Aarhus University
Åbogade 34, 8200 Aarhus N
Tversted@post.au.dk

December 27, 2020

Contents

1 Embeddings - what even are they?

Talking points:

- Good representations of data
- Potentially in lower dimension
- Some data may be redundant
- Potentially reducing number of features
- Can speed up algorithms, or enable visualization

2 Principal Component Analysis

We want to project feature vectors onto k-dimensional subspace in order to reduce the number of features. Specifically, we want to find k orthogonal unit vectors $z_1, \cdots, z_k$ that form a new orthonormal basis. We can then write our x in this new basis as

$$\hat{x} = \begin{pmatrix} x^T z_i \\ \vdots \\ x^T z_k \end{pmatrix} \tag{1}$$

We will then project our feature vectors x to

$$p = \sum_{i=1}^{k} (x^T z_i) z_i \tag{2}$$

The problem is picking a basis that keeps a high signal/noise ration for the dataset. One common example is 2 features where data is plotted along a linear path. Here the data could suffieciently be represented in 1 dimension. We have to pick the basis that fits best for this data. We have two ways to express what we want.

1. Maximizing Variance (maximising information)
2. Minimizing Loss (minimising information lost)

When projection on to a subspace, we want to maximize variance in order to "spread out" points in order to preserve as much information about points as possible. The mean is defined as

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{3}$$

And variance

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2 \tag{4}$$

2.1 PCA algorithm (greedy)

- For $i = 1, \cdots, k$
- Let z_i be a unit vector such that
- – z_i is orthogonal to $z_1, \cdots, z_{i-1}$
  – sample variance of $x_1^T z_i, \cdots, x_n^T z_i$ is largest among all unit vectors z that are orthogonal to $z_1, \cdots, z_{n-1}$

2.2 PCA - using eigenvectors

Another goal can be to minimize the Loss defined as $\sum_{j=1}^{n} \|x_j - p_{x_j}\|^2$. The loss is $x - p_x$, which is the information lost due to the projection on to the subspace.

A way to select a the best basis Z is as follows

- For $i = 1, \cdots, k$
- Let z_i be a unit vector such that
  - Pick z_i to be the eigenvector of $X^T X$ that corresponds to the ith largest eigenvalue.

This minimises the "squared loss" of projections

$$\sum_{j=1}^{n} \|x_j - p_{x_j}\|^2 \tag{5}$$

Where x can be written in this basis

$$\hat{x} = \begin{pmatrix} x^T z_i \\ \vdots \\ x^T z_k \end{pmatrix} \tag{6}$$

MÅSKE HA NOGLE NEDSLAG FRA BEVISET DER GIVER EN ID´E OM HVORFOR AT EGENVEKTORER ER DEN BEDSTE LØSNING

2.3 Practical considerations

Eigenvectors are effected by scaling of matrix. Outliers can dominate the matrix. If the j'th feature is scaled by a large factor c

$$X = \begin{pmatrix} x_{1,1} & \cdots & cx_{1,j} & \cdots & x_{1,d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & \cdots & cx_{n,j} & \cdots & x_{n,d} \end{pmatrix} \tag{7}$$

As j'th row in $X^T X$ scales by c, j'th column scales by c and j'th diagonal scales by c^2, and so

$$X^T X \approx c^2 \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & 1 & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \tag{8}$$

So $X^T X e_j \approx c^2 e_j$, so z_1 tends to e_j, and so j'th feature stays as it is.

The solution is to scale each feature such that the mean over the data setis 0 and the variance is 1.

Feaure scaling for 0-mean: $X_{i,j} = X_{i,j} - \sum_{i=1}^{n} \frac{x_{i,j}}{n}$ (subtract mean of feature)
Making variance 0: If mean is 0, then variance is $\sum_{i=1}^{n} (x_{i,j} - \mu)^2 = \sum_{i=1}^{n} x_{i,j}^2$. By scaling entries with $(\sum_{i=1}^{n} x_{i,j}^2)^{-\frac{1}{2}}$ then $\sum_{i=1}^{n} x_{i,j}^2$ becomes 1.

Remember, that for prediction you need to subtract μ_j from the j'th feature, and scale by σ_j before computing $x^T Z$.

3 Johnson-Lindenstrauss

PCA does not maintain distance between datapoints after projection. Johnson-Lindenstrauss is an embedding that tries to respect this and make a bound on how much the distances can be off.

Formally: For x_i, x_j, our target is $\|x_i - x_j\|^2 \approx \|f(x_i) - f(x_j)\|^2$

- Given n feature vectors $x_1, \cdots, x_n \in R^d$
- For any precision $0 < \epsilon < \frac{1}{2}$, there exists a mapping $f : R^d \to R^k$ with $k = \Theta(\epsilon^{-2} ln(n))$, such that for all x_i, x_j
- $(1-\epsilon)\|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1+\epsilon)\|x_i - x_j\|^2$

Informally: Any n feature vectors can be embedded into $\theta(\epsilon^{-2} \ln n)$ dimensions while preserving pairwise dististances within a factor $(1 \pm \epsilon)$

3.1 Johnson-Lindenstrauss Transform

Project on to random subspace of dimension $k = \theta(\epsilon^{-2} \ln(n))$ spanned by k orthonormal vectors. Scale vectors up by a factor of $\sqrt{d/k}$.

3.2 Proof sketch

Proof is not presented, but a quick summary:

Let A by a $k \times d$ matrix with each entry $a_{i,j} \sim N(0,1)$. Scale A with $\frac{1}{\sqrt{k}}A$.
Step 1: Show that when $k = \theta(\epsilon^{-2} \ln(\frac{1}{\delta}))$ it holds for any unit vector x that

$$Pr[|\|Lx\|^2 - 1| \geq \epsilon] \leq \delta \tag{9}$$

(norm of x is preserved)

Step 2: Set $\delta = 1/n^3$ and reach that all pairwise distances between vectors $x_1, \cdots, x_n$ are preserved with prob $\geq 1 - 1/n$

It all comes together when $k = \theta(\epsilon^{-2} \ln(\frac{1}{\delta})) = \theta(\epsilon^{-2} \ln(n))$

Proof was not practised due to time, and because presenting it would not show breath of relevant knowledge.

4 PCA vs Johnson-Lindenstrauss

Johnson-Lindenstrauss preserves distances, but PCA guarantees best reconstruction of XZZ^T, but it is much easier to compute the Johnson-Lindenstrauss transformation. Z does not depend on the data, and can be chosen beforehand.

5 Autoencoders

Another way to embed features in a subspace is using neural networks. If we can construct a neural network that computes the identity function which has a bottleneck (a layer whose dimension is stricly smaller than the input and output dimensions), then this bottleneck layer is an embedding of our input data in a lower dimension.

Embedding is then computed with $\phi(xM_1 + b_1)$, the input is decoded with $y = \phi(xM_1 + b_1)M_2 + b_2$.

Let $L(x, y) = \sum_{i=1}^{d}(x_i - y_i)^2$ be a pointwise loss function, then we can minimise loss on training data as $\frac{1}{n}\sum_{j=1}^{n} L(x^j, ae(x^j))$. This can be trained using a normal mini-batch SGD.

5.1 PCA as a special case of autoencoding

If Z is an embedding matrix from PCA (top k eigenvectors of X^TX as columns), then

$$M_1 = Z, M_2 = Z^T \tag{10}$$

With no bias and identity activation. Then $xZ = \phi(xM_1)$ and $y = \phi(xM_1)M_2 = xZZ^T$.

5.2 Combating overfitting

If little training data available, then overfitting is possible. This can be combated by making it insensitive to noise.

Let $\hat{x} = x + n$ where n is random noise, then we can measure loss $L(x, ae(\hat{(}x)))$ against the original input. The idea is that it will learn to extract the actual features of the data.

5.3 Making *embedding* insensitive to changes in put

This contrasts with making the output insensitive.

Sensitivity is defined as the partial derivatives of z w.r.t to x is large at inputs. This means we can change

$$L = \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{d}(x_i^j - ae(x^j)_i)^2 \tag{11}$$

to

$$L + \lambda\sum_{j=1}^{n}\sum_{i=1}^{k}\sum_{h=1}^{d}\left(\frac{\partial z_i}{\partial x_h^j}\right)^2 \tag{12}$$

Here λ is a hyperparameter which weighs the loss of relative large gradients. z refers to the bottleneck layers. z has k features.

6 Bag of words

The problem: Machine Learning on text. So far, all ML techniques have been on things that map nicely to numbers.

The solution: Algebraic representation of text. For example, words map to a one-hot vector where each entry corresponds to a word in a dictionary. For a document, we can then sum all of these vectors together to gain a vector representation of the words in that document.

6.1 N-Grams

One-hot vectors do not encode information about the relationship between words. One way to do this is *n-grams* where we group all n consecutive words together. We can then use a vector which has an entry for each $n - tuple$ in the dictionary.

Choice of n: Too small, lose ability to represent meaning, too big, and few texts have overlap.

6.2 Sparse vector representations

An obvious problem with this approach is that vectors will be large sparse (unncessarily slow algorithms) and extremely sparse (VC dimension issues, hypothesis set is explosively enourmous). A text with m words will have $(m - n + 1)$ n-grams. We will therefore need *sparse vector representations*.

Two ways to solve this problem are

- Ordered list of non-zero entries
- Map mapping n-grams to counters

For example

$$(0, 1, 0, 1, \cdots) = \{(1, 1), (3, 1)\} \tag{13}$$

or

$$\{\text{"giant helicopters":1, "killer robots:"2}\} \tag{14}$$

For n-grams.

This will still break most algorithms, but SVM is an example of something that still works because predictions are

$$\text{sign}(\sum_{a_i \neq 0} a_i y_i K(x_i, x) + b) \tag{15}$$

which rely on inner products of sparse vectors (obviously fast!). Likewise, finding w and b which only on inner products (formulas not shown because they are hideous).

6.3 Inner products of sparse vectors

Linear time (|a| + |b|) algorithm to compute $\langle x, y \rangle$.

Keep two index counters $i, j = 0$. Initialise $sum = 0$. Check if both have an index i, j, if so, multiply and add to sum, otherwise advance lowest index counter.

For dictionaries it can be done in $\min(|a|, |b|)$ but did not solve the exercise, but it probably involves cleverly matching only the elements of the smallest one against the big one.

7 TF-IDF

7.1 Inverse Document Frequency (IDF)

Rare words probably have more meaning as to what a document is about. We will weigh words by how frequently they appear.

Let $d_1, \cdots, d_n$ be a collection of documents. let t be a word (term), then the document frequency is

$$df(t) = \frac{|\{d_i : t \in d_i\}|}{n} \tag{16}$$

and inverse document frequency is

$$idf(t) = \ln(\frac{1}{df(t)}) = \ln(\frac{n}{|\{d_i : t \in d_i\}|}) \tag{17}$$

Rare words weigh higher under this scheme.

Words in every document have $idf(t) = 0$ and words in a single document have $idf(t) = \ln(n)$.

7.2 Normalization of frequencies

Document length may not be related to length, and longer documents accumulate more values. So a solution is to normalize by length.

Let $f_{t,d}$ be the number of occurences of t in d. Then let the term frequency be

$$tf(t, d) = \frac{f_{t,d}}{|d|} \tag{18}$$

7.3 Term-Frequency Inverse Document Frequency (TF-IDF)

Now we are ready to define TF-IDF. Let t in vector document d be

$$tf(t, d) \cdot idf(t) = \frac{f_{t,d}}{|d|} \cdot \ln(\frac{n}{|\{d_i : t \in d_i\}|}) \tag{19}$$

Here words occuring often have larger weights, and rare words are weighed higher than common words.

An alternative is to use

$$tf(t,d) = \ln(1 + f_{t,d}) \tag{20}$$

Which is 0 when $f_{t,d}$, which gives logarithmatic scaling on occurences.

8 Feature hashing

We can try to reduce the number of features in order to combat big, sparse vectors that prevent us from using normal ML algorithms.

We will use feature hashing, where we map some long feature vector with d vectors to a much shorter one with k features, with $k << d$.

Let $[d] = \{0, \cdots, d-1\}$, and h, g be hash functions such that

$$h : [d] \to [k] \tag{21}$$

and

$$g : [d] \to \{-1, 1\} \tag{22}$$

Where

$$u_t = c \implies f(u)_{h(t)} += c \cdot g(t) \tag{23}$$

Remember that $f(u)$ has k entries and u has d entries.

We want a hash function with small chance of collision $Pr_h[h(x) = h(y)] = \frac{1}{k}$. The hash function g should have $g(x)$ and $g(y)$ independent and uniform random for $x \neq y$.

Let u, v be two vectors we consider as inputs. Then coordinate j contributes $u_j v_j$ to the inner product. They will end up in the same bucket under map h. This will contribute $g(j)u_j g(j)v_j = u_j v_j$ after hashing. ($g(j) = 1$ gives $1u_j 1v_j = u_j v_j$ and $g(j) = -1$ gives $-1u_j - 1v_j$).

Coordinates i and j contribute 0 to inner product when $i = j$. They may end up in the same bucket, and contribute $g(i)u_i g(j)v_j$, but if signs are independent, we *expect* 0.

There is some connection to the Johnson-Lindenstrauss transformation, where the entries a_{ij} could also be chosen from $\{-1, 1\}$, but feature hashing does not preserve distances.

9 Co-occurence matrix

If we want to create word-vectors that preserve word-similarity, then we can create them by looking at similar sentence structures.

For training documents $d_1, \cdots, d_n$, create a matrix with an entry for each pair of words. Then count co-occurences (by way of context windows) (t_1, t_2) in $d_1, \cdots, d_2$.

Our matrix is A is symmetric and real, and very large. As with PCA, we can do eigendecomposition in order to reduce its size.

For any real symmetric $n \times n$ matrix A, we can write it as

$$A = EDE^T \tag{24}$$

Where D is a diagonal matrix with the i'th diagonal entry being the i'th largest eigenvalue and E is an orthonormal matrix, with the i'th column being the unit eigenvector μ_i corresponding to λ_i, as in $A\mu_i = \lambda_i \mu_i$.

Use rows as embeddings of word.

This approach is slow and memory heavy.

10 Continuous Bag of Words

From documents $d_1, \cdots, d_n$ generate training data:

- Construct dictionary with all the words that occur
- For each document and each word t in document:
- – generate training example with context as input features and t as target label
  – All one-hot encoded as indices into dictionary

11 Skip-gram