COMS 4771 Spring 2015

Features

Daniel Hsu *Warning: notes are incomplete and unchecked for correctness*

1 Feature expansions

In many applications, no linear classifier over the "raw" set of features will perfectly separate the data. One recourse is to find additional features that are predictive of the label. This is called *feature engineering*, and is often a substantial part of the job of a machine learning practitioner. In some applications, it is possible to "throw in the kitchen sink"—i.e., include all possible features that might possibly be relevant. For instance, in document classification, one can include a feature for each possible word in the vocabulary that indicates whether that word is present in the given document (or counts the number of occurrences). One can also include a feature for each possible pair of consecutive words ("bi-grams"), each possible triple of consecutive words ("tri-grams"), and so on. In general, it is common to automatically generate features based on existing features $\boldsymbol{x} \in \mathbb{R}^d$, such as quadratic interaction features $\boldsymbol{x} \mapsto (x_1x_2, x_1x_3, \ldots, x_1x_d, x_2x_3, \ldots, x_{d-1}x_d) \in \mathbb{R}^{\binom{d}{2}}$, as well as higher-order interaction features.

The main drawback of these "kitchen sink" feature expansions is that it may be computationally expensive to work explicitly in the expanded feature space. Fortunately, there are some ways around this.

1.1 The kernel trick

Suppose $\boldsymbol{\phi}\colon \mathbb{R}^d \to \mathbb{R}^D$ is the mapping of the base feature vector $\boldsymbol{x}$ to some expanded feature representation $\boldsymbol{\phi}(\boldsymbol{x})$ (where possibly $D \gg d$). Imagine running the Online Perceptron algorithm in this expanded feature space $\mathbb{R}^D$. Naïvely, it takes $\Theta(D)$ time to compute $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{w} \rangle$ by explicitly working with the expanded feature representation. However, in some cases, this can be greatly improved. Recall that at any point in the algorithm, Online Perceptron's weight vector is equal to

$$\boldsymbol{w} = \sum_{(\boldsymbol{x},y)\in\mathcal{M}} y\boldsymbol{\phi}(\boldsymbol{x})$$

where $\mathcal{M}$ is the set of examples on which Online Perceptron has so far made mistsakes (and hence used to update the weight vector). Therefore,

to compute a prediction on a new point $\boldsymbol{x}'$ (with feature expansion $\boldsymbol{\phi}(\boldsymbol{x}')$), we need to compute the inner product

$$\langle \boldsymbol{\phi}(\boldsymbol{x}'), \boldsymbol{w} \rangle = \sum_{(\boldsymbol{x},y)\in\mathcal{M}} y \langle \boldsymbol{\phi}(\boldsymbol{x}'), \boldsymbol{\phi}(\boldsymbol{x}) \rangle.$$

This can be done in time $\Theta(|\mathcal{M}| \cdot \tau_K)$, where τ_K is the time to compute the inner product $\langle \boldsymbol{\phi}(\boldsymbol{x}'), \boldsymbol{\phi}(\boldsymbol{x}) \rangle$. Naïvely, τ_K appears to be $\Theta(D)$, so this may not appear to be an improvement. But fortunately, in many cases of interest, τ_K is much smaller. This is called the *kernel trick*.

All degree $\leq$2 interaction features. Consider all $D := 1 + 2d + \binom{d}{2}$ products of at most two features:

$$\begin{aligned}\boldsymbol{\phi}(\boldsymbol{x}) := \Big(&1, \sqrt{2}x_1, \sqrt{2}x_2, \ldots, \sqrt{2}x_d, \\ &x_1^2, x_2^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots, \sqrt{2}x_1x_d, \sqrt{2}x_2x_3, \ldots, \sqrt{2}x_{d-1}x_d\Big).\end{aligned}$$

There is some seemingly arbitrary scaling by $\sqrt{2}$ in some parts, but this is not a critical problem for linear classifiers. We can compute $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle$ in $\tau_K = O(d)$ time:

$$\begin{aligned}(1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle)^2 &= 1 + 2\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle^2 \\ &= 1 + \sum_{i=1}^{d} (\sqrt{2}x_i)(\sqrt{2}x_i') + \sum_{i=1}^{d} x_i^2 x_i'^2 + \sum_{1\leq i<j\leq d} (\sqrt{2}x_i x_j)(\sqrt{2}x_i' x_j') \\ &= \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle.\end{aligned}$$

Products of all feature subsets. Consider all $D := 2^d$ products of subsets of features:

$$\boldsymbol{\phi}(\boldsymbol{x}) := \left(\prod_{i\in S} x_i : S \subseteq [d] \right).$$

We can compute $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle$ in $\tau_K = O(d)$ time:

$$\begin{aligned}\prod_{i=1}^{d} (1 + x_i x_i') &= \sum_{S\subseteq[d]} \prod_{i\in S} (x_i x_i') \\ &= \sum_{S\subseteq[d]} \prod_{i\in S} x_i \prod_{i\in S} x_i' \\ &= \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle.\end{aligned}$$

An infinite dimensional feature expansion. For any $\sigma > 0$, there is some infinite feature expansion $\boldsymbol{\phi}\colon \mathbb{R}^d \to \mathbb{R}^\infty$ (involving Hermite polynomials of all orders) such that

$$\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2} \right),$$

which can be computed in $\tau_K = O(d)$ time. This inner product is called the *Gaussian kernel with bandwidth* σ.

Kernels. A positive definite *kernel function* $K\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a symmetric function that satisfies the following property: for any $n \geq 1$, any $c_1, c_2, \ldots, c_n \in \mathbb{R}^n$, and any $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0.$$

That is, for any set of data points $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, the symmetric matrix whose (i, j)-th entry is $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is positive semi-definite. For any positive definite kernel K, there is some feature mapping $\boldsymbol{\phi}$ such that $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle = K(\boldsymbol{x}, \boldsymbol{x}')$ for any pair $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$. The mapping is not necessarily to $\mathbb{R}^D$ for any finite D, but rather to an inner product space called a *Reproducing Kernel Hilbert Space* (RKHS). Also, kernel functions need not be defined over Euclidean spaces $\mathbb{R}^d$; they can in fact be (and have been) defined over general sets (e.g., strings, trees, genetic sequences).

2 Kernel approximation

A drawback of using the kernel trick with Perceptron is that the representation of the weight vector $\boldsymbol{w} \in \mathbb{R}^D$ may grow with the number of the training examples. One way to alleviate this drawback is to limit the number of data points used in the representation of the weight vector $\boldsymbol{w} \in \mathbb{R}^D$.

An alternative is to approximate the kernel function directly by finding a mapping $\boldsymbol{z}\colon \mathbb{R}^d \to \mathbb{R}^m$ such that

$$\langle \boldsymbol{z}(\boldsymbol{x}), \boldsymbol{z}(\boldsymbol{x}') \rangle \approx K(\boldsymbol{x}, \boldsymbol{x}').$$

This can often be done with $m \ll D$ (especially when $D = \infty$) using *randomized feature expansions.*

Shift-invariant kernels. A very popular class of kernels are shift-invariant kernels: positive definite kernels of the form $K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x} - \boldsymbol{x}', \boldsymbol{0}) =: K(\boldsymbol{x} - \boldsymbol{x}')$. This includes Gaussian kernels, Laplacian kernels, Cauchy kernels, and many others. The randomized approximation is grounded in *Bochner's theorem*, which states that for any continuous shift-invariant kernel $K\colon \mathbb{R}^d \to \mathbb{R}$ on $\mathbb{R}^d$, then for some non-negative function $\lambda\colon \mathbb{R}^d \to \mathbb{R}_+$,

$$K(\boldsymbol{\delta}) = \int_{\mathbb{R}^d} \exp(i\langle \boldsymbol{\delta}, \boldsymbol{\omega} \rangle) \lambda(\boldsymbol{\omega}) d\boldsymbol{\omega}$$

where $i = \sqrt{-1}$. This says that K is the inverse Fourier transform of a non-negative function λ. Given any continuous shift-invariant kernel K, we can obtain λ via the Fourier transform:

$$\lambda(\boldsymbol{\omega}) = \int_{\mathbb{R}^d} \exp(-i\langle \boldsymbol{\delta}, \boldsymbol{\omega} \rangle) K(\boldsymbol{\delta}) d\boldsymbol{\delta}.$$

By appropriately scaling the kernel, the function λ above will be a probability density.

For example, if $K(\boldsymbol{\delta}) = \exp(-\|\boldsymbol{\delta}\|^2/(2\sigma^2))/(2\pi)^d$ is the (scaled) Gaussian kernel, then

$$\lambda(\boldsymbol{\omega}) = \frac{1}{(2\pi\sigma^{-2})^{d/2}} \exp\left(-\frac{\|\boldsymbol{\omega}\|^2}{2\sigma^{-2}}\right),$$

which is simply the multivariate Gaussian density with spherical covariance $\sigma^{-2}\boldsymbol{I}$. Therefore, we can write the (scaled) Gaussian kernel as the following expectation over a Gaussian random vector $\boldsymbol{\omega} \sim \mathrm{N}(\boldsymbol{0}, \sigma^{-2}\boldsymbol{I})$:

$$\begin{aligned} K(\boldsymbol{x} - \boldsymbol{x}') &= \mathbb{E}\big(\exp\big(i\langle \boldsymbol{x} - \boldsymbol{x}', \boldsymbol{\omega} \rangle\big)\big) \\ &= \mathbb{E}\big(\exp(i\langle \boldsymbol{x}, \boldsymbol{\omega} \rangle) \exp\big(-i\langle \boldsymbol{x}', \boldsymbol{\omega} \rangle\big)\big) \\ &= \mathbb{E}\big(\cos(\langle \boldsymbol{x}, \boldsymbol{\omega} \rangle) \cos(\langle \boldsymbol{x}', \boldsymbol{\omega} \rangle) + \sin(\langle \boldsymbol{x}, \boldsymbol{\omega} \rangle) \sin(\langle \boldsymbol{x}', \boldsymbol{\omega} \rangle)\big). \end{aligned}$$

The same holds for other appropriately scaled continuous shift-invariant kernels K when $\boldsymbol{\omega} \sim \lambda$ for the corresponding probability density λ.

Random Fourier features. Let λ be the probability density obtained as the Fourier transform of an appropriately scaled continuous shift-invariant kernel K. Suppose we randomly draw $\boldsymbol{\omega} \sim \lambda$, and then define the feature map $\boldsymbol{z}_{\boldsymbol{\omega}}\colon \mathbb{R}^d \to \mathbb{R}^2$ by

$$\boldsymbol{z}_{\boldsymbol{\omega}}(\boldsymbol{x}) := (\cos(\langle \boldsymbol{x}, \boldsymbol{\omega} \rangle), \sin(\langle \boldsymbol{x}, \boldsymbol{\omega} \rangle)).$$

Then for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$,

$$\begin{aligned}\mathbb{E}\langle \boldsymbol{z}_{\boldsymbol{\omega}}(\boldsymbol{x}), \boldsymbol{z}_{\boldsymbol{\omega}}(\boldsymbol{x}')\rangle &= \mathbb{E}\big(\cos(\langle \boldsymbol{x}, \boldsymbol{\omega}\rangle)\cos(\langle \boldsymbol{x}', \boldsymbol{\omega}\rangle) + \sin(\langle \boldsymbol{x}, \boldsymbol{\omega}\rangle)\sin(\langle \boldsymbol{x}', \boldsymbol{\omega}\rangle)\big) \\ &= K(\boldsymbol{x} - \boldsymbol{x}').\end{aligned}$$

Now suppose we draw m independent random vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m \sim \lambda$, and define $\boldsymbol{z}\colon \mathbb{R}^d \to \mathbb{R}^{2m}$ by

$$\boldsymbol{z}(\boldsymbol{x}) := \sqrt{\frac{1}{m}}(\cos(\langle \boldsymbol{x}, \boldsymbol{\omega}_1\rangle), \sin(\langle \boldsymbol{x}, \boldsymbol{\omega}_1\rangle), \ldots, \cos(\langle \boldsymbol{x}, \boldsymbol{\omega}_m\rangle), \sin(\langle \boldsymbol{x}, \boldsymbol{\omega}_m\rangle)).$$

Then, again, for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$,

$$\begin{aligned}&\mathbb{E}\langle \boldsymbol{z}(\boldsymbol{x}), \boldsymbol{z}(\boldsymbol{x}')\rangle \\ &= \mathbb{E}\left(\frac{1}{m}\sum_{i=1}^{m} \cos(\langle \boldsymbol{x}, \boldsymbol{\omega}_i\rangle)\cos(\langle \boldsymbol{x}', \boldsymbol{\omega}_i\rangle) + \sin(\langle \boldsymbol{x}, \boldsymbol{\omega}_i\rangle)\sin(\langle \boldsymbol{x}', \boldsymbol{\omega}_i\rangle)\right) \\ &= K(\boldsymbol{x} - \boldsymbol{x}').\end{aligned}$$

The variance of $\langle \boldsymbol{z}(\boldsymbol{x}), \boldsymbol{z}(\boldsymbol{x}')\rangle$ is roughly $O(1/m)$ since the $\{\boldsymbol{\omega}_i\}_{i\in[n]}$ are independent. Therefore if $m = \Omega(1/\epsilon^2)$, then $|\langle \boldsymbol{z}(\boldsymbol{x}), \boldsymbol{z}(\boldsymbol{x}')\rangle - K(\boldsymbol{x} - \boldsymbol{x}')| \le \epsilon$ with high probability. (This can be made precise using via Hoeffding's inequality.)

To recap, we have shown that using a relatively large number m of these *random Fourier features* based on an appropriate probability distribution, we are able to approximate the kernel function using standard inner products in $\mathbb{R}^m$ (well, $\mathbb{R}^{2m}$). Moreover, this mapping $\boldsymbol{z}$ is constructed in a completely data-oblivious way.

3 Dimensionality reduction

In some cases, it is feasible to explicitly compute the feature expansion $\boldsymbol{\phi}\colon \mathbb{R}^d \to \mathbb{R}^D$ because it has a compact implicit representation. For instance, if the base feature vectors $\boldsymbol{x} \in \mathbb{R}^d$ are sparse (i.e., they have only a small number of non-zero entries), then low-degree interaction features will also be (relatively) sparse. However, it may still be undesirable to maintain a high-dimensional weight vector in $\mathbb{R}^D$. A possible recourse is dimensionality reduction: construct a mapping $\boldsymbol{A}\colon \mathbb{R}^D \to \mathbb{R}^m$ of high-dimensional feature vectors $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^D$ to lower-dimensional vectors in $\mathbb{R}^m$ (for some $m \ll D$), and then learn a linear classifier in $\mathbb{R}^m$. As before, we would like the mapping $\boldsymbol{A}$ to approximately preserve inner products:

$$\langle \boldsymbol{A}(\boldsymbol{\phi}(\boldsymbol{x})), \boldsymbol{A}(\boldsymbol{\phi}(\boldsymbol{x}'))\rangle \approx \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}')\rangle.$$

Henceforth, we ignore the distinction between $\boldsymbol{x}$ and $\boldsymbol{\phi}(\boldsymbol{x})$, and simply discuss techniques for constructing mappings $\boldsymbol{A}$ from $\mathbb{R}^d$ to $\mathbb{R}^m$. In fact, the mappings we construct will be *linear maps*, and hence will be representable as matrices $\boldsymbol{A} \in \mathbb{R}^{m \times d}$.

Gaussian random matrix. Again, we shall use randomization to construct the mapping $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ in a data-oblivious way. Fix a pair of vectors $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$. We would like the mapping $\boldsymbol{A}$ to approximately preserve their inner product: $\langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}' \rangle \approx \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$. It turns out that because of the structure of Euclidean space, it suffices to preserve the (squared Euclidean) lengths of $\boldsymbol{x} + \boldsymbol{x}'$ and $\boldsymbol{x} - \boldsymbol{x}'$. This is because for any pair of vectors $\boldsymbol{u}, \boldsymbol{v}$ in Euclidean space,

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \frac{1}{4}\big(\|\boldsymbol{u} + \boldsymbol{v}\|^2 - \|\boldsymbol{u} - \boldsymbol{v}\|^2\big).$$

The upshot is that if $\|\boldsymbol{A}(\boldsymbol{x}+\boldsymbol{x}')\|^2 \approx \|\boldsymbol{x}+\boldsymbol{x}'\|$ and $\|\boldsymbol{A}(\boldsymbol{x}-\boldsymbol{x}')\|^2 \approx \|\boldsymbol{x}-\boldsymbol{x}'\|$, then $\langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}' \rangle \approx \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$.

Our first approach is to let $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ be a matrix of i.i.d. Gaussian random variables, each with variance $1/m$.

Theorem 1. *Let $\boldsymbol{A} := (1/\sqrt{m})\boldsymbol{Z}$, where $\boldsymbol{Z}$ is a $k \times d$ matrix of i.i.d.* $\mathrm{N}(0,1)$ *random variables. Fix any $\boldsymbol{v} \in \mathbb{R}^d$ and $\delta \in (0,1)$. With probability at least $1 - \delta$,*

$$\|\boldsymbol{v}\|^2 \left(1 - 2\sqrt{\frac{\ln(2/\delta)}{m}}\right) \leq \|\boldsymbol{A}\boldsymbol{v}\|^2 \leq \|\boldsymbol{v}\|^2 \left(1 + 2\sqrt{\frac{\ln(2/\delta)}{m}} + \frac{2\ln(2/\delta)}{m}\right).$$

Proof. Fix $\boldsymbol{v} \in \mathbb{R}^d$ and $\delta \in (0,1)$. Without loss of generality, we may assume $\boldsymbol{v}$ is a unit vector. Let $Y_i := \sum_{j=1}^d Z_{i,j} v_j$ for each $i \in [m]$, so

$$\|\boldsymbol{A}\boldsymbol{v}\|^2 = \frac{1}{m}\|\boldsymbol{Z}\boldsymbol{v}\|^2 = \frac{1}{m}\sum_{i=1}^m \left(\sum_{j=1}^d Z_{i,j} v_j\right)^2 = \frac{1}{m}\sum_{i=1}^m Y_i^2.$$

Since Y_i is a linear combination of standard Gaussian random variables $Z_{i,1}, Z_{i,2}, \ldots, Z_{i,d}$, it is itself distributed as a Gaussian random variable, with mean zero and variance $\sum_{j=1}^d v_j^2 = 1$, i.e., $Y_i \sim \mathrm{N}(0,1)$. The independence of the $Z_{i,j}$ implies that the Y_i are independent. The sum-of-squares of m independent standard Gaussian random variables $\sum_{i=1}^m Y_i^2$ is distributed according to the χ^2 distribution with m degrees of freedom. In expectation,

$\sum_{i=1}^{m} Y_i^2$ is m. Moreover,

$$\Pr\left(\sum_{i=1}^{m} Y_i^2 > m + 2\sqrt{mt} + 2t\right) \leq e^{-t}$$

and

$$\Pr\left(\sum_{i=1}^{m} Y_i^2 < m - 2\sqrt{mt}\right) \leq e^{-t}$$

for any $t > 0$. Letting $t := \ln(2/\delta)$ and combining these probability tail bounds with a union bound completes the proof. □

Suppose $\|\boldsymbol{x}\|$ and $\|\boldsymbol{x}'\|$ are both $O(1)$. Theorem 1 implies that if $m = \Omega(1/\epsilon^2)$, then $|\langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}'\rangle - \langle \boldsymbol{x}, \boldsymbol{x}'\rangle| \leq O(\epsilon)$ with high probability.

Sparse random matrix. It is possible to improve on Theorem 1 in terms of computation. Observe that if $\boldsymbol{x}$ has k non-zero entries, then computing $\boldsymbol{A}\boldsymbol{x}$ requires $O(km)$ time. When $\boldsymbol{A}$ is a Gaussian random matrix, there is not really any structure that can be exploited to speed-up this computation.

It turns out it is possible to construct $\boldsymbol{A}$ more cleverly so that it has only $O(\sqrt{m \log d})$ non-zero entries per column. This way, computing $\boldsymbol{A}\boldsymbol{x}$ can be done in $O(k\sqrt{m \log d})$ time.

The construction is achieved using binary code vectors $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_d \in \{0,1\}^m$, each with exactly s non-zero entries, that collectively satisfy

$$\langle \boldsymbol{c}_i, \boldsymbol{c}_j \rangle \leq \frac{2s^2}{m}, \quad \text{for all } 1 \leq i < j \leq d.$$

It can be shown that such codes exist when $s = \Omega(\sqrt{m \log d})$. In fact, a construction of these codes can be achieved using a (good) hash function $h\colon [s] \times [d] \to [m/s]$ as follows. The j-th code vector $\boldsymbol{c}_j \in \{0,1\}^m$ is viewed as the concatenation of s vectors $\boldsymbol{c}_{1,j}, \boldsymbol{c}_{2,j}, \ldots, \boldsymbol{c}_{s,j} \in \{0,1\}^{m/s}$; the i-th vector $\boldsymbol{c}_{i,j}$ has a single non-zero entry in the $h(i,j)$-th position.

The linear mapping $\boldsymbol{A}$ is constructed as the following:

$$\boldsymbol{A} := \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \\ \vdots \\ \boldsymbol{A}_s \end{bmatrix} \in \{-1, 0, 1\}^{m \times d}$$

where, for each $i \in [s]$,

$$\boldsymbol{A}_i := \frac{1}{\sqrt{s}} \left[\; \sigma_{i,1}\boldsymbol{c}_{i,1} \;\middle|\; \sigma_{i,2}\boldsymbol{c}_{i,2} \;\middle|\; \cdots \;\middle|\; \sigma_{i,d}\boldsymbol{c}_{i,d} \; \right] \in \{-1, 0, 1\}^{(m/s) \times d},$$

and the $\{\sigma_{i,j}\}_{i\in[s],j\in[d]}$ are i.i.d. random variables with

$$\Pr(\sigma_{i,j}=1)=\Pr(\sigma_{i,j}=-1)=\frac{1}{2}.$$

(The $\{\sigma_{i,j}\}$ are often called *Rademacher random variables* or *random signs*.) For all practical purposes, the random signs can also be thought of as being specified by a hash function $\sigma\colon [d]\times[s]\to\{\pm1\}$, although the next theorem relies on random signs.

Theorem 2. *Let $\boldsymbol{A}\in\{-1,0,1\}^{m\times d}$ be the random matrix constructed from the code vectors and random signs as described above, and let s be the number of non-zero entries in each code vector. Fix any $\boldsymbol{v}\in\mathbb{R}^d$ and $\delta\in(0,1)$. With probability at least $1-\delta$,*

$$\left|\|\boldsymbol{A}\boldsymbol{v}\|^2-\|\boldsymbol{v}\|^2\right|\le\|\boldsymbol{v}\|^2\cdot O\left(\sqrt{\frac{\log(1/\delta)}{m}}+\frac{\log(1/\delta)}{s}\right).$$

Suppose $\|\boldsymbol{x}\|$ and $\|\boldsymbol{x}'\|$ are both $O(1)$. Theorem 2 implies that if $m=\Omega(1/\epsilon^2)$ and $s=\Omega(\sqrt{m\log d})$ (this latter condition is also used to guarantee the existence of the required code vectors), then $|\langle\boldsymbol{A}\boldsymbol{x},\boldsymbol{A}\boldsymbol{x}'\rangle-\langle\boldsymbol{x},\boldsymbol{x}'\rangle|\le O(\epsilon)$ with high probability.

Let $h\colon[s]\times[d]\to[m/s]$ and $\sigma\colon[s]\times[d]\to\{\pm1\}$ be the hash functions representing $\boldsymbol{A}$. Computing $\boldsymbol{A}\boldsymbol{x}$ is done as follows:

Let $\boldsymbol{y}:=\boldsymbol{0}\in\mathbb{R}^m$.
for $i=1,2,\ldots,s$ **do**
  for each non-zero entry x_j in $\boldsymbol{x}$ (with $j\in[d]$ and $x_j\in\mathbb{R}\setminus\{0\}$) **do**
    Let $y_{(i-1)s+h(i,j)}:=y_{(i-1)s+h(i,j)}+\sigma(i,j)x_j$
  end for
end for

If $\boldsymbol{x}$ has k non-zero entries, then the computation required is $O(ks)=O(k\sqrt{m\log d})$ (compared to $O(km)$ for the Gaussian random matrix). Computing inner products $\langle\boldsymbol{A}\boldsymbol{x},\boldsymbol{w}\rangle$ with a weight vector $\boldsymbol{w}\in\mathbb{R}^m$ is similar. It is also possible to show non-trivial guarantees for $s=1$ and also when the random signs σ are omitted.