# Feature hashing

In machine learning, **feature hashing**, also known as the **hashing trick** (by analogy to the kernel trick), is a fast and space-efficient way of vectorizing features, i.e. turning arbitrary features into indices in a vector or matrix.[1][2] It works by applying a hash function to the features and using their hash values as indices directly, rather than looking the indices up in an associative array. This trick is often attributed to Weinberger et al., but there exists a much earlier description of this method published by John Moody in 1989.[2][1]

## Contents



## Motivating example

In a typical document classification task, the input to the machine learning algorithm (both during learning and classification) is free text. From this, a bag of words (BOW) representation is constructed: the individual tokens are extracted and counted, and each distinct token in the training set defines a feature (independent variable) of each of the documents in both the training and test sets.

Machine learning algorithms, however, are typically defined in terms of numerical vectors. Therefore, the bags of words for a set of documents is regarded as a term-document matrix where each row is a single document, and each column is a single feature/word; the entry $i, j$ in such a matrix captures the frequency (or weight) of the $j$'th term of the *vocabulary* in document $i$. (An alternative convention swaps the rows and columns of the matrix, but this difference is immaterial.) Typically, these vectors are extremely sparse—according to Zipf's law.

The common approach is to construct, at learning time or prior to that, a *dictionary* representation of the vocabulary of the training set, and use that to map words to indices. Hash tables and tries are common candidates for dictionary implementation. E.g., the three documents

- *John likes to watch movies.*
- *Mary likes movies too.*
- *John also likes football.*

can be converted, using the dictionary

| Term | Index |
|---|---|
| John | 1 |
| likes | 2 |
| to | 3 |
| watch | 4 |
| movies | 5 |
| Mary | 6 |
| too | 7 |
| also | 8 |
| football | 9 |

to the term-document matrix

$$\begin{pmatrix} \text{John} & \text{likes} & \text{to} & \text{watch} & \text{movies} & \text{Mary} & \text{too} & \text{also} & \text{football} \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

(Punctuation was removed, as is usual in document classification and clustering.)

The problem with this process is that such dictionaries take up a large amount of storage space and grow in size as the training set grows.[3] On the contrary, if the vocabulary is kept fixed and not increased with a growing training set, an adversary may try to invent new words or misspellings that are not in the stored vocabulary so as to circumvent a machine learned filter. This difficulty is why feature hashing has been tried for spam filtering at Yahoo! Research.[4]

Note that the hashing trick isn't limited to text classification and similar tasks at the document level, but can be applied to any problem that involves large (perhaps unbounded) numbers of features.

# Feature vectorization using hashing trick

Instead of maintaining a dictionary, a feature vectorizer that uses the hashing trick can build a vector of a pre-defined length by applying a hash function $h$ to the features (e.g., words), then using the hash values directly as feature indices and updating the resulting vector at those indices. Here, we assume that feature actually means feature vector.

```
function hashing_vectorizer(features : array of string, N : integer):
    x := new vector[N]
    for f in features:
        h := hash(f)
        x[h mod N] += 1
    return x
```

Thus, if our feature vector is ["cat","dog","cat"] and hash function is $hash(x_f) = 1$ if $x_f$ is "cat" and $2$ if $x_f$ is "dog". Let us take the output feature vector dimension (N) to be 4. Then output x will be [0,2,1,0]. It has been suggested that a second, single-bit output hash function $\xi$ be used to determine the sign of the update value, to counter the effect of hash collisions.[2] If such a hash function is used, the algorithm becomes

```
function hashing_vectorizer(features : array of string, N : integer):
    x := new vector[N]
    for f in features:
        h := hash(f)
        idx := h mod N
        if ξ(f) == 1:
            x[idx] += 1
        else:
            x[idx] -= 1
    return x
```

The above pseudocode actually converts each sample into a vector. An optimized version would instead only generate a stream of $(h,\xi)$ pairs and let the learning and prediction algorithms consume such streams; a linear model can then be implemented as a single hash table representing the coefficient vector.

## Properties

When a second hash function $\xi$ is used to determine the sign of a feature's value, the expected mean of each column in the output array becomes zero because $\xi$ causes some collisions to cancel out.[2] E.g., suppose an input contains two symbolic features $f_1$ and $f_2$ that collide with each other, but not with any other features in the same input; then there are four possibilities which, if we make no assumptions about $\xi$, have equal probability, as listed in the table on the right.

| $\xi(f_1)$ | $\xi(f_2)$ | Final value, $\xi(f_1) + \xi(f_2)$ |
|---|---|---|
| -1 | -1 | -2 |
| -1 | 1 | 0 |
| 1 | -1 | 0 |
| 1 | 1 | 2 |

In this example, there is a 50% probability that the hash collision cancels out. Multiple hash functions can be used to further reduce the risk of collisions.[5]

Furthermore, if $\varphi$ is the transformation implemented by a hashing trick with a sign hash $\xi$ (i.e. $\varphi(x)$ is the feature vector produced for a sample $x$), then inner products in the hashed space are unbiased:

$$\mathbb{E}[\langle \varphi(x), \varphi(x') \rangle] = \langle x, x' \rangle$$

where the expectation is taken over the hashing function $\varphi$.[2] It can be verified that $\langle \varphi(x), \varphi(x') \rangle$ is a positive semi-definite kernel.[2][6]

## Extensions and variations

Recent work extends the hashing trick to supervised mappings from words to indices,[7] which are explicitly learned to avoid collisions of important terms.

## Applications and practical performance

Ganchev and Dredze showed that in text classification applications with random hash functions and several tens of thousands of columns in the output vectors, feature hashing need not have an adverse effect on classification performance, even without the signed hash function.[3] Weinberger et al. applied their variant of hashing to the problem of spam filtering, formulating this as a multi-task learning problem where the input features are pairs (user, feature) so that a single parameter vector captured per-user spam filters as well as a global filter for several hundred thousand users, and found that the accuracy of the filter went up.[2]

# Implementations

Implementations of the hashing trick are present in:

- Apache Mahout[5]
- Gensim[8]
- scikit-learn[9]
- sofia-ml[10]
- Vowpal Wabbit
- Apache Spark[11]
- R[12]
- TensorFlow[13]

## See also

- Bloom filter
- Count–min sketch
- Heaps' law
- Locality-sensitive hashing
- MinHash

## References


1. Moody, John (1989). "Fast learning in multi-resolution hierarchies" (http://papers.nips.cc/paper/175-fast-learning-in-multi-resolution-hierarchies.pdf) (PDF). *Advances in Neural Information Processing Systems*.
2. Kilian Weinberger; Anirban Dasgupta; John Langford; Alex Smola; Josh Attenberg (2009). *Feature Hashing for Large Scale Multitask Learning* (http://alex.smola.org/papers/2009/Weinbergeretal09.pdf) (PDF). Proc. ICML.
3. K. Ganchev; M. Dredze (2008). *Small statistical models by random feature mixing* (http://www.cs.jhu.edu/~mdredze/publications/mobile_nlp_feature_mixing.pdf) (PDF). Proc. ACL08 HLT Workshop on Mobile Language Processing.
4. Josh Attenberg; Kilian Weinberger; Alex Smola; Anirban Dasgupta; Martin Zinkevich (2009). "Collaborative spam filtering with the hashing trick" (https://www.virusbulletin.com/virusbulletin/2009/11/collaborative-spam-filtering-hashing-trick). *Virus Bulletin*.
5. Owen, Sean; Anil, Robin; Dunning, Ted; Friedman, Ellen (2012). *Mahout in Action*. Manning. pp. 261–265.
6. Shi, Q.; Petterson J.; Dror G.; Langford J.; Smola A.; Strehl A.; Vishwanathan V. (2009). *Hash Kernels*. AISTATS.
7. Bai, B.; Weston J.; Grangier D.; Collobert R.; Sadamasa K.; Qi Y.; Chapelle O.; Weinberger K. (2009). *Supervised semantic indexing* (http://www.cs.cornell.edu/~kilian/papers/ssi-cikm.pdf) (PDF). CIKM. pp. 187–196.
8. "gensim: corpora.hashdictionary – Construct word<->id mappings" (http://radimrehurek.com/gensim/corpora/hashdictionary.html). Radimrehurek.com. Retrieved 2014-02-13.
9. "4.1. Feature extraction — scikit-learn 0.14 documentation" (http://scikit-learn.org/stable/modules/feature_extraction.html#feature-hashing). Scikit-learn.org. Retrieved 2014-02-13.
10. "sofia-ml - Suite of Fast Incremental Algorithms for Machine Learning. Includes methods for learning classification and ranking models, using Pegasos SVM, SGD-SVM, ROMMA, Passive-Aggressive Perceptron, Perceptron with Margins, and Logistic Regression" (https://code.google.com/p/sofia-ml/). Retrieved 2014-02-13.

11. "Hashing TF" (https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.feature.HashingTF). Retrieved 4 September 2015. "Maps a sequence of terms to their term frequencies using the hashing trick."
12. "FeatureHashing: Creates a Model Matrix via Feature Hashing With a Formula Interface" (https://cran.r-project.org/web/packages/FeatureHashing/index.html).
13. "tf.keras.preprocessing.text.hashing_trick — TensorFlow Core v2.0.1" (https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/hashing_trick). Retrieved 2020-04-29. "Converts a text to a sequence of indexes in a fixed-size hashing space."

# External links

- Hashing Representations for Machine Learning (http://hunch.net/~jl/projects/hash_reps/index.html) on John Langford's website
- What is the "hashing trick"? - MetaOptimize Q+A (https://web.archive.org/web/20120609232923/http://metaoptimize.com/qa/questions/6943/what-is-the-hashing-trick)