Generative adversarial networks for increasing the veracity of big data

Conference Paper · December 2017

DOI: 10.1109/BigData.2017.8258219

Conrad Tucker
Pennsylvania State University
96 PUBLICATIONS **493** CITATIONS

Generative Adversarial Networks for Increasing the Veracity of Big Data

Matthew L. Dering
Computer Science and Engineering
Penn State University
University Park, Pennsylvania 16802
Email: mld284@cse.psu.edu

Conrad S. Tucker
Engineering Design and Industrial Engineering
Penn State University
University Park, Pennsylvania 16802
Email: ctucker4@psu.edu

***Abstract*—This work describes how automated data generation integrates in a big data pipeline. A lack of veracity in big data can cause models that are inaccurate, or biased by trends in the training data. This can lead to issues as a pipeline matures that are difficult to overcome. This work describes the use of a Generative Adversarial Network to generate sketch data, such as those that might be used in a human verification task. These generated sketches are verified as recognizable using a crowd-sourcing methodology, and finds that the generated sketches were correctly recognized 43.8% of the time, in contrast to human drawn sketches which were 87.7% accurate. This method is scalable and can be used to generate realistic data in many domains and bootstrap a dataset used for training a model prior to deployment.**

***Keywords*-Generative Models, Big Data, Deep Learning, GANs, Sketches**

I. Introduction

The utility of automatic, realistic, synthetic data generation in big data problems has been demonstrated in both Velocity [1] and Variety [2]. Generative models are also frequently employed in big data settings for capturing trends within data [3], [4]. A generative model is one which models data as a distribution, or combination of distributions, which can then be sampled. In some cases, this ability to be sampled is a byproduct of the technique used [5]. In other cases, sampling this distribution is the goal [6], as obtaining new or unique data is critical to many applications.

Typically big data problems apply a filter or transform to reduce the size of their data, and gain insights, or other benefits. While finding the signal in the noise has many benefits, this work considers the benefits of generating realistic data with known characteristics prior to implementing a big data system. This method will have benefits in bootstrapping as well. For instance, a company that wishes to put in place a robust “CAPTCHA” style human verification system might use such a model to generate a large number of images. These generated images could be used to either fool non-human actors, or could conceivably be produced by a human actor and thus used for training. This generated data would be based on a much smaller amount of data, and thus could be more secure from the beginning, and less prone to bias in the training set.

For this reason, the authors of this work describe how generating large amounts of human quality data could be used for training a big data platform before deployment. These ideas have been incorporated in the past for building more diverse models [7], but it is becoming increasingly important as model sizes grow. Adversarial training of models has shown to be helpful [8], [9], [10] in providing a model that is more robust to error. Big data settings have a vested interest in handling edge cases in predictable ways. By incorporating generated data, these edge cases can be predicted ahead of time. This approach is valuable in assuring the *veracity* of data, which has long proved to be a substantial challenge when drawing inferences from high dimensional data such as text [11] or images [12].

The main contributions of this work are as follows:

- A Generative Adversarial Network (GAN) is proposed which can quickly generate human-readable visual images objects.
- These images have use in both training of a big data model, pre-deployment, and in big data decision making, post-deployment.

II. Related Work

A. Generated Big Data

Generated data has reached the field of big data in unusual ways. Some work has examined how to integrate generating realistic data into big data pipelines, so that the pipelines can be tested for load, accuracy, or other metrics without using random data [1]. Recent work has explored generated data as an attack vector [13]. Like other attack vectors, an understanding of the problem can lead to increased security [14], [15]. Even in big data situations where security is not critical, these insecure outcomes are due to systems which have not been properly trained.

It is important to contrast this work with other synthetic data techniques as well. Randomly generated data may be appropriate in some settings, but this data can quickly become unfeasible or unreasonable once the dimensionality of the data grows (for instance, text or image data). Synthetic data generation techniques focus on solving class imbalance problems, with approaches such as SMOTE [16]. More recent techniques are driven by sampling and synthesizing from the training data as well [17], [18]. These ideas are helpful when dealing with lower dimensional data or a class imbalance problem, but suffer from the same problems as random data when considering high dimensional data.

This work proposes to use generated data in a big data pipeline during training, so that the models are resistant to inaccuracies due to new data. This work proposes that by generating new novel data like that which the model may be exposed to in the real world, while still understanding what training outcomes are desired for this generated data, the resulting model is more robust to varying levels of data veracity.

B. Generative Models

There are many types of generative models. As stated above, generative models are simply models which can be sampled and used to generate new data. For instance, Latent Dirichlet Allocation [19], a tool popular for topic modeling in text which is commonly used in social networks [20], [21]. Sampling from these can lead to a representation of topic words within a text corpus. Gaussian Mixture Models are a type of model which assumes that the data is a linear combination of several Gaussian distributions in space. Sampling from these distributions is often used as a method of clustering, where the center of the distribution is treated as the centroid of the cluster. These methods are very common for modeling and understanding numerical data [22].

The advent of deep learning, a type of machine learning that involves stacked non-linear combinations of neurons, has brought about new types of generative models. While not strictly generative, these types of model often contain millions of parameters, and convincing results have been obtained by simply optimizing the input to generate a desired output [23]. More explicitly, a type of network called a Recurrent Neural Network can be used to generate text [24], caption images [25], generate images [26], and even generate images from captions [27], [28]. However, because of their high quality outputs, the most popular generative models are Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs). Both model types work by discovering a lower dimensional representation of a high dimensional data type (for instance, images). A VAE works by encoding an input into a low dimensional vector of zero mean and unit variance. This vector is then decoded back into the original image, and by generating random vectors of zero mean and unit variance, new data can be produced. On the other hand, a GAN challenges a network (the discriminator) to determine if a given input is generated by a neural network (the generator), or drawn from the dataset. As with the VAE, the input to this generator is also a random vector with zero mean and unit variance. The work presented here employs a GAN which is augmented by a class prior as well as a random prior.

As shown above, there are many types of generative models, suited to many purposes. This work considers how automated image generation would provide value in training big data pipelines, as well as bootstrapping small datasets.

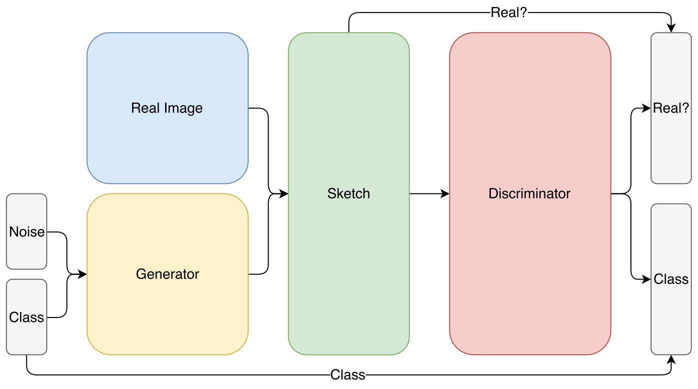


Fig. 1: An outline of the GAN in this work

III. Method

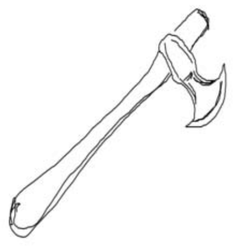
This section is organized as follows: Section III-A outlines the data used by this work. Section III-B describes the class of network used. Section III-C outlines the training process and Section III-D puts forth how the results are evaluated.

A. Data description

The data considered in this work is a set of labeled sketches of various objects. Image data consists of a dense matrix of pixel values, and serves as partial input data for the Discriminator network. For example, an image of $h \times w$ pixels and n color channels is represented as a $h \times w \times n$ matrix. Since this method concerns black and white sketches, in this case $n = 1$, though this method could generalize to RGB data, in which case $n = 3$ (one channel each for Red Green and Blue). A sample sketch from the dataset can be seen in Fig. 2a, and how a small portion of a sketch can be viewed as pixels can be seen in Fig. 2b. Since sketches are a simplified image modality, the values can be represented by a binary value so that the value at a given coordinate is 0 if that pixel is colored in, and 1 otherwise. This methodology assumes sketches are of a fixed size.

Hand drawn sketch data differs from color image modality in several ways. First, they are sparse in content. In contrast to a photograph, where even the content other than the subject (i.e. the background, or context), is meaningful and important. Sketches are changes (that is, marks) made by a designer given a blank space. Second, there is an expectation of continuity of these marks. Generally, sketches are made using long continuous strokes, rather than short small ones, and very rarely contain detailing such as dots. Finally, the choice of adding detail to sketches is not made lightly by the artist, and generally is only made when semantically meaningful. For instance, consider the object *tennis racket*. Without the laces running across the frame, it would be difficult to determine the difference between a racket and a spoon.

Each sketch also has a class label (such as *cup*, or *airplane*), represented by a binary vector c in length, where c is the number of classes in the dataset. This vector is all zeroes except in the position corresponding the given class.


(a) A sketch of an axe from the dataset.

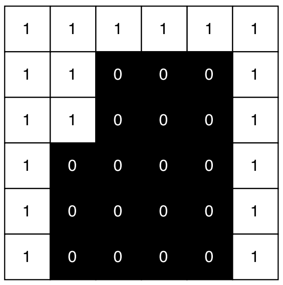

(b) A $6 \times 6 \times 1$ subset of the right top edge of the ax blade.

Fig. 2: Image modalities considered in this method.

B. Generative Adversarial Network

A GAN has two main components: The generator G and the discriminator D. G is a function which maps a class C, and a random latent variable Z to a sketch S ($G : \mathbb{R}^C \times \mathbb{R}^Z \to \mathbb{R}^S$). Z is intended as a random prior, sampled uniformly from a unit norm space across z dimensions. While other distributions may be used, the purpose of this is to learn to map a uniform low dimensional vector to a range of images, so using non-uniform distributions would result in lower diversity of output. Similarly, D is a function which maps a sketch S to a class C and a binary variable, which is 1 if the sketch and photos match, and 0 otherwise ($D : \mathbb{R}^S \to \{0, 1\} \times \mathbb{R}^C$).

Neural networks are layers of simple mathematical units called *neurons*. A neuron has three main components: weights, biases, and activation function. Typically weights and biases are learned, while the activation function is fixed. The operation performed by these neurons can be seen in Equation 1:

$$y = f(w \times x + b) \tag{1}$$

where y is the output of a neuron, f is the activation function, w are the weights, x is the input, and b is the bias. Each network layer is distinguished by type as well, which primarily changes what is considered input to the neurons in a given layer. For instance, for a fully connected layer, the input x will be every neuron of the previous layer. As such, these layers have high memory demands, since each of these inputs requires a learned weight. In the instance of a convolution layer, the input is a small locally adjacent region of the previous layer close to the location of the neuron in the new layer.

The primary activation function used in this network is the leaky Rectified Linear Unit (LeakyReLU), which is given by equation 2:

$$f(x) = \begin{cases} x & x > 0 \\ \alpha \times x & x \leq 0 \end{cases} \tag{2}$$

where x is the output of the neuron prior to activation, and α is a set parameter, generally less than 1 (many works suggest values of 0.1 or 0.2). This operation adds non-linearity to the network, by allowing positive gradients to pass through to the following layer, while squashing negative gradients without discarding them completely (as in the case of ReLU, where $\alpha = 0$).

1) Generative Network: Generative networks have a diverse set of designs, but briefly each will learn to generate output based on some prior input. Some instances use random noise as their only prior, others may use metadata such as object class or textual description. The network proposed in this work uses as a prior a random vector which represents the features of a sketch of the object, and a class vector.

It is important to note the exact nature of the task performed by this network. This network must learn to draw convincing sketches by searching the input space, given only random noise and a class vector, using only a binary variable and a class prediction as feedback. By convincing, this means it must generate output which is confused by a trained network with samples drawn from the training set. The structure of the network, similar to the reference network proposed by Odena et al˙ [29] can be seen in Fig. 3.

Because of the nature of sketches discussed in Section III-A, it is important that this generator network be able to accommodate these concerns. Possible issues include overfitting due to an overly large network (especially problematic in the case of the discriminator network). Since these networks are very large, and the target modality is simple in nature (the dataset in the case study reports a median number of strokes of 14), the risk of overfitting and failing to generate realistic output is significant.

Another one of the key aspects of this work is the use of *embeddings* for class identification. Embeddings are maps of data in high dimensional space, often used for text information. Using these embeddings, classes can be represented not as one-hot vectors (i.e. this is a cup), but by some learned vector in space. For example, in a two dimensional space, a hammer may be located at $(-1, 1)$, while a cup may be at $(1, -1)$. Since these locations are learned by the network, these locations have meaning. For example, an axe is similar to a hammer (e.g. both have handles and heads), so it may be closer to the hammer than the cup in space. This has a few advantages. For one, this allows these vector locations to be changed to explore the design space *in between* classes (i.e. 90% cup, 10% hammer, 80% cup, 20% hammer and so on). Next, these embeddings can be fuzzed, or manipulated slightly, as a form of data augmentation, such as in [28].

2) Discriminator Network: The other half of a GAN is the discriminator. In the game-like arrangement that constitutes a GAN, the discriminator's role is to learn the difference between generated input and real input. This network will also learn to classify input into a class label (e.g. cup, car).

This network consists of three parts. The first part is a feature extractor of the image, which decomposes the image into a vector of fixed length based on its contents. Using this vector as input, the second part makes a decision about if the generated input is generated or a member of the dataset. The final portion uses this same vector to make a classification of the input into an object type. More detail can be seen in Fig. 4.

The goal of the discriminator is not to perfectly learn the

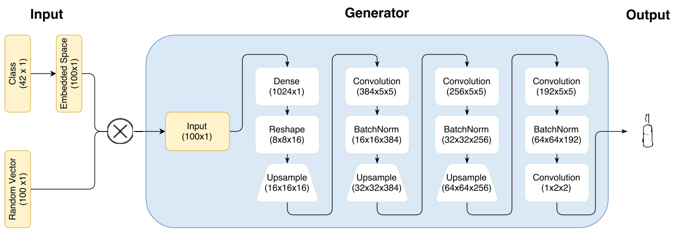


Fig. 3: Generator Network architecture. After each batch Normalization there is a ReLU. The sizes shown below each unit represent output size, except for the convolutions which represent kernel size and channel count.

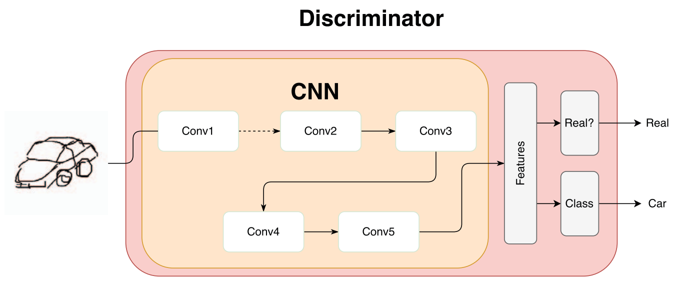


Fig. 4: Discriminator architecture. Orange denotes the CNN. White boxes in the CNN denote Convolutional Layers. Solid lines denote LeakyReLU with Batch Normalization. Dotted lines denote LeakyReLU, no Batch Normalization. Grey boxes denote vectors.

difference between data from the dataset and generated data. Instead, its primary function is to provide a differentiable loss which can be used to tune the weights of the generator. While having a highly accurate or fully featured discriminator may prove beneficial, the current state of the art emphasizes that they avoid overfitting instead. As discussed in Section III-B1, the sketch data modality presents unique challenges. The output of the discriminator has no practical benefit to this method, apart from defining a loss space which the generator explores via the generated input. The primary concern is finding a balance in network size between large networks and small networks. If the network is too large, it may overfit and learn the entirety of the dataset. This prevents the generator from ever "winning", which will slow or even prevent the generator from training. Conversely, if the network is too small, the generator may be able to fool the discriminator too easily, and hinder its ability to create convincing output.

C. Training And Evaluation

The training process for an Auxiliary Classifier GAN (AC-GAN) involves training the networks both separately and together. An overview of this can be seen in Algorithm 1. For each batch of input, the generator will generate a random sketch of a random class. This, along with input drawn from the dataset, is used to train the discriminator, which calculates a probability of being drawn from the dataset, as well as a class prediction.

Algorithm 1: Training Process

```
  Input: Sketches $\mathsf{S}$ and associated classes $\mathsf{C}$
1 for batch $\mathsf{b} \in \mathsf{S}$ do
2     Train Generator;
3     Generate fake data with generator;
4     Train discriminator on real and generated data;
5 end
```

One active area of research for these types of networks is addressing their lack of convergence. Intuitively, as the discriminator becomes less likely to be fooled, the generator must become more adept at fooling it. From this arrangement, it is unclear at what point a model can be considered converged and training should cease. In practice, these models oscillate, with the generator occasionally outperforming the discriminator, and the discriminator occasionally outperforming the generator. This is a drawback of a GAN based approach instead of a VAE or FVBN. In this work, the network is trained until the discriminator is able to accurately recognize object classes, as this is a metric which both the generator and discriminator wish to minimize. In determining the convergence point, the classification loss is also considered, as this is a goal of both networks and is not adversarial in nature.

Neural networks are prone to learning very high activations and probabilities, especially in adversarial arrangements [30]. Hence, this network is trained using *One Sided Smoothing*, where the target prediction for real examples is a value less than 1, rather than 1. In this case, the value 0.9 was chosen, as suggested in [30].

D. Evaluation

This network is evaluated by human survey participants who are asked to categorize random objects shown to them drawn by the generator network. Additionally, the ability of this generator to generate new concepts will be proven by generating imputed classes.

The primary evaluation of the generator's ability to generate convincing sketches is made using a survey. This survey was given to participants on Amazon Mechanical Turk, who were compensated monetarily. They are asked to pick from 5 random classes (one of which is correct) as well as a "Do not know/none of the above" option. Importantly, the images selected for this survey must satisfy two conditions: they must be correctly classified by the discriminator, and the discriminator must incorrectly believe that they are genuine images. Since the network uses two priors, a class and random noise, the random noise space is imputed while class is held constant. This is repeated for each class, with the same 20 randomly sampled noise priors for each of the classes.

IV. Case Study

A. Data

A case study is presented in order to evaluate the effectiveness of this method. Sketches are drawn from the

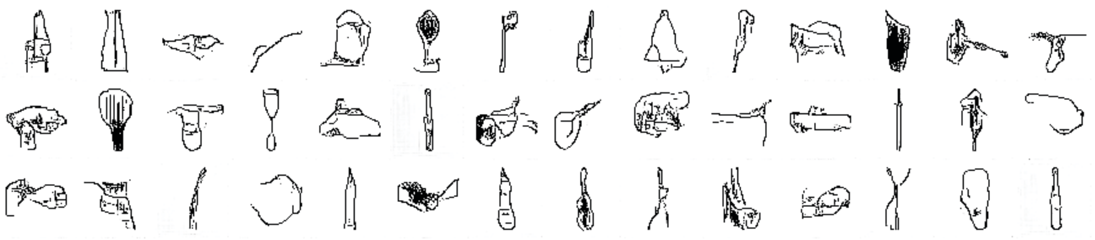


Fig. 5: Random generated samples across all 42 classes.

chair, airplane, alarm clock, axe, bell, bench, bicycle, blimp, cannon, car (sedan), motorcycle, piano, pickup truck,pistol, racket, rocket, sailboat, rifle, scissors, couch, cup, door, eyeglasses, fan, guitar, hammer, harp, helicopter, hot-air balloon, hourglass, knife, shoe, spoon, sword, table, tank, teapot, trumpet, violin, wheelchair, windmill, wine bottle

TABLE I: Classes generated in this case study.

Airplane, Alarm Clock, Bell, Bench, Bicycle, Blimp, Cannon, Car, Chair, Couch, Cup, Door, Eyeglasses, Fan, Guitar, Hammer, Harp, Helicopter, Hot-Air Balloon, Hourglass, Piano, Pickup Truck, Pistol, Racket, Spoon, Table, Tank, Teapot, Trumpet, Violin, Windmill, Wine Bottle

TABLE II: Classes which could fool the discriminator.

Sketchy Database [31]. This dataset contains 75,471 sketches across 125 categories based on 12,500 photos. 10922 of these sketches were removed for reasons cited by the dataset including ambiguity, erroneously including context such as background, incorrect object pose, or artist error.

Additionally, naturally occurring objects were removed from this dataset, leaving 42 classes, which can be seen in Table I. In total, 19,633 sketches were used to train this GAN.

B. Network Description And Training

The ACGAN used in this method is a straightforward network similar to Odena et al. [29], with a few differences. The major difference is to accommodate the smaller generated sketch size. The sketches are originally 256×256 and are scaled down to 64×64 to speed up training.

The model was trained for 100 epochs, with a batch size of 64, using an Adam optimizer with the hyperparameters suggested in the reference work [29].

C. Evaluation

This network is evaluated by a survey. The survey was conducted on Amazon Mechanical Turk, using an open call and offering a monetary reward. For each network, up to 20 images for each class are generated randomly, which are correctly classified but are misclassified by the discriminator as being real (with a probability of > 0.7).

Next, 500 random sets of 5 images are drawn from this sample. For each image, 3 incorrect classes, the correct class, and an option for “Do Not Know/None of the above”, are available, and a user must answer. Respondents who “clicked through” and took less than 5 seconds per image are not considered, as the quality of their response cannot be ensured.

V. Results

A. Qualitative Evaluation

Of the 42 classes trained, results were collected for 32 classes. This is primarily due to the difficulty of generating

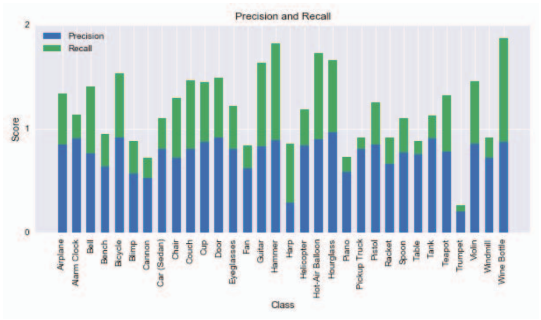


Fig. 6: Precision and Recall of the qualitative assessment

examples which could fool the discriminator. The discriminator outputs a probability of being sampled from the dataset (between 0 and 1), and only samples which had a probability of > 0.7 were considered. Those classes are listed in Table II For the large network, 281 responses were left after filtering out responses which were too fast, and therefore unreliable.

The precision and recall for the qualitative measurement can be seen in Fig. 6. Overall, the predictions were 43.8% accurate. Note that several classes were highly accurate, including Wine Bottle (100%), Hammer (92.8%), Hot Air Balloon (83.6%), Guitar (80.7%), and Hourglass (69.6%). One of the poorest performers, Harp, only had 7 cases included in the evaluation phase (the second least common class was Couch), but is included here for reference. Overall Pickup Truck and Trumpet had the lowest F Score.

Of interest in this study is the ability of these networks to generate recognizable objects of both high and low detail. Among the highest detail objects in the dataset is racket. As discussed, rackets are distinguished by laces in a frame, and rackets without laces are removed from the dataset as ambiguous. The precision of the racket class was among the best.

To better contextualize these results, another survey was conducted. Drawing random images from the training set, users on Mechanical Turk were asked about five random images drawn from the dataset, and given the opportunity

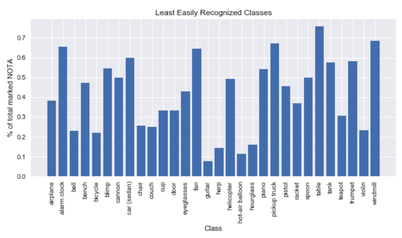


Fig. 7: Classes marked none of the above or do not know

to reply none of the above. From the 500 respondents, 252 responses met the time threshold. Overall, the users were 87.7% accurate at categorizing these sketches. The primary source of these inaccuracies came from 3 classes: Rifle, Sailboat and Rocket. Each of these classes registered a none of the above response over 80% of the time.

B. NOTA Analysis

The users in this survey were given the option of selecting none of the above (NOTA) or do not know as one of the choices. In addition to noting which classes were identified correctly, it is also worth examining which classes were generated that users could not select, even given that the correct choice was presented to them. In other words, these classes were generated poorly enough that the user was unable to recognize the sketch, even when primed with the correct choice.

The results of this can be seen in Fig. 7. The classes which was most easily recognized is guitar, followed by hot air balloon and hourglass. Violin, bicycle, and bell also rarely generated enough confusion for the user to pick NOTA. Two classes were never categorized as NOTA, hammer and wine bottle. Hammer is a distinctive shape, with a clear handle and head, so it is encouraging that the GAN was able to generate clear examples of these. On the other hand, wine bottle is a very simple shape, with most examples having minimal detailing. However, these classes are also not very diverse, both in shape and viewpoint.

In contrast, over 70% of tables were unable to be recognized. This may be due to having a high diversity of viewpoint, making it difficult for the network to learn a consistent model. Tables are also relatively low in detail, and some sketches include extraneous detail such as coloring in the surface. A similar issue seems to have yielded poor performance from the pickup truck class, and car class, as there is very little consistency of viewpoint across these classes.

Another possible source of error is sketch complexity, as measured by mean number of strokes in the training set. Comparing the f-score of the generated sketches with the average number of strokes in the training set, a general downward trend is demonstrated. This can be seen in Fig. 8.

C. Design Exploration

This section demonstrates the proposed network's effectiveness in exploring designs across classes. This is accomplished

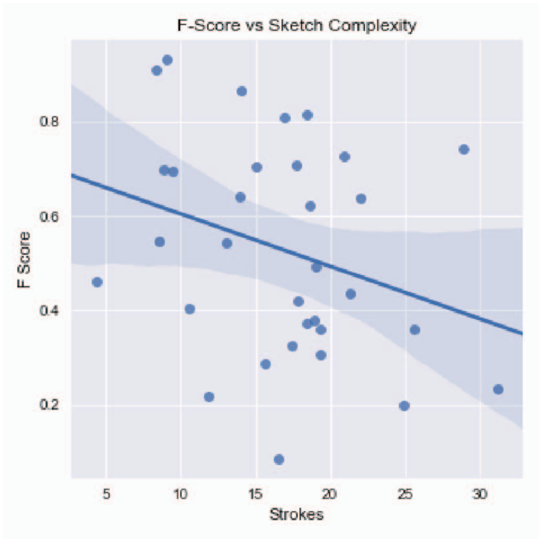


Fig. 8: Class recognition performance vs average sketch complexity

by generating 20 numbers, spaced evenly through the space, and using those as input across each class. The diversity of these designs can be seen in Figure 9. Note that the center row, for which the random number is a 100 dimensional vector of 0s, the output is all the same. This reflects the bias of the network. Since the latent space is 100 dimensions, sampling just 20 points represents a small fraction of the space. A more in depth study of this space may provide a much deeper insight.

A similar method can be used to demonstrate the ability of the objects to be morphed between each other. Figure 10 shows the output of the generator given a constant noise vector, but imputing the space between two classes. Since this network uses embedding, each class is represented by a vector, enabling arithmetic such as addition, subtraction, and averaging. For clarity, 6 pairs of classes are considered, whose class vector values are derived by taking a weighted average of two classes, though this can easily be expanded to three or more classes.

D. Data Quality

The use of GANs converges with data quality issues in several ways. First, it provides a helpful macroscopic method of detecting outliers. Since the discriminator is created and trained to detect if the input is derived from the dataset or is artificial, any erroneous classification of real data as artificial could be considered an outlier. In some cases, this may even guard against possible intrusion or injection of false records by an adversarial actor. In the motivating example of a CAPTCHA style image verification system, these outliers could give historical notifications to previous intrusions by automated actors. That is to say, any misclassification in the dataset due to outliers could potentially trigger an integrity check, and could indicate that the existing human verification system has been compromised.

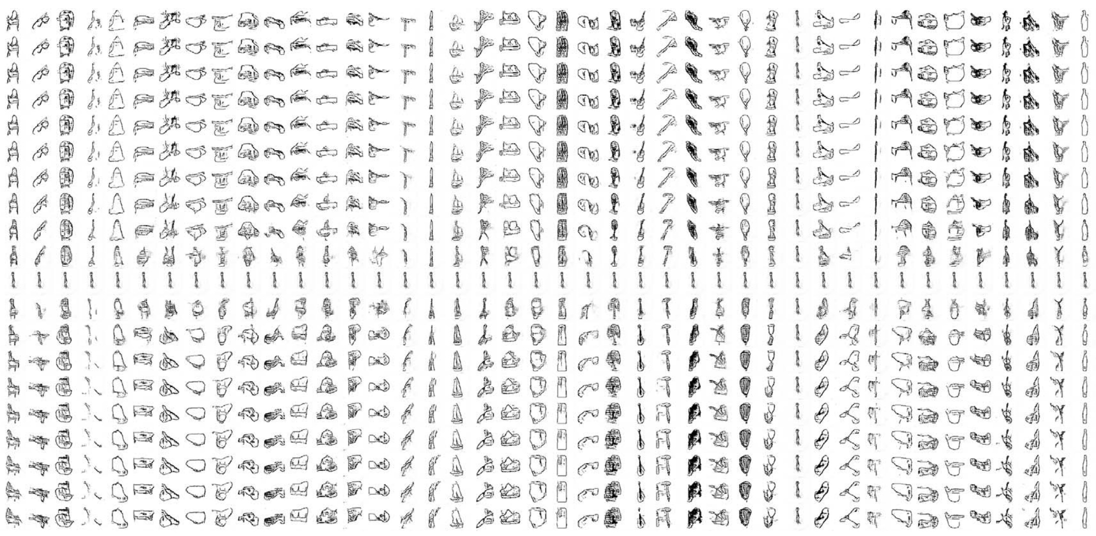


Fig. 9: 20 generated samples for each of the 42 classes

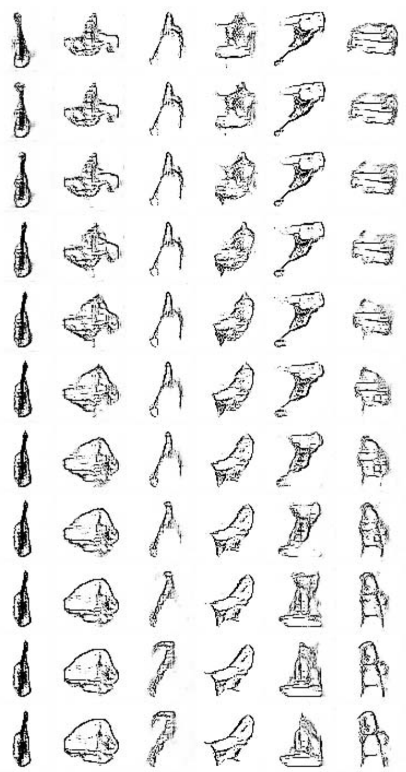


Fig. 10: Exploring the space between classes. Class Pairs (in columns) from left to right, top to bottom: Violin – Guitar; Car – Truck; Hammer - Ax; Airplane - Helicopter; Rifle – Pistol; Chair – Bench. The ratios change by 10% for each row (i.e. 100/0, 90/10 . . . 10/90, 0/100).

VI. Conclusions

This work has outlined how a GAN would integrate in a big data pipeline, and improve the veracity of data. This improvement takes place both prior to deployment during training and after deployment as a filter. This would also allow the adverse impact of low veracity data to be mitigated, since the model would be robust to many more types of data. The utility of this GAN was demonstrated by training it with sketched images of 42 classes of objects. The images generated by the GAN were shown to human observers who were 43.8% accurate in classifying the images, while 87.7% accurate in classifying human generated images. These generated images could be used to augment an existing dataset by providing more input for training, or to guard a system against high quality computer generated images. By ensuring that low veracity data can be handled gracefully and integrated or protected against, a GAN such as this one can be a valuable part of many big data systems.

Acknowledgments

The authors would like to acknowledge the NSF DUE/IUSE #1449650: Investigating the Impact of Co-Learning Systems in Providing Customized, Real-time Student Feedback and DARPA FUN DESIGN #HR00111820008.

References

[1] R. J. Nowling and J. Vyas, “A domain-driven, generative data model for big pet store,” in *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*. IEEE, 2014, pp. 49–55.

[2] C. Beecks, M. S. Uysal, and T. Seidl, “Gradient-based signatures for big multimedia data,” in *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2834–2835.

[3] X. Li, M. Cheung, and J. She, “Connection discovery using shared images by gaussian relational topic model,” in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 931–936.

[4] X. Jia, A. Khandelwal, J. Gerber, K. Carlson, P. West, and V. Kumar, “Learning large-scale plantation mapping from imperfect annotators,” in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1192–1201.

[5] T. Mukherjee, B. Parajuli, P. Kumar, and E. Pasiliao, “Truthcore: Non-parametric estimation of truth from a collection of authoritative sources,” in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 976–983.

[6] X. Jia, A. Wang, X. Li, G. Xun, W. Xu, and A. Zhang, “Multi-modal learning for video recommendation based on mobile application usage,” in *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE, 2015, pp. 837–842.

[7] P. Melville and R. J. Mooney, “Creating diversity in ensembles using artificial data,” *Information Fusion*, vol. 6, no. 1, pp. 99–111, 2005.

[8] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, “Domain-adversarial training of neural networks,” *Journal of Machine Learning Research*, vol. 17, no. 59, pp. 1–35, 2016.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.

[10] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, “Adversarial autoencoders,” *arXiv preprint arXiv:1511.05644*, 2015.

[11] T. Bodnar, C. Tucker, K. Hopkinson, and S. G. Bilén, “Increasing the veracity of event detection on social media networks through user trust modeling,” in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 636–643.

[12] M. A. Schuh and R. A. Angryk, “Massive labeled solar image data benchmarks for automated feature recognition,” in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 53–60.

[13] M. Wojnowicz, B. Cruz, X. Zhao, B. Wallace, M. Wolff, J. Luan, and C. Crable, “influence sketching: Finding influential samples in large-scale regressions,” in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 3601–3612.

[14] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against deep learning systems using adversarial examples,” *arXiv preprint arXiv:1602.02697*, 2016.

[15] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.

[16] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[17] A. Pourhabib, B. K. Mallick, and Y. Ding, “Absent data generating classifier for imbalanced class sizes,” *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 2695–2724, 2015.

[18] H. He, Y. Bai, E. A. Garcia, and S. Li, “Adasyn: Adaptive synthetic sampling approach for imbalanced learning,” in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 1322–1328.

[19] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[20] D. Fried, M. Surdeanu, S. Kobourov, M. Hingle, and D. Bell, “Analyzing the language of food on social media,” in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 778–783.

[21] T. Bodnar, M. L. Dering, C. Tucker, and K. M. Hopkinson, “Using large-scale social media networks as a scalable sensing system for modeling real-time energy utilization patterns,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2016.

[22] H.-Â. Cao, T. K. Wijaya, K. Aberer, and N. Nunes, “Estimating human interactions with electrical appliances for activity-based energy savings recommendations,” in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1301–1308.

[23] L. A. Gatys, A. S. Ecker, and M. Bethge, “Image style transfer using convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2414–2423.

[24] I. Sutskever, J. Martens, and G. E. Hinton, “Generating text with recurrent neural networks,” in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.

[25] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and tell: A neural image caption generator,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[26] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” *arXiv preprint arXiv:1601.06759*, 2016.

[27] E. Mansimov, E. Parisotto, J. L. Ba, and R. Salakhutdinov, “Generating images from captions with attention,” *arXiv preprint arXiv:1511.02793*, 2015.

[28] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative adversarial text to image synthesis,” *arXiv preprint arXiv:1605.05396*, 2016.

[29] A. Odena, C. Olah, and J. Shlens, “Conditional image synthesis with auxiliary classifier gans,” *arXiv preprint arXiv:1610.09585*, 2016.

[30] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[31] P. Sangkloy, N. Burnell, C. Ham, and J. Hays, “The sketchy database: Learning to retrieve badly drawn bunnies,” *ACM Transactions on Graphics (proceedings of SIGGRAPH)*, 2016.

Cybersecurity Policies and Their Impact on Dynamic Data Driven Application Systems

Conference Paper · September 2017

DOI: 10.1109/FAS-W.2017.175

Conrad Tucker
Pennsylvania State University
96 PUBLICATIONS **493** CITATIONS

Cybersecurity Policies and their Impact on Dynamic Data Driven Application Systems

Conrad S. Tucker
Science and Policy Fellow
Brent Scowcroft Center on International Security
Atlantic Council
Washington, D.C., USA
ctucker@atlanticcouncil.org

Mathew Burrows
Director: Foresight, Strategy and Risks
Brent Scowcroft Center on International Security
Atlantic Council
Washington, D.C., USA
mburrows@atlanticcouncil.org

Kevin Lesniak
Data Scientist
Brent Scowcroft Center on International Security
Atlantic Council
Washington, D.C., USA
kevinalesniak@gmail.com

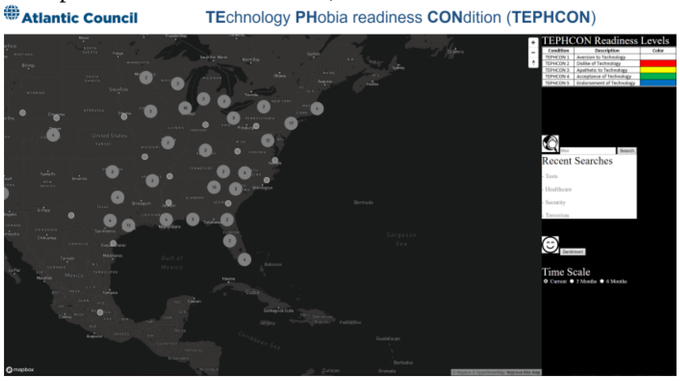
Samuel Klein
Program Assistant: Foresight, Strategy and Risks
Brent Scowcroft Center on International Security
Atlantic Council
Washington, D.C., USA
sklein@atlanticcouncil.org

***Abstract*—The objective of this paper is to explore how cybersecurity policies pertaining to data privacy, data acquisition, data fusion and data mining, impact the feasibility and functionality of Dynamic Data Driven Application Systems (DDDAS). In this work, a social media network model, will serve as the DDDAS of study in order to reveal how varying cybersecurity policies, could alter the functionality and usefulness of the system. At its fundamental level, the social media network model proposed in this work, TEchnology PHobia readiness CONdition (TEPHCON), seeks to serve as a decision support system that dynamically captures and visualizes society's affinity/aversion towards current or emerging technologies. However, at its core, this DDDAS is built upon a data platform that is highly susceptible to changes in cybersecurity policies. For example, a change in cybersecurity policies pertaining to freedom of speech in a cyber environment, may significantly alter the access and availability of publicly-available data. On the other hand, a more hands-off cyber policy pertaining to who controls cyber infrastructure networking speeds, may result in an imbalance of data that may threaten the veracity of TEPHCON. Therefore, cybersecurity policies are an integral component of DDDAS systems such as TEPHCON and as such, their impact should be explored in detail.**

Keywords*—DDDAS; social media; text mining; data visualization; technology phobia;

I. CONTRIBUTIONS

Cybersecurity seeks to protect both hardware and software components of computer systems from theft, damage, disruption, misdirection, and unintended use [1]. Policy makers have a direct impact on the success of cybersecurity efforts, as a stringent or lenient cybersecurity policy, could be the deciding factor in determining society's trust in cyber systems. Dynamic Data Driven Application Systems (DDDAS) inherently rely on a strong cyberinfrastructure that is robust against cybersecurity threats. Formally defined, DDDAS dynamically incorporate measurement data into a system's execution model in order to improve the accuracy or the speed of the model, with the execution model having the ability to control the measurement process [2]. From the definition of DDDAS, it can be seen that cybersecurity policies could impact DDDAS' ability to be *dynamic*, incorporate *measurement data*, and advance *execution models*.



TEPHCON Readiness Levels

Condition	Description	Color
TEPHCON 1	Aversion to Technology	⬜
TEPHCON 2	Dislike of Technology	🟥
TEPHCON 3	Apathetic to Technology	🟨
TEPHCON 4	Acceptance of Technology	🟩
TEPHCON 5	Endorsement of Technology	🟦

Figure 1: Visual Example of the TEPHCON Platform

The DDDAS model proposed in this work (Figure 1), TEchnology PHobia readiness CONdition (TEPHCON), is a data acquisition, data synthesizer, data mining and data visualization platform that seeks to serve as a decision support tool for assessing society's phobias (or lack thereof) towards current and future technologies. The readiness levels quantified within the TEPHCON platform are analogous to the Department of Defense's DEFense readiness CONditions

(DEFCON), with TEPHCON 5, being the lowest phobia level (i.e., affinity) of a given technology, and TEPHCON 1, being the highest phobia level (i.e., aversion) of a given technology. Here, the term *technology* is used in a general sense to include man-made hardware (e.g., smartphone) and software (e.g,. social media app) products across a wide range of domains.

A conceptual model of TEPHCON is presented in Figure 2 and includes the variable definitions and structure of data that is acquired and synthesized in this work. Given a total of *N* publicly-available sources of social media data:

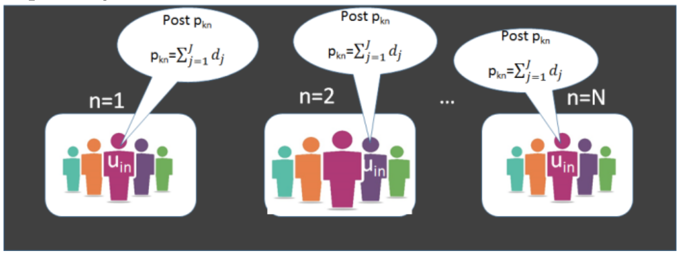


Figure 2: TEPHCON Conceptual Outline

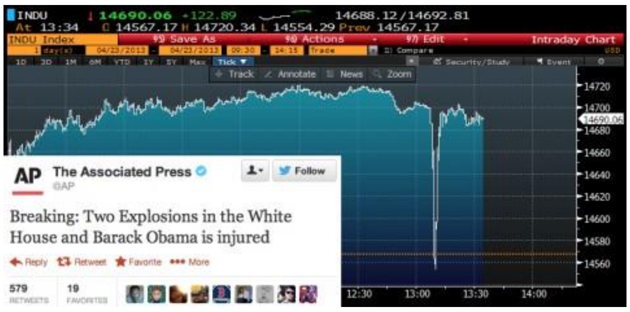
- ○ *n:* represents a publicly-available source of data (e.g., a publicly-available social media network such as Twitter) that contains data pertinent to TEPHCON ($n=1,\ldots, N$)
- ○ u_{in}: represents a user *i* of social media network *n*
- ○ p_{kn}: represents a post *k* generated by one of the users of social media network *n*
- ○ d_j: represents a data type (textual, image, geospatial, etc.) that is contained within a social media user *i's* post ($j=1,\ldots J$)

Based on the data generated within large scale social media networks, data-driven models can be created that serve as decision support systems. For example, using the textual data expressed by a social media user in their post p_{kn} , a sentiment of a post can be quantified by employing algorithms such as Sentistrength [3]. I.e., when post p_{kn} contains a textual data type $d_j= [w_1, w_2,\ldots,w_W]$, where w_i represents a word or emoticon, the Sentistrength algorithm would transform a textual phrase into a sentiment score [3].

The first objective of the TEPHCON platform is to separate *relevant* data (i.e., a specific topic of interest) from *irrelevant data* (i.e., all other data) by employing query filtering methods. With the time stamped *relevant* data, sentiment mining algorithms are employed to the different data types in order to provide a real time assessment of society's phobia level, given a specific topic of inquiry. This data is then visualized for decision support (Figure 1). Using the conceptual outline of TEPHCON and its associated data structures, several scenarios pertaining to cyber policies are presented in order to demonstrate how DDDAS systems such as TEPHCON may be impacted.

Cyber Policy Impacting the *Dynamic* Nature of DDDAS

- <u>Hypothetical Cyber policy #1:</u> *Providers of cyberinfrastructure have the authority to provide varying data speeds to access different data repositories, based on their economic objectives.*



AP The Associated Press @AP

Breaking: Two Explosions in the White House and Barack Obama is injured

Figure 3: Twitter Hack of the Associated Press Account and the Corresponding Stock Market Plunge [4]

On April 23, 2013, the official Twitter account of the Associated Press was hacked, sending the message "**Breaking: Two Explosions in the White House and Barack Obama is injured**" to its more than 2 million Twitter followers [4]. The economic ramifications of this cybersecurity breach were almost instantaneous, with the stock market losing approximately $136 Billion dollars in value (Figure 3), before correcting itself after another message was sent out informing everyone of the cybersecurity breach.

Given that this cybersecurity breach occurred on one of the social media platforms that could potentially be used as a TEPHCON data source, the question is: how resilient is TEPHCON to misinformation that could potentially threaten users' trust in the veracity of the system? In order to maintain trust, the DDDAS *execution model* must be able to:

- Detect misinformation generated within the data source
- Minimize the time needed to detect misinformation
- Automatically correct/quarantine/delete misinformation

Given the above scenario, let us assume that there exists three unique social media users from two separate social media platforms outlined in Figure 2, where *n=1* contains one user and *n=2* contains two users. Let us also assume that social media platform *n=1* has an existing financial partnership with the cyberinfrastructure provider that enables communication to and from this social medial network, enabling their users to achieve the highest possible data transmission rates. Users of social media platform *n=2* do not have a financial relationship and hence, their data transmission speeds are throttled, in order to give priority to users of social media platform *n=1*. When the Associated Press' Tweet about an explosion at the White House is first disseminated (i.e., at time *t=0*), the probability that the information is indeed true is relatively high, given the reputation and prestige of the source (i.e., the Associated Press). In other words:

p(bombing at the White House=True | Source=Associated Press)$\geq\beta$

Where β is the threshold for which information is deemed to be true. At time t>0, the objective of a resilient DDDAS would therefore be to ensure the veracity of each piece of information that is disseminated in a timely and efficient manner, especially one that has the potential to instantaneously reach 2 million other users, as is in the case of the Associated Press cybersecurity breach.

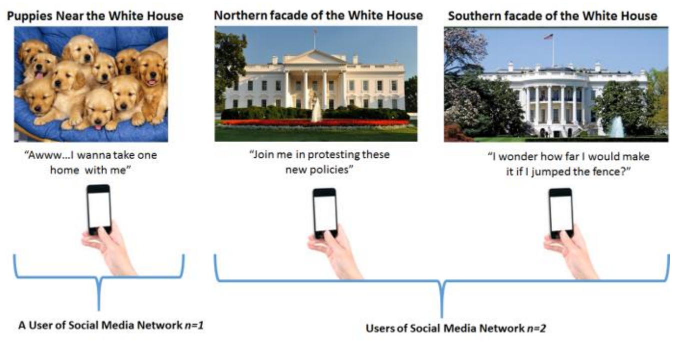

Puppies Near the White House
Northern facade of the White House
Southern facade of the White House
"Awww…I wanna take one home with me"
"Join me in protesting these new policies"
"I wonder how far I would make it if I jumped the fence?"
A User of Social Media Network n=1
Users of Social Media Network n=2

Figure 4: 3 different users sharing pictures on social media

In order to falsify the claim of the White House bombing, Figure 4 presents a scenario where 3 images captured by 3 different individuals are shared on different social media networks, moments after the Associated Press cybersecurity breach. Assuming that these three messages were generated within the vicinity of the White House, image 1 that shows a litter of puppies, would do little to reduce the *p(bombing at the White House=true)*. Furthermore, due to the fact that the user of this social media network exists in *n=1* (i.e., the social media network that has a financial partnership enabling faster, priority data sharing), the image from user 1 may be captured by TEPHCON before images 2 (i.e., Northern facade of the White House) and image 3 (i.e,. Southern facade of the White House). For users of social media network *n*=2, the other two images (Northern facade of the White House and Southern facade of the White House) which may actually help decipher whether the Associated Press Tweet is indeed false, may be delayed in terms of data transmission due to the lack of existing partnerships established between *n=2* and the cyberinfrastructure provider. In both images showing the White House, it can be clearly seen that there is no damage done to the infrastructure, hereby bringing into question the news story disseminated by the Associated Press. Bodnar et al. have proposed algorithms aimed at increasing the veracity of information that is disseminated in online social media networks, based on a proposed user trust network [5]. As cyber policies change however, there is an ever-increasing need for novel and robust algorithms that enable DDDAS to maintain their dynamic characteristics.

Cyber Policy Impacting the *Execution Models* of DDDAS

- <u>Hypothetical Cyber policy #2:</u> *Posts by users of social media may be deemed hate speech/speech inciting violence and may be admissible as evidence in court.*

The balance between freedom of speech and national security is an intricate one. While freedom of speech is a fundamental component of free societies, there is a risk that certain speech has the potential to incite violence or be used as terrorist propaganda. Looking at the example in Figure 4, we observe that in addition to the images shared by each of the social media users, a corresponding textual message is also sent. A cyber policy that exposes a social media user to potential legal liability for their posts, may severely limit the quality and quantity of data that is generated within social media networks, hereby potentially diminishing TEPHCON's ability to serve as a real time visual decision support tool that reflects the views of a significant portion of society. For example, the message accompanying the "Northern façade of the White House" reads [*join me in protesting these new policies*], which may in some cases, be viewed as an attempt to incite a rally/protest. Similarly, the message accompanying the "Southern facade of the White House" reads [*I wonder how far I would make it if I jumped the fence?*], a hypothetical question, but however, a question that could be perceived as a preemptive attempt to cause a national security incident. While the textual messages of these users may be concerning, the actual images that they share of the White House being safe from the threat of a bomb, has the potential to separate misinformation from true information (e.g., the Associated Press Twitter breach). A cybersecurity policy that threatens the freedom of speech, may have resulted in the social media users who posted pictures of the White House, to have abstained from posting altogether due to fear of being criminally prosecuted due to their hypothetical statements.

II. RESULTS AND CONCLUSION

The cybersecurity examples presented in this work illustrate how DDDAS systems such as TEPHCON, could potentially be significantly impacted, based on the cyber systems and data policies adopted by a society. The examples demonstrate how the dynamic nature of DDDAS systems could be altered, simply based on a cyber infrastructure policy pertaining to network management and data prioritization. In addition, the quantity and quality of data available for DDDAS systems such as TEPHCON, could be adversely affected due to the cybersecurity policies pertaining to freedom of speech, liability assignment given a cybersecurity incident, etc. These and other cyber security policies are integral to the real-world feasibility of DDDAS systems such as TEPHCON.

ACKNOWLEDGMENT

This research is funded by the Atlantic Council. Any opinions, findings, or conclusions found in this paper are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] C.-W. Ten, G. Manimaran, and C.-C. Liu, "Cybersecurity for critical infrastructures: Attack and defense modeling," *IEEE Trans. Syst. Man Cybern.-Part Syst. Hum.*, vol. 40, no. 4, pp. 853–865, 2010.
[2] A. Aved, F. Darema, and E. Blasch, *Dynamic data driven application systems*. 2014.
[3] M. Thelwall, "Heart and soul: Sentiment strength detection in the social web with sentistrength," *Proc. CyberEmotions*, vol. 5, pp. 1–14, 2013.
[4] "Syrian hackers claim AP hack that tipped stock market by $136 billion. Is it terrorism? - The Washington Post." [Online]. Available: https://www.washingtonpost.com/news/worldviews/wp/2013/04/23/syrian-hackers-claim-ap-hack-that-tipped-stock-market-by-136-billion-is-it-terrorism/?utm_term=.9d30b5901b86. [Accessed: 26-Jul-2017].
[5] T. Bodnar, C. Tucker, K. Hopkinson, and S. G. Bilén, "Increasing the veracity of event detection on social media networks through user trust modeling," in *Big Data (Big Data), 2014 IEEE International Conference on*, 2014, pp. 636–643.

AI and Social Science – Brendan O'Connor
cognition, language, social systems;
statistics, visualization, computation

Cosine similarity, Pearson correlation, and OLS coefficients

Posted on March 13, 2012

Cosine similarity, Pearson correlations, and OLS coefficients can all be viewed as vari
tweaked in different ways for centering and magnitude (i.e. location and scale, or som

Details:

You have two vectors x and y and want to measure similarity between them. A basic s
inner product

$$Inner(x, y) = \sum_i x_i y_i = \langle x, y \rangle$$

If x tends to be high where y is also high, and low where y is low, the inner product wi
more similar.

The inner product is unbounded. One way to make it bounded between -1 and 1 is to c
norms, giving the **cosine similarity**

$$CosSim(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} = \frac{\langle x, y \rangle}{||x||\ ||y||}$$

This is actually bounded between 0 and 1 if x and y are non-negative. Cosine similarit
cosine of the angle between the two vectors; you can illustrate this for vectors in $\mathbb{R}^2$ (e

Cosine similarity is not invariant to shifts. If x was shifted to x+1, the cosine similarity
invariant, though, is the **Pearson correlation**. Let $\bar{x}$ and $\bar{y}$ be the respective means

$$
\begin{aligned}
Corr(x,y) &= \frac{\sum_i (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum (x_i-\bar{x})^2}\sqrt{\sum (y_i-\bar{y})^2}} \\
&= \frac{\langle x-\bar{x},\ y-\bar{y}\rangle}{||x-\bar{x}||\ ||y-\bar{y}||} \\
&= CosSim(x-\bar{x}, y-\bar{y})
\end{aligned}
$$

Correlation is the cosine similarity between centered versions of x and y, again bound
usually talk about cosine similarity in terms of vector angles, but it can be loosely thou
you think of the vectors as paired samples. Unlike the cosine, the correlation is invari
changes of x and y.

This isn't the usual way to derive the Pearson correlation; usually it's presented as a n
covariance, which is a centered average inner product (no normalization)

$$Cov(x,y) = \frac{\sum (x_i-\bar{x})(y_i-\bar{y})}{n} = \frac{\langle x-\bar{x},\ y-\bar{y}\rangle}{n}$$

Finally, these are all related to the coefficient in a **one-variable linear regression**
with Gaussian noise, whose MLE is the least-squares problem $\arg\min_a \sum (y_i - a x_i)^2$,
a is

$$OLSCoef(x,y) = \frac{\sum x_i y_i}{\sum x_i^2} = \frac{\langle x, y\rangle}{||x||^2}$$

This looks like another normalized inner product. But unlike cosine similarity, we are
— instead we only use x's norm (and use it twice): denominator of $||x||\ ||y||$ versus $||x$

Not normalizing for y is what you want for the linear regression: if y was stretched to
would need to increase a to match, to get your predictions spread out too.

Often it's desirable to do the OLS model with an intercept term: $\min_{a,b} \sum (y - a x_i - b)^2$

$$
\begin{aligned}
OLSCoefWithIntercept(x,y) &= \frac{\sum (x_i - \bar{x}) y_i}{\sum (x_i - \bar{x})^2} = \frac{\langle x-\bar{x},}{||x-\bar{x}} \\
&= OLSCoef(x-\bar{x}, y)
\end{aligned}
$$

It's different because the intercept term picks up the slack associated with where x's c

OLSCoefWithIntercept is invariant to shifts of x. It's still different than cosine similar normalizing at all for y. Though, subtly, it does actually control for shifts of y. This isn but with a little arithmetic it's easy to derive that $\langle x-\bar{x},\ y\rangle = \langle x-\bar{x},\ y+c\rangle$ for any nice geometric interpretation of this.)

Finally, what if x and y are standardized: both centered and normalized to unit standa coefficient for that is the same as the Pearson correlation between the original vectors means or if it's a useful fact, but:

$$OLSCoef\left(\sqrt{n}\frac{x-\bar{x}}{||x-\bar{x}||}, \sqrt{n}\frac{y-\bar{y}}{||y-\bar{y}||}\right) = Corr(x,y)$$

Summarizing: Cosine similarity is normalized inner product. Pearson correlation is ce one-variable OLS coefficient is like cosine but with one-sided normalization. With an

Of course we need a summary table. "Symmetric" means, if you swap the inputs, do y "Invariant to shift in input" means, if you add an arbitrary constant to either input, do

Function	Equation	Symmetric?	Output range	Invariant to shift in input
Inner(x,y)	$\langle x, y\rangle$	Yes	$\mathbb{R}$	No
CosSim(x,y)	$\frac{\langle x, y\rangle}{\|\|x\|\|\ \|\|y\|\|}$	Yes	[-1,1] or [0,1] if inputs non-neg	No
Corr(x,y)	$\frac{\langle x-\bar{x},\ y-\bar{y}\rangle}{\|\|x-\bar{x}\|\|\ \|\|y-\bar{y}\|\|}$	Yes	[-1,1]	Yes
Cov(x,y)	$\frac{\langle x-\bar{x},\ y-\bar{y}\rangle}{n}$	Yes	$\mathbb{R}$	Yes

OLSCoefNoIntcpt(x,y)	$\frac{\langle x, y \rangle}{\|\|x\|\|^2}$	No	$\mathbb{R}$	No
OLSCoefWithIntcpt(x,y)	$\frac{\langle x - \bar{x},\ y \rangle}{\|\|x - \bar{x}\|\|^2}$	No	$\mathbb{R}$	Yes

Are there any implications? I've been wondering for a while why cosine similarity ten
language processing applications. Maybe this has something to do with it. Or not. On
product stuff is computational strategies to make it faster when there's high-dimensic
Friedman et al. 2010 glmnet paper talks about this in the context of coordinate descer
Dhillon et al., NIPS 2011 applies LSH in a similar setting (but haven't read it yet). Anc
LSH for cosine similarity; e.g. van Durme and Lall 2010 [slides].

Any other cool identities? Any corrections to the above?

References: I use Hastie et al 2009, chapter 3 to look up linear regression, but it's cov
places. I linked to a nice chapter in Tufte's little 1974 book that he wrote before he we
visualization stuff. (He calls it "two-variable regression", but I think "one-variable reg
"one-feature" or "one-covariate" might be most accurate.) In my experience, cosine si
often in text processing or machine learning contexts.

This entry was posted in Uncategorized. Bookmark the permalink.

23 Responses to *Cosine similarity, Pearson correlation, and OLS coefficie*

VC **Victor Chahuneau** *says:*
March 15, 2012 at 1:21 am

I think your OLSCoefWithIntercept is wrong unless y is centered: the right part of the dot product shoul
Then the invariance by translation is obvious…
Otherwise you would get <x-, y+c> = <x-,y> + c(n-1)
See Wikipedia for the equation

VC **Victor Chahuneau** *says:*

Chapter 3

Finding Similar Items

A fundamental data-mining problem is to examine data for "similar" items. We shall take up applications in Section 3.1, but an example would be looking at a collection of Web pages and finding near-duplicate pages. These pages could be plagiarisms, for example, or they could be mirrors that have almost the same content but differ in information about the host and about other mirrors.

We begin by phrasing the problem of similarity as one of finding sets with a relatively large intersection. We show how the problem of finding textually similar documents can be turned into such a set problem by the technique known as "shingling." Then, we introduce a technique called "minhashing," which compresses large sets in such a way that we can still deduce the similarity of the underlying sets from their compressed versions. Other techniques that work when the required degree of similarity is very high are covered in Section 3.9.

Another important problem that arises when we search for similar items of any kind is that there may be far too many pairs of items to test each pair for their degree of similarity, even if computing the similarity of any one pair can be made very easy. That concern motivates a technique called "locality-sensitive hashing," for focusing our search on pairs that are most likely to be similar.

Finally, we explore notions of "similarity" that are not expressible as intersection of sets. This study leads us to consider the theory of distance measures in arbitrary spaces. It also motivates a general framework for locality-sensitive hashing that applies for other definitions of "similarity."

3.1 Applications of Near-Neighbor Search

We shall focus initially on a particular notion of "similarity": the similarity of sets by looking at the relative size of their intersection. This notion of similarity is called "Jaccard similarity," and will be introduced in Section 3.1.1. We then examine some of the uses of finding similar sets. These include finding textually similar documents and collaborative filtering by finding similar customers and similar products. In order to turn the problem of textual similarity of documents into one of set intersection, we use a technique called "shingling," which is introduced in Section 3.2.

3.1.1 Jaccard Similarity of Sets

The *Jaccard similarity* of sets S and T is $|S \cap T|/|S \cup T|$, that is, the ratio of the size of the intersection of S and T to the size of their union. We shall denote the Jaccard similarity of S and T by $\text{SIM}(S, T)$.

Example 3.1 : In Fig. 3.1 we see two sets S and T. There are three elements in their intersection and a total of eight elements that appear in S or T or both. Thus, $\text{SIM}(S, T) = 3/8$. □

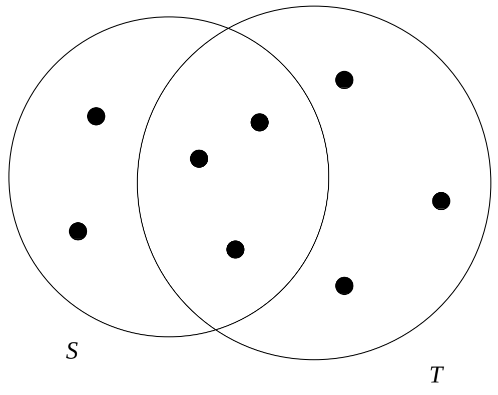


Figure 3.1: Two sets with Jaccard similarity 3/8

3.1.2 Similarity of Documents

An important class of problems that Jaccard similarity addresses well is that of finding textually similar documents in a large corpus such as the Web or a collection of news articles. We should understand that the aspect of similarity we are looking at here is character-level similarity, not "similar meaning," which requires us to examine the words in the documents and their uses. That problem is also interesting but is addressed by other techniques, which we hinted at in Section 1.3.1. However, textual similarity also has important uses. Many of these involve finding duplicates or near duplicates. First, let us observe that testing whether two documents are exact duplicates is easy; just compare the two documents character-by-character, and if they ever differ then they are not the same. However, in many applications, the documents are not identical, yet they share large portions of their text. Here are some examples:

Plagiarism

Finding plagiarized documents tests our ability to find textual similarity. The plagiarizer may extract only some parts of a document for his own. He may alter a few words and may alter the order in which sentences of the original appear. Yet the resulting document may still contain 50% or more of the original. No simple process of comparing documents character by character will detect a sophisticated plagiarism.

Mirror Pages

It is common for important or popular Web sites to be duplicated at a number of hosts, in order to share the load. The pages of these *mirror* sites will be quite similar, but are rarely identical. For instance, they might each contain information associated with their particular host, and they might each have links to the other mirror sites but not to themselves. A related phenomenon is the appropriation of pages from one class to another. These pages might include class notes, assignments, and lecture slides. Similar pages might change the name of the course, year, and make small changes from year to year. It is important to be able to detect similar pages of these kinds, because search engines produce better results if they avoid showing two pages that are nearly identical within the first page of results.

Articles from the Same Source

It is common for one reporter to write a news article that gets distributed, say through the Associated Press, to many newspapers, which then publish the article on their Web sites. Each newspaper changes the article somewhat. They may cut out paragraphs, or even add material of their own. They most likely will surround the article by their own logo, ads, and links to other articles at their site. However, the core of each newspaper's page will be the original article. News aggregators, such as Google News, try to find all versions of such an article, in order to show only one, and that task requires finding when two Web pages are textually similar, although not identical.[1]

3.1.3 Collaborative Filtering as a Similar-Sets Problem

Another class of applications where similarity of sets is very important is called *collaborative filtering*, a process whereby we recommend to users items that were liked by other users who have exhibited similar tastes. We shall investigate collaborative filtering in detail in Section 9.3, but for the moment let us see some common examples.

[1] News aggregation also involves finding articles that are about the same topic, even though not textually similar. This problem too can yield to a similarity search, but it requires techniques other than Jaccard similarity of sets.

On-Line Purchases

Amazon.com has millions of customers and sells millions of items. Its database records which items have been bought by which customers. We can say two customers are similar if their sets of purchased items have a high Jaccard similarity. Likewise, two items that have sets of purchasers with high Jaccard similarity will be deemed similar. Note that, while we might expect mirror sites to have Jaccard similarity above 90%, it is unlikely that any two customers have Jaccard similarity that high (unless they have purchased only one item). Even a Jaccard similarity like 20% might be unusual enough to identify customers with similar tastes. The same observation holds for items; Jaccard similarities need not be very high to be significant.

Collaborative filtering requires several tools, in addition to finding similar customers or items, as we discuss in Chapter 9. For example, two Amazon customers who like science-fiction might each buy many science-fiction books, but only a few of these will be in common. However, by combining similarity-finding with clustering (Chapter 7), we might be able to discover that science-fiction books are mutually similar and put them in one group. Then, we can get a more powerful notion of customer-similarity by asking whether they made purchases within many of the same groups.

Movie Ratings

NetFlix records which movies each of its customers rented, and also the ratings assigned to those movies by the customers. We can see movies as similar if they were rented or rated highly by many of the same customers, and see customers as similar if they rented or rated highly many of the same movies. The same observations that we made for Amazon above apply in this situation: similarities need not be high to be significant, and clustering movies by genre will make things easier.

When our data consists of ratings rather than binary decisions (bought/did not buy or liked/disliked), we cannot rely simply on sets as representations of customers or items. Some options are:

1. Ignore low-rated customer/movie pairs; that is, treat these events as if the customer never watched the movie.

2. When comparing customers, imagine two set elements for each movie, "liked" and "hated." If a customer rated a movie highly, put the "liked" for that movie in the customer's set. If they gave a low rating to a movie, put "hated" for that movie in their set. Then, we can look for high Jaccard similarity among these sets. We can do a similar trick when comparing movies.

3. If ratings are 1-to-5-stars, put a movie in a customer's set n times if they rated the movie n-stars. Then, use *Jaccard similarity for bags* when measuring the similarity of customers. The Jaccard similarity for bags

B and C is defined by counting an element n times in the intersection if n is the minimum of the number of times the element appears in B and C. In the union, we count the element the sum of the number of times it appears in B and in C.[2]

Example 3.2 : The bag-similarity of bags $\{a, a, a, b\}$ and $\{a, a, b, b, c\}$ is 1/3. The intersection counts a twice and b once, so its size is 3. The size of the union of two bags is always the sum of the sizes of the two bags, or 9 in this case. Since the highest possible Jaccard similarity for bags is 1/2, the score of 1/3 indicates the two bags are quite similar, as should be apparent from an examination of their contents. □

3.1.4 Exercises for Section 3.1

Exercise 3.1.1 : Compute the Jaccard similarities of each pair of the following three sets: $\{1, 2, 3, 4\}$, $\{2, 3, 5, 7\}$, and $\{2, 4, 6\}$.

Exercise 3.1.2 : Compute the Jaccard bag similarity of each pair of the following three bags: $\{1, 1, 1, 2\}$, $\{1, 1, 2, 2, 3\}$, and $\{1, 2, 3, 4\}$.

!! **Exercise 3.1.3 :** Suppose we have a universal set U of n elements, and we choose two subsets S and T at random, each with m of the n elements. What is the expected value of the Jaccard similarity of S and T?

3.2 Shingling of Documents

The most effective way to represent documents as sets, for the purpose of identifying lexically similar documents is to construct from the document the set of short strings that appear within it. If we do so, then documents that share pieces as short as sentences or even phrases will have many common elements in their sets, even if those sentences appear in different orders in the two documents. In this section, we introduce the simplest and most common approach, shingling, as well as an interesting variation.

3.2.1 k-Shingles

A document is a string of characters. Define a k-shingle for a document to be any substring of length k found within the document. Then, we may associate

[2]Although the union for bags is normally (e.g., in the SQL standard) defined to have the sum of the number of copies in the two bags, this definition causes some inconsistency with the Jaccard similarity for sets. Under this definition of bag union, the maximum Jaccard similarity is 1/2, not 1, since the union of a set with itself has twice as many elements as the intersection of the same set with itself. If we prefer to have the Jaccard similarity of a set with itself be 1, we can redefine the union of bags to have each element appear the maximum number of times it appears in either of the two bags. This change does not simply double the similarity in each case, but it also gives a reasonable measure of bag similarity.

with each document the set of k-shingles that appear one or more times within that document.

Example 3.3 : Suppose our document D is the string `abcdabd`, and we pick $k = 2$. Then the set of 2-shingles for D is $\{\texttt{ab}, \texttt{bc}, \texttt{cd}, \texttt{da}, \texttt{bd}\}$.

Note that the substring `ab` appears twice within D, but appears only once as a shingle. A variation of shingling produces a bag, rather than a set, so each shingle would appear in the result as many times as it appears in the document. However, we shall not use bags of shingles here. □

There are several options regarding how white space (blank, tab, newline, etc.) is treated. It probably makes sense to replace any sequence of one or more white-space characters by a single blank. That way, we distinguish shingles that cover two or more words from those that do not.

Example 3.4 : If we use $k = 9$, but eliminate whitespace altogether, then we would see some lexical similarity in the sentences "`The plane was ready for touch down`". and "`The quarterback scored a touchdown`". However, if we retain the blanks, then the first has shingles `touch dow` and `ouch down`, while the second has `touchdown`. If we eliminated the blanks, then both would have `touchdown`. □

3.2.2 Choosing the Shingle Size

We can pick k to be any constant we like. However, if we pick k too small, then we would expect most sequences of k characters to appear in most documents. If so, then we could have documents whose shingle-sets had high Jaccard similarity, yet the documents had none of the same sentences or even phrases. As an extreme example, if we use $k = 1$, most Web pages will have most of the common characters and few other characters, so almost all Web pages will have high similarity.

How large k should be depends on how long typical documents are and how large the set of typical characters is. The important thing to remember is:

- k should be picked large enough that the probability of any given shingle appearing in any given document is low.

Thus, if our corpus of documents is emails, picking $k = 5$ should be fine. To see why, suppose that only letters and a general white-space character appear in emails (although in practice, most of the printable ASCII characters can be expected to appear occasionally). If so, then there would be $27^5 =$ 14,348,907 possible shingles. Since the typical email is much smaller than 14 million characters long, we would expect $k = 5$ to work well, and indeed it does.

However, the calculation is a bit more subtle. Surely, more than 27 characters appear in emails, However, all characters do not appear with equal probability. Common letters and blanks dominate, while "z" and other letters that have high point-value in Scrabble are rare. Thus, even short emails will have many 5-shingles consisting of common letters, and the chances of unrelated emails sharing these common shingles is greater than would be implied by the calculation in the paragraph above. A good rule of thumb is to imagine that there are only 20 characters and estimate the number of k-shingles as 20^k. For large documents, such as research articles, choice $k = 9$ is considered safe.

3.2.3 Hashing Shingles

Instead of using substrings directly as shingles, we can pick a hash function that maps strings of length k to some number of buckets and treat the resulting bucket number as the shingle. The set representing a document is then the set of integers that are bucket numbers of one or more k-shingles that appear in the document. For instance, we could construct the set of 9-shingles for a document and then map each of those 9-shingles to a bucket number in the range 0 to $2^{32} - 1$. Thus, each shingle is represented by four bytes instead of nine. Not only has the data been compacted, but we can now manipulate (hashed) shingles by single-word machine operations.

Notice that we can differentiate documents better if we use 9-shingles and hash them down to four bytes than to use 4-shingles, even though the space used to represent a shingle is the same. The reason was touched upon in Section 3.2.2. If we use 4-shingles, most sequences of four bytes are unlikely or impossible to find in typical documents. Thus, the effective number of different shingles is much less than $2^{32} - 1$. If, as in Section 3.2.2, we assume only 20 characters are frequent in English text, then the number of different 4-shingles that are likely to occur is only $(20)^4 = 160{,}000$. However, if we use 9-shingles, there are many more than 2^{32} likely shingles. When we hash them down to four bytes, we can expect almost any sequence of four bytes to be possible, as was discussed in Section 1.3.2.

3.2.4 Shingles Built from Words

An alternative form of shingle has proved effective for the problem of identifying similar news articles, mentioned in Section 3.1.2. The exploitable distinction for this problem is that the news articles are written in a rather different style than are other elements that typically appear on the page with the article. News articles, and most prose, have a lot of stop words (see Section 1.3.1), the most common words such as "and," "you," "to," and so on. In many applications, we want to ignore stop words, since they don't tell us anything useful about the article, such as its topic.

However, for the problem of finding similar news articles, it was found that defining a shingle to be a stop word followed by the next two words, regardless of whether or not they were stop words, formed a useful set of shingles. The advantage of this approach is that the news article would then contribute more shingles to the set representing the Web page than would the surrounding elements. Recall that the goal of the exercise is to find pages that had the same articles, regardless of the surrounding elements. By biasing the set of shingles in favor of the article, pages with the same article and different surrounding material have higher Jaccard similarity than pages with the same surrounding material but with a different article.

Example 3.5 : An ad might have the simple text "`Buy Sudzo`." However, a news article with the same idea might read something like "*A* `spokesperson` *for the* `Sudzo Corporation revealed today` *that* `studies` *have* `shown` *it is* `good` *for* `people` *to* `buy Sudzo products`." Here, we have italicized all the likely stop words, although there is no set number of the most frequent words that should be considered stop words. The first three shingles made from a stop word and the next two following are:

```
A spokesperson for
for the Sudzo
the Sudzo Corporation
```

There are nine shingles from the sentence, but none from the "ad." □

3.2.5 Exercises for Section 3.2

Exercise 3.2.1 : What are the first ten 3-shingles in the first sentence of Section 3.2?

Exercise 3.2.2 : If we use the stop-word-based shingles of Section 3.2.4, and we take the stop words to be all the words of three or fewer letters, then what are the shingles in the first sentence of Section 3.2?

Exercise 3.2.3 : What is the largest number of k-shingles a document of n bytes can have? You may assume that the size of the alphabet is large enough that the number of possible strings of length k is at least as n.

3.3 Similarity-Preserving Summaries of Sets

Sets of shingles are large. Even if we hash them to four bytes each, the space needed to store a set is still roughly four times the space taken by the document. If we have millions of documents, it may well not be possible to store all the shingle-sets in main memory.[3]

Our goal in this section is to replace large sets by much smaller representations called "signatures." The important property we need for signatures is that we can compare the signatures of two sets and estimate the Jaccard similarity of the underlying sets from the signatures alone. It is not possible that

[3]There is another serious concern: even if the sets fit in main memory, the number of pairs may be too great for us to evaluate the similarity of each pair. We take up the solution to this problem in Section 3.4.

the signatures give the exact similarity of the sets they represent, but the estimates they provide are close, and the larger the signatures the more accurate the estimates. For example, if we replace the 200,000-byte hashed-shingle sets that derive from 50,000-byte documents by signatures of 1000 bytes, we can usually get within a few percent.

3.3.1 Matrix Representation of Sets

Before explaining how it is possible to construct small signatures from large sets, it is helpful to visualize a collection of sets as their *characteristic matrix*. The columns of the matrix correspond to the sets, and the rows correspond to elements of the universal set from which elements of the sets are drawn. There is a 1 in row r and column c if the element for row r is a member of the set for column c. Otherwise the value in position (r, c) is 0.

Element	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

Figure 3.2: A matrix representing four sets

Example 3.6 : In Fig. 3.2 is an example of a matrix representing sets chosen from the universal set $\{a, b, c, d, e\}$. Here, $S_1 = \{a, d\}$, $S_2 = \{c\}$, $S_3 = \{b, d, e\}$, and $S_4 = \{a, c, d\}$. The top row and leftmost columns are not part of the matrix, but are present only to remind us what the rows and columns represent. □

It is important to remember that the characteristic matrix is unlikely to be the way the data is stored, but it is useful as a way to visualize the data. For one reason not to store data as a matrix, these matrices are almost always *sparse* (they have many more 0's than 1's) in practice. It saves space to represent a sparse matrix of 0's and 1's by the positions in which the 1's appear. For another reason, the data is usually stored in some other format for other purposes.

As an example, if rows are products, and columns are customers, represented by the set of products they bought, then this data would really appear in a database table of purchases. A tuple in this table would list the item, the purchaser, and probably other details about the purchase, such as the date and the credit card used.

3.3.2 Minhashing

The signatures we desire to construct for sets are composed of the results of a large number of calculations, say several hundred, each of which is a "minhash" of the characteristic matrix. In this section, we shall learn how a minhash is computed in principle, and in later sections we shall see how a good approximation to the minhash is computed in practice.

To *minhash* a set represented by a column of the characteristic matrix, pick a permutation of the rows. The minhash value of any column is the number of the first row, in the permuted order, in which the column has a 1.

Example 3.7: Let us suppose we pick the order of rows *beadc* for the matrix of Fig. 3.2. This permutation defines a minhash function h that maps sets to rows. Let us compute the minhash value of set S_1 according to h. The first column, which is the column for set S_1, has 0 in row b, so we proceed to row e, the second in the permuted order. There is again a 0 in the column for S_1, so we proceed to row a, where we find a 1. Thus. $h(S_1) = a$.

Element	S_1	S_2	S_3	S_4
b	0	0	1	0
e	0	0	1	0
a	1	0	0	1
d	1	0	1	1
c	0	1	0	1

Figure 3.3: A permutation of the rows of Fig. 3.2

Although it is not physically possible to permute very large characteristic matrices, the minhash function h implicitly reorders the rows of the matrix of Fig. 3.2 so it becomes the matrix of Fig. 3.3. In this matrix, we can read off the values of h by scanning from the top until we come to a 1. Thus, we see that $h(S_2) = c$, $h(S_3) = b$, and $h(S_4) = a$. □

3.3.3 Minhashing and Jaccard Similarity

There is a remarkable connection between minhashing and Jaccard similarity of the sets that are minhashed.

- The probability that the minhash function for a random permutation of rows produces the same value for two sets equals the Jaccard similarity of those sets.

To see why, we need to picture the columns for those two sets. If we restrict ourselves to the columns for sets S_1 and S_2, then rows can be divided into three classes:

1. Type X rows have 1 in both columns.

2. Type Y rows have 1 in one of the columns and 0 in the other.

3. Type Z rows have 0 in both columns.

Since the matrix is sparse, most rows are of type Z. However, it is the ratio of the numbers of type X and type Y rows that determine both $\text{SIM}(S_1, S_2)$ and the probability that $h(S_1) = h(S_2)$. Let there be x rows of type X and y rows of type Y. Then $\text{SIM}(S_1, S_2) = x/(x + y)$. The reason is that x is the size of $S_1 \cap S_2$ and $x + y$ is the size of $S_1 \cup S_2$.

Now, consider the probability that $h(S_1) = h(S_2)$. If we imagine the rows permuted randomly, and we proceed from the top, the probability that we shall meet a type X row before we meet a type Y row is $x/(x + y)$. But if the first row from the top other than type Z rows is a type X row, then surely $h(S_1) = h(S_2)$. On the other hand, if the first row other than a type Z row that we meet is a type Y row, then the set with a 1 gets that row as its minhash value. However the set with a 0 in that row surely gets some row further down the permuted list. Thus, we know $h(S_1) \neq h(S_2)$ if we first meet a type Y row. We conclude the probability that $h(S_1) = h(S_2)$ is $x/(x + y)$, which is also the Jaccard similarity of S_1 and S_2.

3.3.4 Minhash Signatures

Again think of a collection of sets represented by their characteristic matrix M. To represent sets, we pick at random some number n of permutations of the rows of M. Perhaps 100 permutations or several hundred permutations will do. Call the minhash functions determined by these permutations $h_1, h_2, \ldots, h_n$. From the column representing set S, construct the *minhash signature* for S, the vector $[h_1(S), h_2(S), \ldots, h_n(S)]$. We normally represent this list of hash-values as a column. Thus, we can form from matrix M a *signature matrix*, in which the ith column of M is replaced by the minhash signature for (the set of) the ith column.

Note that the signature matrix has the same number of columns as M but only n rows. Even if M is not represented explicitly, but in some compressed form suitable for a sparse matrix (e.g., by the locations of its 1's), it is normal for the signature matrix to be much smaller than M.

3.3.5 Computing Minhash Signatures

It is not feasible to permute a large characteristic matrix explicitly. Even picking a random permutation of millions or billions of rows is time-consuming, and the necessary sorting of the rows would take even more time. Thus, permuted matrices like that suggested by Fig. 3.3, while conceptually appealing, are not implementable.

Fortunately, it is possible to simulate the effect of a random permutation by a random hash function that maps row numbers to as many buckets as there are rows. A hash function that maps integers $0, 1, \ldots, k-1$ to bucket numbers 0 through $k-1$ typically will map some pairs of integers to the same bucket and leave other buckets unfilled. However, the difference is unimportant as long as

k is large and there are not too many collisions. We can maintain the fiction that our hash function h "permutes" row r to position $h(r)$ in the permuted order.

Thus, instead of picking n random permutations of rows, we pick n randomly chosen hash functions $h_1, h_2, \ldots, h_n$ on the rows. We construct the signature matrix by considering each row in their given order. Let $\text{SIG}(i, c)$ be the element of the signature matrix for the ith hash function and column c. Initially, set $\text{SIG}(i, c)$ to ∞ for all i and c. We handle row r by doing the following:

1. Compute $h_1(r), h_2(r), \ldots, h_n(r)$.

2. For each column c do the following:

   (a) If c has 0 in row r, do nothing.

   (b) However, if c has 1 in row r, then for each $i = 1, 2, \ldots, n$ set $\text{SIG}(i, c)$ to the smaller of the current value of $\text{SIG}(i, c)$ and $h_i(r)$.

Row	S_1	S_2	S_3	S_4	$x + 1 \mod 5$	$3x + 1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

Figure 3.4: Hash functions computed for the matrix of Fig. 3.2

Example 3.8: Let us reconsider the characteristic matrix of Fig. 3.2, which we reproduce with some additional data as Fig. 3.4. We have replaced the letters naming the rows by integers 0 through 4. We have also chosen two hash functions: $h_1(x) = x+1 \mod 5$ and $h_2(x) = 3x+1 \mod 5$. The values of these two functions applied to the row numbers are given in the last two columns of Fig. 3.4. Notice that these simple hash functions are true permutations of the rows, but a true permutation is only possible because the number of rows, 5, is a prime. In general, there will be collisions, where two rows get the same hash value.

Now, let us simulate the algorithm for computing the signature matrix. Initially, this matrix consists of all ∞'s:

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

First, we consider row 0 of Fig. 3.4. We see that the values of $h_1(0)$ and $h_2(0)$ are both 1. The row numbered 0 has 1's in the columns for sets S_1 and

S_4, so only these columns of the signature matrix can change. As 1 is less than ∞, we do in fact change both values in the columns for S_1 and S_4. The current estimate of the signature matrix is thus:

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Now, we move to the row numbered 1 in Fig. 3.4. This row has 1 only in S_3, and its hash values are $h_1(1) = 2$ and $h_2(1) = 4$. Thus, we set $\text{SIG}(1,3)$ to 2 and $\text{SIG}(2,3)$ to 4. All other signature entries remain as they are because their columns have 0 in the row numbered 1. The new signature matrix:

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

The row of Fig. 3.4 numbered 2 has 1's in the columns for S_2 and S_4, and its hash values are $h_1(2) = 3$ and $h_2(2) = 2$. We could change the values in the signature for S_4, but the values in this column of the signature matrix, $[1, 1]$, are each less than the corresponding hash values $[3, 2]$. However, since the column for S_2 still has ∞'s, we replace it by $[3, 2]$, resulting in:

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

Next comes the row numbered 3 in Fig. 3.4. Here, all columns but S_2 have 1, and the hash values are $h_1(3) = 4$ and $h_2(3) = 0$. The value 4 for h_1 exceeds what is already in the signature matrix for all the columns, so we shall not change any values in the first row of the signature matrix. However, the value 0 for h_2 is less than what is already present, so we lower $\text{SIG}(2,1)$, $\text{SIG}(2,3)$ and $\text{SIG}(2,4)$ to 0. Note that we cannot lower $\text{SIG}(2,2)$ because the column for S_2 in Fig. 3.4 has 0 in the row we are currently considering. The resulting signature matrix:

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

Finally, consider the row of Fig. 3.4 numbered 4. $h_1(4) = 0$ and $h_2(4) = 3$. Since row 4 has 1 only in the column for S_3, we only compare the current signature column for that set, $[2, 0]$ with the hash values $[0, 3]$. Since $0 < 2$, we change $\text{SIG}(1,3)$ to 0, but since $3 > 0$ we do not change $\text{SIG}(2,3)$. The final signature matrix is:

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

We can estimate the Jaccard similarities of the underlying sets from this signature matrix. Notice that columns 1 and 4 are identical, so we guess that $\text{SIM}(S_1, S_4) = 1.0$. If we look at Fig. 3.4, we see that the true Jaccard similarity of S_1 and S_4 is 2/3. Remember that the fraction of rows that agree in the signature matrix is only an estimate of the true Jaccard similarity, and this example is much too small for the law of large numbers to assure that the estimates are close. For additional examples, the signature columns for S_1 and S_3 agree in half the rows (true similarity 1/4), while the signatures of S_1 and S_2 estimate 0 as their Jaccard similarity (the correct value). □

3.3.6 Exercises for Section 3.3

Exercise 3.3.1: Verify the theorem from Section 3.3.3, which relates the Jaccard similarity to the probability of minhashing to equal values, for the particular case of Fig. 3.2.

(a) Compute the Jaccard similarity of each of the pairs of columns in Fig. 3.2.

! (b) Compute, for each pair of columns of that figure, the fraction of the 120 permutations of the rows that make the two columns hash to the same value.

Exercise 3.3.2: Using the data from Fig. 3.4, add to the signatures of the columns the values of the following hash functions:

(a) $h_3(x) = 2x + 4 \mod 5$.

(b) $h_4(x) = 3x - 1 \mod 5$.

Element	S_1	S_2	S_3	S_4
0	0	1	0	1
1	0	1	0	0
2	1	0	0	1
3	0	0	1	0
4	0	0	1	1
5	1	0	0	0

Figure 3.5: Matrix for Exercise 3.3.3

Exercise 3.3.3: In Fig. 3.5 is a matrix with six rows.

(a) Compute the minhash signature for each column if we use the following three hash functions: $h_1(x) = 2x + 1 \mod 6$; $h_2(x) = 3x + 2 \mod 6$; $h_3(x) = 5x + 2 \mod 6$.

(b) Which of these hash functions are true permutations?

(c) How close are the estimated Jaccard similarities for the six pairs of columns to the true Jaccard similarities?

! **Exercise 3.3.4 :** Now that we know Jaccard similarity is related to the probability that two sets minhash to the same value, reconsider Exercise 3.1.3. Can you use this relationship to simplify the problem of computing the expected Jaccard similarity of randomly chosen sets?

! **Exercise 3.3.5 :** Prove that if the Jaccard similarity of two columns is 0, then minhashing always gives a correct estimate of the Jaccard similarity.

!! **Exercise 3.3.6 :** One might expect that we could estimate the Jaccard similarity of columns without using all possible permutations of rows. For example, we could only allow cyclic permutations; i.e., start at a randomly chosen row r, which becomes the first in the order, followed by rows $r+1$, $r+2$, and so on, down to the last row, and then continuing with the first row, second row, and so on, down to row $r-1$. There are only n such permutations if there are n rows. However, these permutations are not sufficient to estimate the Jaccard similarity correctly. Give an example of a two-column matrix where averaging over all the cyclic permutations does not give the Jaccard similarity.

! **Exercise 3.3.7 :** Suppose we want to use a MapReduce framework to compute minhash signatures. If the matrix is stored in chunks that correspond to some columns, then it is quite easy to exploit parallelism. Each Map task gets some of the columns and all the hash functions, and computes the minhash signatures of its given columns. However, suppose the matrix were chunked by rows, so that a Map task is given the hash functions and a set of rows to work on. Design Map and Reduce functions to exploit MapReduce with data in this form.

3.4 Locality-Sensitive Hashing for Documents

Even though we can use minhashing to compress large documents into small signatures and preserve the expected similarity of any pair of documents, it still may be impossible to find the pairs with greatest similarity efficiently. The reason is that the number of pairs of documents may be too large, even if there are not too many documents.

Example 3.9 : Suppose we have a million documents, and we use signatures of length 250. Then we use 1000 bytes per document for the signatures, and the entire data fits in a gigabyte – less than a typical main memory of a laptop. However, there are $\binom{1,000,000}{2}$ or half a trillion pairs of documents. If it takes a microsecond to compute the similarity of two signatures, then it takes almost six days to compute all the similarities on that laptop. □

If our goal is to compute the similarity of every pair, there is nothing we can do to reduce the work, although parallelism can reduce the elapsed time. However, often we want only the most similar pairs or all pairs that are above some lower bound in similarity. If so, then we need to focus our attention only on pairs that are likely to be similar, without investigating every pair. There is a general theory of how to provide such focus, called *locality-sensitive hashing* (LSH) or *near-neighbor search*. In this section we shall consider a specific form of LSH, designed for the particular problem we have been studying: documents, represented by shingle-sets, then minhashed to short signatures. In Section 3.6 we present the general theory of locality-sensitive hashing and a number of applications and related techniques.

3.4.1 LSH for Minhash Signatures

One general approach to LSH is to "hash" items several times, in such a way that similar items are more likely to be hashed to the same bucket than dissimilar items are. We then consider any pair that hashed to the same bucket for any of the hashings to be a *candidate pair*. We check only the candidate pairs for similarity. The hope is that most of the dissimilar pairs will never hash to the same bucket, and therefore will never be checked. Those dissimilar pairs that do hash to the same bucket are *false positives*; we hope these will be only a small fraction of all pairs. We also hope that most of the truly similar pairs will hash to the same bucket under at least one of the hash functions. Those that do not are *false negatives*; we hope these will be only a small fraction of the truly similar pairs.

If we have minhash signatures for the items, an effective way to choose the hashings is to divide the signature matrix into b bands consisting of r rows each. For each band, there is a hash function that takes vectors of r integers (the portion of one column within that band) and hashes them to some large number of buckets. We can use the same hash function for all the bands, but we use a separate bucket array for each band, so columns with the same vector in different bands will not hash to the same bucket.

Example 3.10 : Figure 3.6 shows part of a signature matrix of 12 rows divided into four bands of three rows each. The second and fourth of the explicitly shown columns each have the column vector $[0, 2, 1]$ in the first band, so they will definitely hash to the same bucket in the hashing for the first band. Thus, regardless of what those columns look like in the other three bands, this pair of columns will be a candidate pair. It is possible that other columns, such as the first two shown explicitly, will also hash to the same bucket according to the hashing of the first band. However, since their column vectors are different, $[1, 3, 0]$ and $[0, 2, 1]$, and there are many buckets for each hashing, we expect the chances of an accidental collision to be very small. We shall normally assume that two vectors hash to the same bucket if and only if they are identical.

Two columns that do not agree in band 1 have three other chances to become a candidate pair; they might be identical in any one of these other bands.

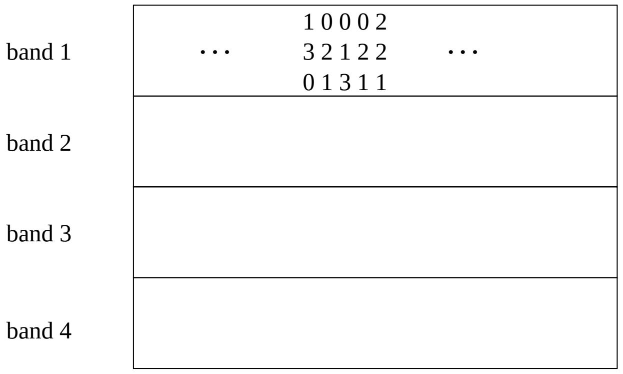


Figure 3.6: Dividing a signature matrix into four bands of three rows per band

However, observe that the more similar two columns are, the more likely it is that they will be identical in some band. Thus, intuitively the banding strategy makes similar columns much more likely to be candidate pairs than dissimilar pairs. □

3.4.2 Analysis of the Banding Technique

Suppose we use b bands of r rows each, and suppose that a particular pair of documents have Jaccard similarity s. Recall from Section 3.3.3 that the probability the minhash signatures for these documents agree in any one particular row of the signature matrix is s. We can calculate the probability that these documents (or rather their signatures) become a candidate pair as follows:

1. The probability that the signatures agree in all rows of one particular band is s^r.

2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$.

3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$.

4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

It may not be obvious, but regardless of the chosen constants b and r, this function has the form of an *S-curve*, as suggested in Fig. 3.7. The *threshold*, that is, the value of similarity s at which the probability of becoming a candidate is $1/2$, is a function of b and r. The threshold is roughly where the rise is the steepest, and for large b and r there we find that pairs with similarity above the threshold are very likely to become candidates, while those below the threshold are unlikely to become candidates – exactly the situation we want.

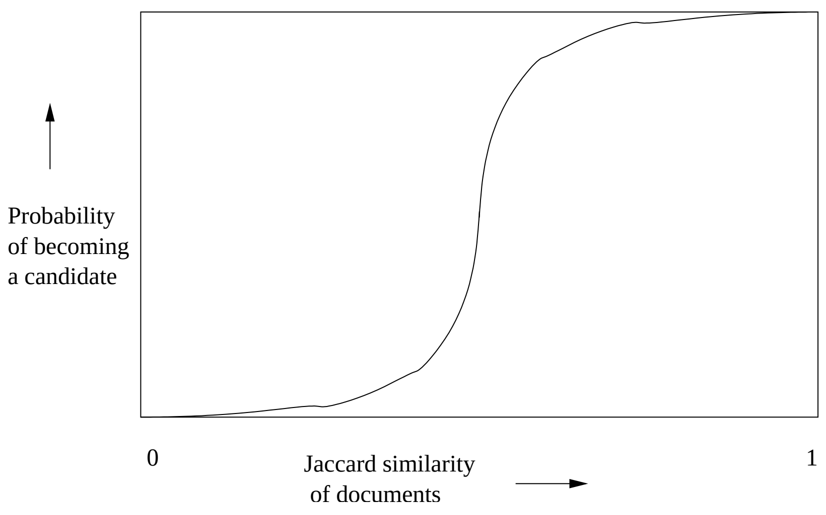


Figure 3.7: The S-curve

An approximation to the threshold is $(1/b)^{1/r}$. For example, if $b = 16$ and $r = 4$, then the threshold is approximately at $s = 1/2$, since the 4th root of 1/16 is 1/2.

Example 3.11 : Let us consider the case $b = 20$ and $r = 5$. That is, we suppose we have signatures of length 100, divided into twenty bands of five rows each. Figure 3.8 tabulates some of the values of the function $1 - (1 - s^5)^{20}$. Notice that the threshold, the value of s at which the curve has risen halfway, is just slightly more than 0.5. Also notice that the curve is not exactly the ideal step function that jumps from 0 to 1 at the threshold, but the slope of the curve in the middle is significant. For example, it rises by more than 0.6 going from $s = 0.4$ to $s = 0.6$, so the slope in the middle is greater than 3.

s	$1 - (1 - s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

Figure 3.8: Values of the S-curve for $b = 20$ and $r = 5$

For example, at $s = 0.8$, $1 - (0.8)^5$ is about 0.672. If you raise this number to the 20th power, you get about 0.00035. Subtracting this fraction from 1 yields 0.99965. That is, if we consider two documents with 80% similarity, then in any one band, they have only about a 33% chance of agreeing in all five rows and thus becoming a candidate pair. However, there are 20 bands and thus 20 chances to become a candidate. Only roughly one in 3000 pairs that are as high as 80% similar will fail to become a candidate pair and thus be a false negative. □

3.4.3 Combining the Techniques

We can now give an approach to finding the set of candidate pairs for similar documents and then discovering the truly similar documents among them. It must be emphasized that this approach can produce false negatives – pairs of similar documents that are not identified as such because they never become a candidate pair. There will also be false positives – candidate pairs that are evaluated, but are found not to be sufficiently similar.

1. Pick a value of k and construct from each document the set of k-shingles. Optionally, hash the k-shingles to shorter bucket numbers.

2. Sort the document-shingle pairs to order them by shingle.

3. Pick a length n for the minhash signatures. Feed the sorted list to the algorithm of Section 3.3.5 to compute the minhash signatures for all the documents.

4. Choose a threshold t that defines how similar documents have to be in order for them to be regarded as a desired "similar pair." Pick a number of bands b and a number of rows r such that $br = n$, and the threshold t is approximately $(1/b)^{1/r}$. If avoidance of false negatives is important, you may wish to select b and r to produce a threshold lower than t; if speed is important and you wish to limit false positives, select b and r to produce a higher threshold.

5. Construct candidate pairs by applying the LSH technique of Section 3.4.1.

6. Examine each candidate pair's signatures and determine whether the fraction of components in which they agree is at least t.

7. Optionally, if the signatures are sufficiently similar, go to the documents themselves and check that they are truly similar, rather than documents that, by luck, had similar signatures.

3.4.4 Exercises for Section 3.4

Exercise 3.4.1: Evaluate the S-curve $1-(1-s^r)^b$ for $s = 0.1, 0.2, \ldots, 0.9$, for the following values of r and b:

- $r = 3$ and $b = 10$.

- $r = 6$ and $b = 20$.
- $r = 5$ and $b = 50$.

! **Exercise 3.4.2:** For each of the (r, b) pairs in Exercise 3.4.1, compute the threshold, that is, the value of s for which the value of $1-(1-s^r)^b$ is exactly 1/2. How does this value compare with the estimate of $(1/b)^{1/r}$ that was suggested in Section 3.4.2?

! **Exercise 3.4.3:** Use the techniques explained in Section 1.3.5 to approximate the S-curve $1 - (1 - s^r)^b$ when s^r is very small.

! **Exercise 3.4.4:** Suppose we wish to implement LSH by MapReduce. Specifically, assume chunks of the signature matrix consist of columns, and elements are key-value pairs where the key is the column number and the value is the signature itself (i.e., a vector of values).

(a) Show how to produce the buckets for all the bands as output of a single MapReduce process. *Hint*: Remember that a Map function can produce several key-value pairs from a single element.

(b) Show how another MapReduce process can convert the output of (a) to a list of pairs that need to be compared. Specifically, for each column i, there should be a list of those columns $j > i$ with which i needs to be compared.

3.5 Distance Measures

We now take a short detour to study the general notion of distance measures. The Jaccard similarity is a measure of how close sets are, although it is not really a distance measure. That is, the closer sets are, the higher the Jaccard similarity. Rather, 1 minus the Jaccard similarity is a distance measure, as we shall see; it is called the *Jaccard distance*.

However, Jaccard distance is not the only measure of closeness that makes sense. We shall examine in this section some other distance measures that have applications. Then, in Section 3.6 we see how some of these distance measures also have an LSH technique that allows us to focus on nearby points without comparing all points. Other applications of distance measures will appear when we study clustering in Chapter 7.

3.5.1 Definition of a Distance Measure

Suppose we have a set of points, called a *space*. A *distance measure* on this space is a function $d(x, y)$ that takes two points in the space as arguments and produces a real number, and satisfies the following axioms:

1. $d(x, y) \geq 0$ (no negative distances).

2. $d(x, y) = 0$ if and only if $x = y$ (distances are positive, except for the distance from a point to itself).

3. $d(x, y) = d(y, x)$ (distance is symmetric).

4. $d(x, y) \leq d(x, z) + d(z, y)$ (the *triangle inequality*).

The triangle inequality is the most complex condition. It says, intuitively, that to travel from x to y, we cannot obtain any benefit if we are forced to travel via some particular third point z. The triangle-inequality axiom is what makes all distance measures behave as if distance describes the length of a shortest path from one point to another.

3.5.2 Euclidean Distances

The most familiar distance measure is the one we normally think of as "distance." An *n-dimensional Euclidean space* is one where points are vectors of n real numbers. The conventional distance measure in this space, which we shall refer to as the L_2*-norm*, is defined:

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

That is, we square the distance in each dimension, sum the squares, and take the positive square root.

It is easy to verify the first three requirements for a distance measure are satisfied. The Euclidean distance between two points cannot be negative, because the positive square root is intended. Since all squares of real numbers are nonnegative, any i such that $x_i \neq y_i$ forces the distance to be strictly positive. On the other hand, if $x_i = y_i$ for all i, then the distance is clearly 0. Symmetry follows because $(x_i - y_i)^2 = (y_i - x_i)^2$. The triangle inequality requires a good deal of algebra to verify. However, it is well understood to be a property of Euclidean space: the sum of the lengths of any two sides of a triangle is no less than the length of the third side.

There are other distance measures that have been used for Euclidean spaces. For any constant r, we can define the L_r*-norm* to be the distance measure d defined by:

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = (\sum_{i=1}^{n} |x_i - y_i|^r)^{1/r}$$

The case $r = 2$ is the usual L_2-norm just mentioned. Another common distance measure is the L_1-norm, or *Manhattan distance*. There, the distance between two points is the sum of the magnitudes of the differences in each dimension. It is called "Manhattan distance" because it is the distance one would have to travel between points if one were constrained to travel along grid lines, as on the streets of a city such as Manhattan.

Another interesting distance measure is the L_∞-norm, which is the limit as r approaches infinity of the L_r-norm. As r gets larger, only the dimension with the largest difference matters, so formally, the L_∞-norm is defined as the maximum of $|x_i - y_i|$ over all dimensions i.

Example 3.12 : Consider the two-dimensional Euclidean space (the customary plane) and the points $(2, 7)$ and $(6, 4)$. The L_2-norm gives a distance of $\sqrt{(2-6)^2 + (7-4)^2} = \sqrt{4^2 + 3^2} = 5$. The L_1-norm gives a distance of $|2 - 6| + |7 - 4| = 4 + 3 = 7$. The L_∞-norm gives a distance of

$$\max(|2 - 6|, |7 - 4|) = \max(4, 3) = 4$$

□

3.5.3 Jaccard Distance

As mentioned at the beginning of the section, we define the *Jaccard distance* of sets by $d(x, y) = 1 - \text{SIM}(x, y)$. That is, the Jaccard distance is 1 minus the ratio of the sizes of the intersection and union of sets x and y. We must verify that this function is a distance measure.

1. $d(x, y)$ is nonnegative because the size of the intersection cannot exceed the size of the union.

2. $d(x, y) = 0$ if $x = y$, because $x \cup x = x \cap x = x$. However, if $x \neq y$, then the size of $x \cap y$ is strictly less than the size of $x \cup y$, so $d(x, y)$ is strictly positive.

3. $d(x, y) = d(y, x)$ because both union and intersection are symmetric; i.e., $x \cup y = y \cup x$ and $x \cap y = y \cap x$.

4. For the triangle inequality, recall from Section 3.3.3 that $\text{SIM}(x, y)$ is the probability a random minhash function maps x and y to the same value. Thus, the Jaccard distance $d(x, y)$ is the probability that a random minhash function *does not* send x and y to the same value. We can therefore translate the condition $d(x, y) \leq d(x, z) + d(z, y)$ to the statement that if h is a random minhash function, then the probability that $h(x) \neq h(y)$ is no greater than the sum of the probability that $h(x) \neq h(z)$ and the probability that $h(z) \neq h(y)$. However, this statement is true because whenever $h(x) \neq h(y)$, at least one of $h(x)$ and $h(y)$ must be different from $h(z)$. They could not both be $h(z)$, because then $h(x)$ and $h(y)$ would be the same.

.

3.5.4 Cosine Distance

The *cosine distance* makes sense in spaces that have dimensions, including Euclidean spaces and discrete versions of Euclidean spaces, such as spaces where points are vectors with integer components or boolean (0 or 1) components. In such a space, points may be thought of as directions. We do not distinguish between a vector and a multiple of that vector. Then the cosine distance between two points is the angle that the vectors to those points make. This angle will be in the range 0 to 180 degrees, regardless of how many dimensions the space has.

We can calculate the cosine distance by first computing the cosine of the angle, and then applying the arc-cosine function to translate to an angle in the 0-180 degree range. Given two vectors x and y, the cosine of the angle between them is the dot product $x.y$ divided by the L_2-norms of x and y (i.e., their Euclidean distances from the origin). Recall that the dot product of vectors $[x_1, x_2, \ldots, x_n].[y_1, y_2, \ldots, y_n]$ is $\sum_{i=1}^{n} x_i y_i$.

Example 3.13 : Let our two vectors be $x = [1, 2, -1]$ and $= [2, 1, 1]$. The dot product $x.y$ is $1 \times 2 + 2 \times 1 + (-1) \times 1 = 3$. The L_2-norm of both vectors is $\sqrt{6}$. For example, x has L_2-norm $\sqrt{1^2 + 2^2 + (-1)^2} = \sqrt{6}$. Thus, the cosine of the angle between x and y is $3/(\sqrt{6}\sqrt{6})$ or $1/2$. The angle whose cosine is $1/2$ is 60 degrees, so that is the cosine distance between x and y. □

We must show that the cosine distance is indeed a distance measure. We have defined it so the values are in the range 0 to 180, so no negative distances are possible. Two vectors have angle 0 if and only if they are the same direction.[4] Symmetry is obvious: the angle between x and y is the same as the angle between y and x. The triangle inequality is best argued by physical reasoning. One way to rotate from x to y is to rotate to z and thence to y. The sum of those two rotations cannot be less than the rotation directly from x to y.

3.5.5 Edit Distance

This distance makes sense when points are strings. The distance between two strings $x = x_1 x_2 \cdots x_n$ and $y = y_1 y_2 \cdots y_m$ is the smallest number of insertions and deletions of single characters that will convert x to y.

Example 3.14 : The edit distance between the strings $x =$ `abcde` and $y =$ `acfdeg` is 3. To convert x to y:

1. Delete `b`.

2. Insert `f` after `c`.

[4]Notice that to satisfy the second axiom, we have to treat vectors that are multiples of one another, e.g. $[1, 2]$ and $[3, 6]$, as the same direction, which they are. If we regarded these as different vectors, we would give them distance 0 and thus violate the condition that only $d(x, x)$ is 0.

3. Insert `g` after `e`.

No sequence of fewer than three insertions and/or deletions will convert x to y. Thus, $d(x, y) = 3$. □

Another way to define and calculate the edit distance $d(x, y)$ is to compute a *longest common subsequence* (LCS) of x and y. An LCS of x and y is a string that is constructed by deleting positions from x and y, and that is as long as any string that can be constructed that way. The edit distance $d(x, y)$ can be calculated as the length of x plus the length of y minus twice the length of their LCS.

Example 3.15 : The strings $x =$ `abcde` and $y =$ `acfdeg` from Example 3.14 have a unique LCS, which is `acde`. We can be sure it is the longest possible, because it contains every symbol appearing in both x and y. Fortunately, these common symbols appear in the same order in both strings, so we are able to use them all in an LCS. Note that the length of x is 5, the length of y is 6, and the length of their LCS is 4. The edit distance is thus $5 + 6 - 2 \times 4 = 3$, which agrees with the direct calculation in Example 3.14.

For another example, consider $x =$ `aba` and $y =$ `bab`. Their edit distance is 2. For example, we can convert x to y by deleting the first `a` and then inserting `b` at the end. There are two LCS's: `ab` and `ba`. Each can be obtained by deleting one symbol from each string. As must be the case for multiple LCS's of the same pair of strings, both LCS's have the same length. Therefore, we may compute the edit distance as $3 + 3 - 2 \times 2 = 2$. □

Edit distance is a distance measure. Surely no edit distance can be negative, and only two identical strings have an edit distance of 0. To see that edit distance is symmetric, note that a sequence of insertions and deletions can be reversed, with each insertion becoming a deletion, and vice versa. The triangle inequality is also straightforward. One way to turn a string s into a string t is to turn s into some string u and then turn u into t. Thus, the number of edits made going from s to u, plus the number of edits made going from u to t cannot be less than the smallest number of edits that will turn s into t.

3.5.6 Hamming Distance

Given a space of vectors, we define the *Hamming distance* between two vectors to be the number of components in which they differ. It should be obvious that Hamming distance is a distance measure. Clearly the Hamming distance cannot be negative, and if it is zero, then the vectors are identical. The distance does not depend on which of two vectors we consider first. The triangle inequality should also be evident. If x and z differ in m components, and z and y differ in n components, then x and y cannot differ in more than $m+n$ components. Most commonly, Hamming distance is used when the vectors are boolean; they consist of 0's and 1's only. However, in principle, the vectors can have components from any set.

Non-Euclidean Spaces

Notice that several of the distance measures introduced in this section are not Euclidean spaces. A property of Euclidean spaces that we shall find important when we take up clustering in Chapter 7 is that the average of points in a Euclidean space always exists and is a point in the space. However, consider the space of sets for which we defined the Jaccard distance. The notion of the "average" of two sets makes no sense. Likewise, the space of strings, where we can use the edit distance, does not let us take the "average" of strings.

Vector spaces, for which we suggested the cosine distance, may or may not be Euclidean. If the components of the vectors can be any real numbers, then the space is Euclidean. However, if we restrict components to be integers, then the space is not Euclidean. Notice that, for instance, we cannot find an average of the vectors $[1, 2]$ and $[3, 1]$ in the space of vectors with two integer components, although if we treated them as members of the two-dimensional Euclidean space, then we could say that their average was $[2.0, 1.5]$.

Example 3.16 : The Hamming distance between the vectors 10101 and 11110 is 3. That is, these vectors differ in the second, fourth, and fifth components, while they agree in the first and third components. □

3.5.7 Exercises for Section 3.5

! **Exercise 3.5.1 :** On the space of nonnegative integers, which of the following functions are distance measures? If so, prove it; if not, prove that it fails to satisfy one or more of the axioms.

(a) $\max(x, y) =$ the larger of x and y.

(b) $\text{diff}(x, y) = |x - y|$ (the absolute magnitude of the difference between x and y).

(c) $\text{sum}(x, y) = x + y$.

Exercise 3.5.2 : Find the L_1 and L_2 distances between the points $(5, 6, 7)$ and $(8, 2, 4)$.

!! **Exercise 3.5.3 :** Prove that if i and j are any positive integers, and $i < j$, then the L_i norm between any two points is greater than the L_j norm between those same two points.

Exercise 3.5.4 : Find the Jaccard distances between the following pairs of sets:

(a) $\{1, 2, 3, 4\}$ and $\{2, 3, 4, 5\}$.

(b) $\{1, 2, 3\}$ and $\{4, 5, 6\}$.

Exercise 3.5.5: Compute the cosines of the angles between each of the following pairs of vectors.[5]

(a) $(3, -1, 2)$ and $(-2, 3, 1)$.

(b) $(1, 2, 3)$ and $(2, 4, 6)$.

(c) $(5, 0, -4)$ and $(-1, -6, 2)$.

(d) $(0, 1, 1, 0, 1, 1)$ and $(0, 0, 1, 0, 0, 0)$.

! **Exercise 3.5.6:** Prove that the cosine distance between any two vectors of 0's and 1's, of the same length, is at most 90 degrees.

Exercise 3.5.7: Find the edit distances (using only insertions and deletions) between the following pairs of strings.

(a) `abcdef` and `bdaefc`.

(b) `abccdabc` and `acbdcab`.

(c) `abcdef` and `baedfc`.

! **Exercise 3.5.8:** There are a number of other notions of edit distance available. For instance, we can allow, in addition to insertions and deletions, the following operations:

i. *Mutation*, where one symbol is replaced by another symbol. Note that a mutation can always be performed by an insertion followed by a deletion, but if we allow mutations, then this change counts for only 1, not 2, when computing the edit distance.

ii. *Transposition*, where two adjacent symbols have their positions swapped. Like a mutation, we can simulate a transposition by one insertion followed by one deletion, but here we count only 1 for these two steps.

Repeat Exercise 3.5.7 if edit distance is defined to be the number of insertions, deletions, mutations, and transpositions needed to transform one string into another.

! **Exercise 3.5.9:** Prove that the edit distance discussed in Exercise 3.5.8 is indeed a distance measure.

Exercise 3.5.10: Find the Hamming distances between each pair of the following vectors: 000000, 110011, 010101, and 011100.

[5]Note that what we are asking for is not precisely the cosine distance, but from the cosine of an angle, you can compute the angle itself, perhaps with the aid of a table or library function.

3.6 The Theory of Locality-Sensitive Functions

The LSH technique developed in Section 3.4 is one example of a family of functions (the minhash functions) that can be combined (by the banding technique) to distinguish strongly between pairs at a low distance from pairs at a high distance. The steepness of the S-curve in Fig. 3.7 reflects how effectively we can avoid false positives and false negatives among the candidate pairs.

Now, we shall explore other families of functions, besides the minhash functions, that can serve to produce candidate pairs efficiently. These functions can apply to the space of sets and the Jaccard distance, or to another space and/or another distance measure. There are three conditions that we need for a family of functions:

1. They must be more likely to make close pairs be candidate pairs than distant pairs. We make this notion precise in Section 3.6.1.

2. They must be statistically independent, in the sense that it is possible to estimate the probability that two or more functions will all give a certain response by the product rule for independent events.

3. They must be efficient, in two ways:

   (a) They must be able to identify candidate pairs in time much less than the time it takes to look at all pairs. For example, minhash functions have this capability, since we can hash sets to minhash values in time proportional to the size of the data, rather than the square of the number of sets in the data. Since sets with common values are colocated in a bucket, we have implicitly produced the candidate pairs for a single minhash function in time much less than the number of pairs of sets.

   (b) They must be combinable to build functions that are better at avoiding false positives and negatives, and the combined functions must also take time that is much less than the number of pairs. For example, the banding technique of Section 3.4.1 takes single minhash functions, which satisfy condition 3a but do not, by themselves have the S-curve behavior we want, and produces from a number of minhash functions a combined function that has the S-curve shape.

Our first step is to define "locality-sensitive functions" generally. We then see how the idea can be applied in several applications. Finally, we discuss how to apply the theory to arbitrary data with either a cosine distance or a Euclidean distance measure.

3.6.1 Locality-Sensitive Functions

For the purposes of this section, we shall consider functions that take two items and render a decision about whether these items should be a candidate pair.

In many cases, the function f will "hash" items, and the decision will be based on whether or not the result is equal. Because it is convenient to use the notation $f(x) = f(y)$ to mean that $f(x, y)$ is "yes; make x and y a candidate pair," we shall use $f(x) = f(y)$ as a shorthand with this meaning. We also use $f(x) \neq f(y)$ to mean "do not make x and y a candidate pair unless some other function concludes we should do so."

A collection of functions of this form will be called a *family* of functions. For example, the family of minhash functions, each based on one of the possible permutations of rows of a characteristic matrix, form a family.

Let $d_1 < d_2$ be two distances according to some distance measure d. A family $\mathbf{F}$ of functions is said to be (d_1, d_2, p_1, p_2)*-sensitive* if for every f in $\mathbf{F}$:

1. If $d(x, y) \leq d_1$, then the probability that $f(x) = f(y)$ is at least p_1.
2. If $d(x, y) \geq d_2$, then the probability that $f(x) = f(y)$ is at most p_2.

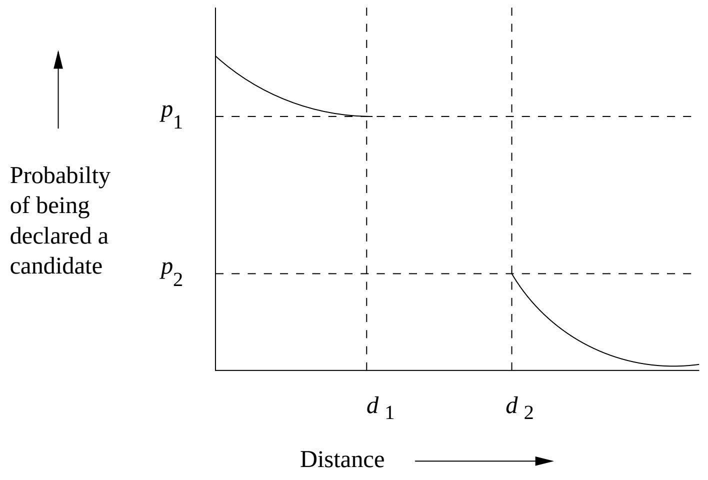


Figure 3.9: Behavior of a (d_1, d_2, p_1, p_2)-sensitive function

Figure 3.9 illustrates what we expect about the probability that a given function in a (d_1, d_2, p_1, p_2)-sensitive family will declare two items to be a candidate pair. Notice that we say nothing about what happens when the distance between the items is strictly between d_1 and d_2, but we can make d_1 and d_2 as close as we wish. The penalty is that typically p_1 and p_2 are then close as well. As we shall see, it is possible to drive p_1 and p_2 apart while keeping d_1 and d_2 fixed.

3.6.2 Locality-Sensitive Families for Jaccard Distance

For the moment, we have only one way to find a family of locality-sensitive functions: use the family of minhash functions, and assume that the distance measure is the Jaccard distance. As before, we interpret a minhash function h to make x and y a candidate pair if and only if $h(x) = h(y)$.

- The family of minhash functions is a $(d_1, d_2, 1-d_1, 1-d_2)$-sensitive family for any d_1 and d_2, where $0 \leq d_1 < d_2 \leq 1$.

The reason is that if $d(x, y) \leq d_1$, where d is the Jaccard distance, then $\text{SIM}(x, y) = 1 - d(x, y) \geq 1 - d_1$. But we know that the Jaccard similarity of x and y is equal to the probability that a minhash function will hash x and y to the same value. A similar argument applies to d_2 or any distance.

Example 3.17 : We could let $d_1 = 0.3$ and $d_2 = 0.6$. Then we can assert that the family of minhash functions is a $(0.3, 0.6, 0.7, 0.4)$-sensitive family. That is, if the Jaccard distance between x and y is at most 0.3 (i.e., $\text{SIM}(x, y) \geq 0.7$) then there is at least a 0.7 chance that a minhash function will send x and y to the same value, and if the Jaccard distance between x and y is at least 0.6 (i.e., $\text{SIM}(x, y) \leq 0.4$), then there is at most a 0.4 chance that x and y will be sent to the same value. Note that we could make the same assertion with another choice of d_1 and d_2; only $d_1 < d_2$ is required. □

3.6.3 Amplifying a Locality-Sensitive Family

Suppose we are given a (d_1, d_2, p_1, p_2)-sensitive family $\mathbf{F}$. We can construct a new family $\mathbf{F}'$ by the *AND-construction* on $\mathbf{F}$, which is defined as follows. Each member of $\mathbf{F}'$ consists of r members of $\mathbf{F}$ for some fixed r. If f is in $\mathbf{F}'$, and f is constructed from the set $\{f_1, f_2, \ldots, f_r\}$ of members of $\mathbf{F}$, we say $f(x) = f(y)$ if and only if $f_i(x) = f_i(y)$ for all $i = 1, 2, \ldots, r$. Notice that this construction mirrors the effect of the r rows in a single band: the band makes x and y a candidate pair if every one of the r rows in the band say that x and y are equal (and therefore a candidate pair according to that row).

Since the members of $\mathbf{F}$ are independently chosen to make a member of $\mathbf{F}'$, we can assert that $\mathbf{F}'$ is a $\big(d_1, d_2, (p_1)^r, (p_2)^r\big)$-sensitive family. That is, for any p, if p is the probability that a member of $\mathbf{F}$ will declare (x, y) to be a candidate pair, then the probability that a member of $\mathbf{F}'$ will so declare is p^r.

There is another construction, which we call the *OR-construction*, that turns a (d_1, d_2, p_1, p_2)-sensitive family $\mathbf{F}$ into a $\big(d_1, d_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b\big)$-sensitive family $\mathbf{F}'$. Each member f of $\mathbf{F}'$ is constructed from b members of $\mathbf{F}$, say $f_1, f_2, \ldots, f_b$. We define $f(x) = f(y)$ if and only if $f_i(x) = f_i(y)$ for one or more values of i. The OR-construction mirrors the effect of combining several bands: x and y become a candidate pair if any band makes them a candidate pair.

If p is the probability that a member of $\mathbf{F}$ will declare (x, y) to be a candidate pair, then $1-p$ is the probability it will not so declare. $(1-p)^b$ is the probability that none of $f_1, f_2, \ldots, f_b$ will declare (x, y) a candidate pair, and $1 - (1 - p)^b$ is the probability that at least one f_i will declare (x, y) a candidate pair, and therefore that f will declare (x, y) to be a candidate pair.

Notice that the AND-construction lowers all probabilities, but if we choose **F** and r judiciously, we can make the small probability p_2 get very close to 0, while the higher probability p_1 stays significantly away from 0. Similarly, the OR-construction makes all probabilities rise, but by choosing **F** and b judiciously, we can make the larger probability approach 1 while the smaller probability remains bounded away from 1. We can cascade AND- and OR-constructions in any order to make the low probability close to 0 and the high probability close to 1. Of course the more constructions we use, and the higher the values of r and b that we pick, the larger the number of functions from the original family that we are forced to use. Thus, the better the final family of functions is, the longer it takes to apply the functions from this family.

Example 3.18: Suppose we start with a family **F**. We use the AND-construction with $r = 4$ to produce a family $\mathbf{F}_1$. We then apply the OR-construction to $\mathbf{F}_1$ with $b = 4$ to produce a third family $\mathbf{F}_2$. Note that the members of $\mathbf{F}_2$ each are built from 16 members of **F**, and the situation is analogous to starting with 16 minhash functions and treating them as four bands of four rows each.

p	$1-(1-p^4)^4$
0.2	0.0064
0.3	0.0320
0.4	0.0985
0.5	0.2275
0.6	0.4260
0.7	0.6666
0.8	0.8785
0.9	0.9860

Figure 3.10: Effect of the 4-way AND-construction followed by the 4-way OR-construction

The 4-way AND-function converts any probability p into p^4. When we follow it by the 4-way OR-construction, that probability is further converted into $1-(1-p^4)^4$. Some values of this transformation are indicated in Fig. 3.10. This function is an S-curve, staying low for a while, then rising steeply (although not too steeply; the slope never gets much higher than 2), and then leveling off at high values. Like any S-curve, it has a *fixedpoint*, the value of p that is left unchanged when we apply the function of the S-curve. In this case, the fixedpoint is the value of p for which $p = 1-(1-p^4)^4$. We can see that the fixedpoint is somewhere between 0.7 and 0.8. Below that value, probabilities are decreased, and above it they are increased. Thus, if we pick a high probability above the fixedpoint and a low probability below it, we shall have the desired effect that the low probability is decreased and the high probability is increased.

Suppose **F** is the minhash functions, regarded as a $(0.2, 0.6, 0.8, 0.4)$-sensitive family. Then $\mathbf{F}_2$, the family constructed by a 4-way AND followed by a

4-way OR, is a $(0.2, 0.6, 0.8785, 0.0985)$-sensitive family, as we can read from the rows for 0.8 and 0.4 in Fig. 3.10. By replacing $\mathbf{F}$ by $\mathbf{F}_2$, we have reduced both the false-negative and false-positive rates, at the cost of making application of the functions take 16 times as long. □

p	$\bigl(1-(1-p)^4\bigr)^4$
0.1	0.0140
0.2	0.1215
0.3	0.3334
0.4	0.5740
0.5	0.7725
0.6	0.9015
0.7	0.9680
0.8	0.9936

Figure 3.11: Effect of the 4-way OR-construction followed by the 4-way AND-construction

Example 3.19 : For the same cost, we can apply a 4-way OR-construction followed by a 4-way AND-construction. Figure 3.11 gives the transformation on probabilities implied by this construction. For instance, suppose that $\mathbf{F}$ is a $(0.2, 0.6, 0.8, 0.4)$-sensitive family. Then the constructed family is a

$$(0.2, 0.6, 0.9936, 0.5740)\text{-sensitive}$$

family. This choice is not necessarily the best. Although the higher probability has moved much closer to 1, the lower probability has also raised, increasing the number of false positives. □

Example 3.20 : We can cascade constructions as much as we like. For example, we could use the construction of Example 3.18 on the family of minhash functions and then use the construction of Example 3.19 on the resulting family. The constructed family would then have functions each built from 256 minhash functions. It would, for instance transform a $(0.2, 0.8, 0.8, 0.2)$-sensitive family into a $(0.2, 0.8, 0.9991285, 0.0000004)$-sensitive family. □

3.6.4 Exercises for Section 3.6

Exercise 3.6.1 : What is the effect on probability of starting with the family of minhash functions and applying:

(a) A 2-way AND construction followed by a 3-way OR construction.

(b) A 3-way OR construction followed by a 2-way AND construction.

(c) A 2-way AND construction followed by a 2-way OR construction, followed by a 2-way AND construction.

(d) A 2-way OR construction followed by a 2-way AND construction, followed by a 2-way OR construction followed by a 2-way AND construction.

Exercise 3.6.2: Find the fixedpoints for each of the functions constructed in Exercise 3.6.1.

! **Exercise 3.6.3:** Any function of probability p, such as that of Fig. 3.10, has a slope given by the derivative of the function. The maximum slope is where that derivative is a maximum. Find the value of p that gives a maximum slope for the S-curves given by Fig. 3.10 and Fig. 3.11. What are the values of these maximum slopes?

!! **Exercise 3.6.4:** Generalize Exercise 3.6.3 to give, as a function of r and b, the point of maximum slope and the value of that slope, for families of functions defined from the minhash functions by:

(a) An r-way AND construction followed by a b-way OR construction.

(b) A b-way OR construction followed by an r-way AND construction.

3.7 LSH Families for Other Distance Measures

There is no guarantee that a distance measure has a locality-sensitive family of hash functions. So far, we have only seen such families for the Jaccard distance. In this section, we shall show how to construct locality-sensitive families for Hamming distance, the cosine distance and for the normal Euclidean distance.

3.7.1 LSH Families for Hamming Distance

It is quite simple to build a locality-sensitive family of functions for the Hamming distance. Suppose we have a space of d-dimensional vectors, and $h(x, y)$ denotes the Hamming distance between vectors x and y. If we take any one position of the vectors, say the ith position, we can define the function $f_i(x)$ to be the ith bit of vector x. Then $f_i(x) = f_i(y)$ if and only if vectors x and y agree in the ith position. Then the probability that $f_i(x) = f_i(y)$ for a randomly chosen i is exactly $1 - h(x, y)/d$; i.e., it is the fraction of positions in which x and y agree.

This situation is almost exactly like the one we encountered for minhashing. Thus, the family **F** consisting of the functions $\{f_1, f_2, \ldots, f_d\}$ is a

$$(d_1, d_2, 1 - d_1/d, 1 - d_2/d)\text{-sensitive}$$

family of hash functions, for any $d_1 < d_2$. There are only two differences between this family and the family of minhash functions.

1. While Jaccard distance runs from 0 to 1, the Hamming distance on a vector space of dimension d runs from 0 to d. It is therefore necessary to scale the distances by dividing by d, to turn them into probabilities.

2. While there is essentially an unlimited supply of minhash functions, the size of the family $\mathbf{F}$ for Hamming distance is only d.

The first point is of no consequence; it only requires that we divide by d at appropriate times. The second point is more serious. If d is relatively small, then we are limited in the number of functions that can be composed using the AND and OR constructions, thereby limiting how steep we can make the S-curve be.

3.7.2 Random Hyperplanes and the Cosine Distance

Recall from Section 3.5.4 that the cosine distance between two vectors is the angle between the vectors. For instance, we see in Fig. 3.12 two vectors x and y that make an angle θ between them. Note that these vectors may be in a space of many dimensions, but they always define a plane, and the angle between them is measured in this plane. Figure 3.12 is a "top-view" of the plane containing x and y.

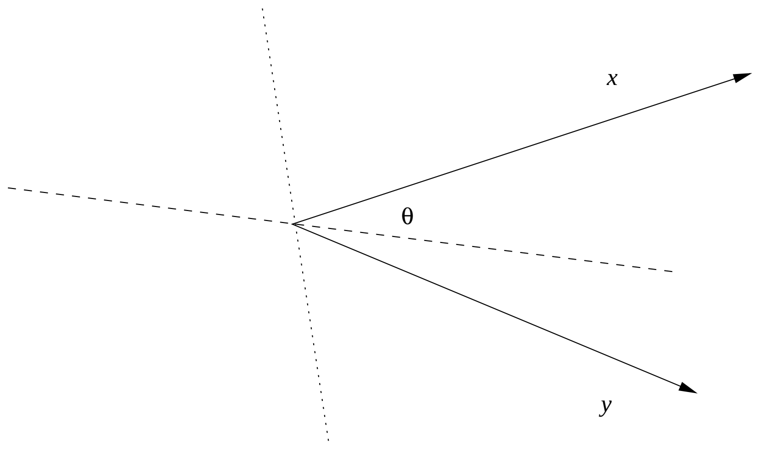


Figure 3.12: Two vectors make an angle θ

Suppose we pick a hyperplane through the origin. This hyperplane intersects the plane of x and y in a line. Figure 3.12 suggests two possible hyperplanes, one whose intersection is the dashed line and the other's intersection is the dotted line. To pick a random hyperplane, we actually pick the normal vector to the hyperplane, say v. The hyperplane is then the set of points whose dot product with v is 0.

First, consider a vector v that is normal to the hyperplane whose projection is represented by the dashed line in Fig. 3.12; that is, x and y are on different sides of the hyperplane. Then the dot products $v.x$ and $v.y$ will have different signs. If we assume, for instance, that v is a vector whose projection onto the plane of x and y is above the dashed line in Fig. 3.12, then $v.x$ is positive, while $v.y$ is negative. The normal vector v instead might extend in the opposite direction, below the dashed line. In that case $v.x$ is negative and $v.y$ is positive, but the signs are still different.

On the other hand, the randomly chosen vector v could be normal to a hyperplane like the dotted line in Fig. 3.12. In that case, both $v.x$ and $v.y$ have the same sign. If the projection of v extends to the right, then both dot products are positive, while if v extends to the left, then both are negative.

What is the probability that the randomly chosen vector is normal to a hyperplane that looks like the dashed line rather than the dotted line? All angles for the line that is the intersection of the random hyperplane and the plane of x and y are equally likely. Thus, the hyperplane will look like the dashed line with probability $\theta/180$ and will look like the dotted line otherwise.

Thus, each hash function f in our locality-sensitive family $\mathbf{F}$ is built from a randomly chosen vector v_f. Given two vectors x and y, say $f(x) = f(y)$ if and only if the dot products $v_f.x$ and $v_f.y$ have the same sign. Then $\mathbf{F}$ is a locality-sensitive family for the cosine distance. The parameters are essentially the same as for the Jaccard-distance family described in Section 3.6.2, except the scale of distances is 0–180 rather than 0–1. That is, $\mathbf{F}$ is a

$$(d_1, d_2, (180 - d_1)/180, (180 - d_2)/180)\text{-sensitive}$$

family of hash functions. From this basis, we can amplify the family as we wish, just as for the minhash-based family.

3.7.3 Sketches

Instead of chosing a random vector from all possible vectors, it turns out to be sufficiently random if we restrict our choice to vectors whose components are $+1$ and -1. The dot product of any vector x with a vector v of $+1$'s and -1's is formed by adding the components of x where v is $+1$ and then subtracting the other components of x – those where v is -1.

If we pick a collection of random vectors, say $v_1, v_2, \ldots, v_n$, then we can apply them to an arbitrary vector x by computing $v_1.x, v_2.x, \ldots, v_n.x$ and then replacing any positive value by $+1$ and any negative value by -1. The result is called the *sketch* of x. You can handle 0's arbitrarily, e.g., by chosing a result $+1$ or -1 at random. Since there is only a tiny probability of a zero dot product, the choice has essentially no effect.

Example 3.21: Suppose our space consists of 4-dimensional vectors, and we pick three random vectors: $v_1 = [+1, -1, +1, +1]$, $v_2 = [-1, +1, -1, +1]$, and $v_3 = [+1, +1, -1, -1]$. For the vector $x = [3, 4, 5, 6]$, the sketch is $[+1, +1, -1]$.

That is, $v_1.x = 3-4+5+6 = 10$. Since the result is positive, the first component of the sketch is +1. Similarly, $v_2.x = 3$ and $v_3.x = -4$, so the second component of the sketch is +1 and the third component is −1.

Consider the vector $y = [4, 3, 2, 1]$. We can similarly compute its sketch to be $[+1, -1, +1]$. Since the sketches for x and y agree in 1/3 of the positions, we estimate that the angle between them is 120 degrees. That is, a randomly chosen hyperplane is twice as likely to look like the dashed line in Fig. 3.12 than like the dotted line.

The above conclusion turns out to be quite wrong. We can calculate the cosine of the angle between x and y to be $x.y$, which is

$$6 \times 1 + 5 \times 2 + 4 \times 3 + 3 \times 4 = 40$$

divided by the magnitudes of the two vectors. These magnitudes are

$$\sqrt{6^2 + 5^2 + 4^2 + 3^2} = 9.274$$

and $\sqrt{1^2 + 2^2 + 3^2 + 4^2} = 5.477$. Thus, the cosine of the angle between x and y is 0.7875, and this angle is about 38 degrees. However, if you look at all 16 different vectors v of length 4 that have +1 and −1 as components, you find that there are only four of these whose dot products with x and y have a different sign, namely v_2, v_3, and their complements $[+1, -1, +1, -1]$ and $[-1, -1, +1, +1]$. Thus, had we picked all sixteen of these vectors to form a sketch, the estimate of the angle would have been $180/4 = 45$ degrees. □

3.7.4 LSH Families for Euclidean Distance

Now, let us turn to the Euclidean distance (Section 3.5.2), and see if we can develop a locality-sensitive family of hash functions for this distance. We shall start with a 2-dimensional Euclidean space. Each hash function f in our family $\mathbf{F}$ will be associated with a randomly chosen line in this space. Pick a constant a and divide the line into segments of length a, as suggested by Fig. 3.13, where the "random" line has been oriented to be horizontal.

The segments of the line are the buckets into which function f hashes points. A point is hashed to the bucket in which its projection onto the line lies. If the distance d between two points is small compared with a, then there is a good chance the two points hash to the same bucket, and thus the hash function f will declare the two points equal. For example, if $d = a/2$, then there is at least a 50% chance the two points will fall in the same bucket. In fact, if the angle θ between the randomly chosen line and the line connecting the points is large, then there is an even greater chance that the two points will fall in the same bucket. For instance, if θ is 90 degrees, then the two points are certain to fall in the same bucket.

However, suppose d is larger than a. In order for there to be any chance of the two points falling in the same bucket, we need $d \cos\theta \leq a$. The diagram of Fig. 3.13 suggests why this requirement holds. Note that even if $d \cos\theta \ll a$ it

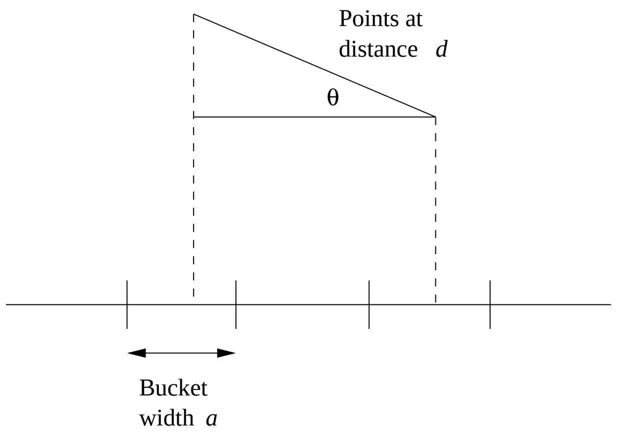


Figure 3.13: Two points at distance $d \gg a$ have a small chance of being hashed to the same bucket

is still not certain that the two points will fall in the same bucket. However, we can guarantee the following. If $d \geq 2a$, then there is no more than a 1/3 chance the two points fall in the same bucket. The reason is that for $\cos\theta$ to be less than 1/2, we need to have θ in the range 60 to 90 degrees. If θ is in the range 0 to 60 degrees, then $\cos\theta$ is more than 1/2. But since θ is the smaller angle between two randomly chosen lines in the plane, θ is twice as likely to be between 0 and 60 as it is to be between 60 and 90.

We conclude that the family **F** just described forms a $(a/2, 2a, 1/2, 1/3)$-sensitive family of hash functions. That is, for distances up to $a/2$ the probability is at least 1/2 that two points at that distance will fall in the same bucket, while for distances at least $2a$ the probability points at that distance will fall in the same bucket is at most 1/3. We can amplify this family as we like, just as for the other examples of locality-sensitive hash functions we have discussed.

3.7.5 More LSH Families for Euclidean Spaces

There is something unsatisfying about the family of hash functions developed in Section 3.7.4. First, the technique was only described for two-dimensional Euclidean spaces. What happens if our data is points in a space with many dimensions? Second, for Jaccard and cosine distances, we were able to develop locality-sensitive families for any pair of distances d_1 and d_2 as long as $d_1 < d_2$. In Section 3.7.4 we appear to need the stronger condition $d_1 < 4d_2$.

However, we claim that there is a locality-sensitive family of hash functions for any $d_1 < d_2$ and for any number of dimensions. The family's hash functions still derive from random lines through the space and a bucket size a that partitions the line. We still hash points by projecting them onto the line. Given that $d_1 < d_2$, we may not know what the probability p_1 is that two points at distance d_1 hash to the same bucket, but we can be certain that it is greater than p_2, the probability that two points at distance d_2 hash to the same bucket. The reason is that this probability surely grows as the distance shrinks. Thus, even if we cannot calculate p_1 and p_2 easily, we know that there is a (d_1, d_2, p_1, p_2)-sensitive family of hash functions for any $d_1 < d_2$ and any given number of dimensions.

Using the amplification techniques of Section 3.6.3, we can then adjust the two probabilities to surround any particular value we like, and to be as far apart as we like. Of course, the further apart we want the probabilities to be, the larger the number of basic hash functions in **F** we must use.

3.7.6 Exercises for Section 3.7

Exercise 3.7.1: Suppose we construct the basic family of six locality-sensitive functions for vectors of length six. For each pair of the vectors 000000, 110011, 010101, and 011100, which of the six functions makes them candidates?

Exercise 3.7.2: Let us compute sketches using the following four "random" vectors:

$$\begin{array}{ll} v_1 = [+1, +1, +1, -1] & v_2 = [+1, +1, -1, +1] \\ v_3 = [+1, -1, +1, +1] & v_4 = [-1, +1, +1, +1] \end{array}$$

Compute the sketches of the following vectors.

(a) $[2, 3, 4, 5]$.

(b) $[-2, 3, -4, 5]$.

(c) $[2, -3, 4, -5]$.

For each pair, what is the estimated angle between them, according to the sketches? What are the true angles?

Exercise 3.7.3: Suppose we form sketches by using all sixteen of the vectors of length 4, whose components are each $+1$ or -1. Compute the sketches of the three vectors in Exercise 3.7.2. How do the estimates of the angles between each pair compare with the true angles?

Exercise 3.7.4: Suppose we form sketches using the four vectors from Exercise 3.7.2.

! (a) What are the constraints on a, b, c, and d that will cause the sketch of the vector $[a, b, c, d]$ to be $[+1, +1, +1, +1]$?

!! (b) Consider two vectors $[a, b, c, d]$ and $[e, f, g, h]$. What are the conditions on $a, b, \ldots, h$ that will make the sketches of these two vectors be the same?

Exercise 3.7.5: Suppose we have points in a 3-dimensional Euclidean space: $p_1 = (1, 2, 3)$, $p_2 = (0, 2, 4)$, and $p_3 = (4, 3, 2)$. Consider the three hash functions defined by the three axes (to make our calculations very easy). Let buckets be of length a, with one bucket the interval $[0, a)$ (i.e., the set of points x such that $0 \le x < a$), the next $[a, 2a)$, the previous one $[-a, 0)$, and so on.

(a) For each of the three lines, assign each of the points to buckets, assuming $a = 1$.

(b) Repeat part (a), assuming $a = 2$.

(c) What are the candidate pairs for the cases $a = 1$ and $a = 2$?

! (d) For each pair of points, for what values of a will that pair be a candidate pair?

3.8 Applications of Locality-Sensitive Hashing

In this section, we shall explore three examples of how LSH is used in practice. In each case, the techniques we have learned must be modified to meet certain constraints of the problem. The three subjects we cover are:

1. *Entity Resolution*: This term refers to matching data records that refer to the same real-world entity, e.g., the same person. The principal problem addressed here is that the similarity of records does not match exactly either the similar-sets or similar-vectors models of similarity on which the theory is built.

2. *Matching Fingerprints*: It is possible to represent fingerprints as sets. However, we shall explore a different family of locality-sensitive hash functions from the one we get by minhashing.

3. *Matching Newspaper Articles*: Here, we consider a different notion of shingling that focuses attention on the core article in an on-line newspaper's Web page, ignoring all the extraneous material such as ads and newspaper-specific material.

3.8.1 Entity Resolution

It is common to have several data sets available, and to know that they refer to some of the same entities. For example, several different bibliographic sources provide information about many of the same books or papers. In the general case, we have records describing entities of some type, such as people or books. The records may all have the same format, or they may have different formats, with different kinds of information.

There are many reasons why information about an entity may vary, even if the field in question is supposed to be the same. For example, names may be expressed differently in different records because of misspellings, absence of a middle initial, use of a nickname, and many other reasons. For example, “Bob S. Jomes” and “Robert Jones Jr.” may or may not be the same person. If records come from different sources, the fields may differ as well. One source’s records may have an “age” field, while another does not. The second source might have a “date of birth” field, or it may have no information at all about when a person was born.

3.8.2 An Entity-Resolution Example

We shall examine a real example of how LSH was used to deal with an entity-resolution problem. Company A was engaged by Company B to solicit customers for B. Company B would pay A a yearly fee, as long as the customer maintained their subscription. They later quarreled and disagreed over how many customers A had provided to B. Each had about 1,000,000 records, some of which described the same people; those were the customers A had provided to B. The records had different data fields, but unfortunately none of those fields was “this is a customer that A had provided to B.” Thus, the problem was to match records from the two sets to see if a pair represented the same person.

Each record had fields for the name, address, and phone number of the person. However, the values in these fields could differ for many reasons. Not only were there the misspellings and other naming differences mentioned in Section 3.8.1, but there were other opportunities to disagree as well. A customer might give their home phone to A and their cell phone to B. Or they might move, and tell B but not A (because they no longer had need for a relationship with A). Area codes of phones sometimes change.

The strategy for identifying records involved scoring the differences in three fields: name, address, and phone. To create a *score* describing the likelihood that two records, one from A and the other from B, described the same person, 100 points was assigned to each of the three fields, so records with exact matches in all three fields got a score of 300. However, there were deductions for mismatches in each of the three fields. As a first approximation, edit-distance (Section 3.5.5) was used, but the penalty grew quadratically with the distance. Then, certain publicly available tables were used to reduce the penalty in appropriate situations. For example, “Bill” and “William” were treated as if they differed in only one letter, even though their edit-distance is 5.

However, it is not feasible to score all one trillion pairs of records. Thus, a simple LSH was used to focus on likely candidates. Three “hash functions” were used. The first sent records to the same bucket only if they had identical names; the second did the same but for identical addresses, and the third did the same for phone numbers. In practice, there was no hashing; rather the records were sorted by name, so records with identical names would appear consecutively and get scored for overall similarity of the name, address, and phone. Then the records were sorted by address, and those with the same

When Are Record Matches Good Enough?

While every case will be different, it may be of interest to know how the experiment of Section 3.8.3 turned out on the data of Section 3.8.2. For scores down to 185, the value of x was very close to 10; i.e., these scores indicated that the likelihood of the records representing the same person was essentially 1. Note that a score of 185 in this example represents a situation where one field is the same (as would have to be the case, or the records would never even be scored), one field was completely different, and the third field had a small discrepancy. Moreover, for scores as low as 115, the value of x was noticeably less than 45, meaning that some of these pairs did represent the same person. Note that a score of 115 represents a case where one field is the same, but there is only a slight similarity in the other two fields.

address were scored. Finally, the records were sorted a third time by phone, and records with identical phones were scored.

This approach missed a record pair that truly represented the same person but none of the three fields matched exactly. Since the goal was to prove in a court of law that the persons were the same, it is unlikely that such a pair would have been accepted by a judge as sufficiently similar anyway.

3.8.3 Validating Record Matches

What remains is to determine how high a score indicates that two records truly represent the same individual. In the example at hand, there was an easy way to make that decision, and the technique can be applied in many similar situations. It was decided to look at the creation-dates for the records at hand, and to assume that 90 days was an absolute maximum delay between the time the service was bought at Company A and registered at B. Thus, a proposed match between two records that were chosen at random, subject only to the constraint that the date on the B-record was between 0 and 90 days after the date on the A-record, would have an average delay of 45 days.

It was found that of the pairs with a perfect 300 score, the average delay was 10 days. If you assume that 300-score pairs are surely correct matches, then you can look at the pool of pairs with any given score s, and compute the average delay of those pairs. Suppose that the average delay is x, and the fraction of true matches among those pairs with score s is f. Then $x = 10f + 45(1 - f)$, or $x = 45 - 35f$. Solving for f, we find that the fraction of the pairs with score s that are truly matches is $(45 - x)/35$.

The same trick can be used whenever:

1. There is a scoring system used to evaluate the likelihood that two records

represent the same entity, and

2. There is some field, not used in the scoring, from which we can derive a measure that differs, on average, for true pairs and false pairs.

For instance, suppose there were a "height" field recorded by both companies A and B in our running example. We can compute the average difference in height for pairs of random records, and we can compute the average difference in height for records that have a perfect score (and thus surely represent the same entities). For a given score s, we can evaluate the average height difference of the pairs with that score and estimate the probability of the records representing the same entity. That is, if h_0 is the average height difference for the perfect matches, h_1 is the average height difference for random pairs, and h is the average height difference for pairs of score s, then the fraction of good pairs with score s is $(h_1 - h)/(h_1 - h_0)$.

3.8.4 Matching Fingerprints

When fingerprints are matched by computer, the usual representation is not an image, but a set of locations in which *minutiae* are located. A minutia, in the context of fingerprint descriptions, is a place where something unusual happens, such as two ridges merging or a ridge ending. If we place a grid over a fingerprint, we can represent the fingerprint by the set of grid squares in which minutiae are located.

Ideally, before overlaying the grid, fingerprints are normalized for size and orientation, so that if we took two images of the same finger, we would find minutiae lying in exactly the same grid squares. We shall not consider here the best ways to normalize images. Let us assume that some combination of techniques, including choice of grid size and placing a minutia in several adjacent grid squares if it lies close to the border of the squares enables us to assume that grid squares from two images have a significantly higher probability of agreeing in the presence or absence of a minutia than if they were from images of different fingers.

Thus, fingerprints can be represented by sets of grid squares – those where their minutiae are located – and compared like any sets, using the Jaccard similarity or distance. There are two versions of fingerprint comparison, however.

- The *many-one* problem is the one we typically expect. A fingerprint has been found on a gun, and we want to compare it with all the fingerprints in a large database, to see which one matches.

- The *many-many* version of the problem is to take the entire database, and see if there are any pairs that represent the same individual.

While the many-many version matches the model that we have been following for finding similar items, the same technology can be used to speed up the many-one problem.

3.8.5 A LSH Family for Fingerprint Matching

We could minhash the sets that represent a fingerprint, and use the standard LSH technique from Section 3.4. However, since the sets are chosen from a relatively small set of grid points (perhaps 1000), the need to minhash them into more succinct signatures is not clear. We shall study here another form of locality-sensitive hashing that works well for data of the type we are discussing.

Suppose for an example that the probability of finding a minutia in a random grid square of a random fingerprint is 20%. Also, assume that if two fingerprints come from the same finger, and one has a minutia in a given grid square, then the probability that the other does too is 80%. We can define a locality-sensitive family of hash functions as follows. Each function f in this family $\mathbf{F}$ is defined by three grid squares. Function f says "yes" for two fingerprints if both have minutiae in all three grid squares, and otherwise f says "no." Put another way, we may imagine that f sends to a single bucket all fingerprints that have minutiae in all three of f's grid points, and sends each other fingerprint to a bucket of its own. In what follows, we shall refer to the first of these buckets as "the" bucket for f and ignore the buckets that are required to be singletons.

If we want to solve the many-one problem, we can use many functions from the family $\mathbf{F}$ and precompute their buckets of fingerprints to which they answer "yes." Then, given a new fingerprint that we want to match, we determine which of these buckets it belongs to and compare it with all the fingerprints found in any of those buckets. To solve the many-many problem, we compute the buckets for each of the functions and compare all fingerprints in each of the buckets.

Let us consider how many functions we need to get a reasonable probability of catching a match, without having to compare the fingerprint on the gun with each of the millions of fingerprints in the database. First, the probability that two fingerprints from different fingers would be in the bucket for a function f in $\mathbf{F}$ is $(0.2)^6 = 0.000064$. The reason is that they will both go into the bucket only if they each have a minutia in each of the three grid points associated with f, and the probability of each of those six independent events is 0.2.

Now, consider the probability that two fingerprints from the same finger wind up in the bucket for f. The probability that the first fingerprint has minutiae in each of the three squares belonging to f is $(0.2)^3 = 0.008$. However, if it does, then the probability is $(0.8)^3 = 0.512$ that the other fingerprint will as well. Thus, if the fingerprints are from the same finger, there is a $0.008 \times 0.512 = 0.004096$ probability that they will both be in the bucket of f. That is not much; it is about one in 200. However, if we use many functions from $\mathbf{F}$, but not too many, then we can get a good probability of matching fingerprints from the same finger while not having too many false positives – fingerprints that must be considered but do not match.

Example 3.22: For a specific example, let us suppose that we use 1024 functions chosen randomly from $\mathbf{F}$. Next, we shall construct a new family $\mathbf{F}_1$ by performing a 1024-way OR on $\mathbf{F}$. Then the probability that $\mathbf{F}_1$

will put fingerprints from the same finger together in at least one bucket is $1 - (1 - 0.004096)^{1024} = 0.985$. On the other hand, the probability that two fingerprints from different fingers will be placed in the same bucket is $(1 - (1 - 0.000064)^{1024} = 0.063$. That is, we get about 1.5% false negatives and about 6.3% false positives. □

The result of Example 3.22 is not the best we can do. While it offers only a 1.5% chance that we shall fail to identify the fingerprint on the gun, it does force us to look at 6.3% of the entire database. Increasing the number of functions from **F** will increase the number of false positives, with only a small benefit of reducing the number of false negatives below 1.5%. On the other hand, we can also use the AND construction, and in so doing, we can greatly reduce the probability of a false positive, while making only a small increase in the false-negative rate. For instance, we could take 2048 functions from **F** in two groups of 1024. Construct the buckets for each of the functions. However, given a fingerprint P on the gun:

1. Find the buckets from the first group in which P belongs, and take the union of these buckets.

2. Do the same for the second group.

3. Take the intersection of the two unions.

4. Compare P only with those fingerprints in the intersection.

Note that we still have to take unions and intersections of large sets of fingerprints, but we compare only a small fraction of those. It is the comparison of fingerprints that takes the bulk of the time; in steps (1) and (2) fingerprints can be represented by their integer indices in the database.

If we use this scheme, the probability of detecting a matching fingerprint is $(0.985)^2 = 0.970$; that is, we get about 3% false negatives. However, the probability of a false positive is $(0.063)^2 = 0.00397$. That is, we only have to examine about 1/250th of the database.

3.8.6 Similar News Articles

Our last case study concerns the problem of organizing a large repository of on-line news articles by grouping together Web pages that were derived from the same basic text. It is common for organizations like The Associated Press to produce a news item and distribute it to many newspapers. Each newspaper puts the story in its on-line edition, but surrounds it by information that is special to that newspaper, such as the name and address of the newspaper, links to related articles, and links to ads. In addition, it is common for the newspaper to modify the article, perhaps by leaving off the last few paragraphs or even deleting text from the middle. As a result, the same news article can appear quite different at the Web sites of different newspapers.

The problem looks very much like the one that was suggested in Section 3.4: find documents whose shingles have a high Jaccard similarity. Note that this problem is different from the problem of finding news articles that tell about the same events. The latter problem requires other techniques, typically examining the set of important words in the documents (a concept we discussed briefly in Section 1.3.1) and clustering them to group together different articles about the same topic.

However, an interesting variation on the theme of shingling was found to be more effective for data of the type described. The problem is that shingling as we described it in Section 3.2 treats all parts of a document equally. However, we wish to ignore parts of the document, such as ads or the headlines of other articles to which the newspaper added a link, that are not part of the news article. It turns out that there is a noticeable difference between text that appears in prose and text that appears in ads or headlines. Prose has a much greater frequency of stop words, the very frequent words such as "the" or "and." The total number of words that are considered stop words varies with the application, but it is common to use a list of several hundred of the most frequent words.

Example 3.23: A typical ad might say simply "Buy Sudzo." On the other hand, a prose version of the same thought that might appear in an article is "I recommend that you buy Sudzo for your laundry." In the latter sentence, it would be normal to treat "I," "that," "you," "for," and "your" as stop words. □

Suppose we define a *shingle* to be a stop word followed by the next two words. Then the ad "Buy Sudzo" from Example 3.23 has no shingles and would not be reflected in the representation of the Web page containing that ad. On the other hand, the sentence from Example 3.23 would be represented by five shingles: "I recommend that," "that you buy," "you buy Sudzo," "for your laundry," and "your laundry x," where x is whatever word follows that sentence.

Suppose we have two Web pages, each of which consists of half news text and half ads or other material that has a low density of stop words. If the news text is the same but the surrounding material is different, then we would expect that a large fraction of the shingles of the two pages would be the same. They might have a Jaccard similarity of 75%. However, if the surrounding material is the same but the news content is different, then the number of common shingles would be small, perhaps 25%. If we were to use the conventional shingling, where shingles are (say) sequences of 10 consecutive characters, we would expect the two documents to share half their shingles (i.e., a Jaccard similarity of 1/3), regardless of whether it was the news or the surrounding material that they shared.

3.8.7 Exercises for Section 3.8

Exercise 3.8.1: Suppose we are trying to perform entity resolution among bibliographic references, and we score pairs of references based on the similarities of their titles, list of authors, and place of publication. Suppose also that all references include a year of publication, and this year is equally likely to be any of the ten most recent years. Further, suppose that we discover that among the pairs of references with a perfect score, there is an average difference in the publication year of 0.1.[6] Suppose that the pairs of references with a certain score s are found to have an average difference in their publication dates of 2. What is the fraction of pairs with score s that truly represent the same publication? *Note*: Do not make the mistake of assuming the average difference in publication date between random pairs is 5 or 5.5. You need to calculate it exactly, and you have enough information to do so.

Exercise 3.8.2: Suppose we use the family $\mathbf{F}$ of functions described in Section 3.8.5, where there is a 20% chance of a minutia in an grid square, an 80% chance of a second copy of a fingerprint having a minutia in a grid square where the first copy does, and each function in $\mathbf{F}$ being formed from three grid squares. In Example 3.22, we constructed family $\mathbf{F}_1$ by using the OR construction on 1024 members of $\mathbf{F}$. Suppose we instead used family $\mathbf{F}_2$ that is a 2048-way OR of members of $\mathbf{F}$.

(a) Compute the rates of false positives and false negatives for $\mathbf{F}_2$.

(b) How do these rates compare with what we get if we organize the same 2048 functions into a 2-way AND of members of $\mathbf{F}_1$, as was discussed at the end of Section 3.8.5?

Exercise 3.8.3: Suppose fingerprints have the same statistics outlined in Exercise 3.8.2, but we use a base family of functions $\mathbf{F}'$ defined like $\mathbf{F}$, but using only two randomly chosen grid squares. Construct another set of functions $\mathbf{F}'_1$ from $\mathbf{F}'$ by taking the n-way OR of functions from $\mathbf{F}'$. What, as a function of n, are the false positive and false negative rates for $\mathbf{F}'_1$?

Exercise 3.8.4: Suppose we use the functions $\mathbf{F}_1$ from Example 3.22, but we want to solve the many-many problem.

(a) If two fingerprints are from the same finger, what is the probability that they will not be compared (i.e., what is the false negative rate)?

(b) What fraction of the fingerprints from different fingers will be compared (i.e., what is the false positive rate)?

! **Exercise 3.8.5:** Assume we have the set of functions $\mathbf{F}$ as in Exercise 3.8.2, and we construct a new set of functions $\mathbf{F}_3$ by an n-way OR of functions in $\mathbf{F}$. For what value of n is the sum of the false positive and false negative rates minimized?

[6] We might expect the average to be 0, but in practice, errors in publication year do occur.

3.9 Methods for High Degrees of Similarity

LSH-based methods appear most effective when the degree of similarity we accept is relatively low. When we want to find sets that are almost identical, there are other methods that can be faster. Moreover, these methods are exact, in that they find every pair of items with the desired degree of similarity. There are no false negatives, as there can be with LSH.

3.9.1 Finding Identical Items

The extreme case is finding identical items, for example, Web pages that are identical, character-for-character. It is straightforward to compare two documents and tell whether they are identical, but we still must avoid having to compare every pair of documents. Our first thought would be to hash documents based on their first few characters, and compare only those documents that fell into the same bucket. That scheme should work well, unless all the documents begin with the same characters, such as an HTML header.

Our second thought would be to use a hash function that examines the entire document. That would work, and if we use enough buckets, it would be very rare that two documents went into the same bucket, yet were not identical. The downside of this approach is that we must examine every character of every document. If we limit our examination to a small number of characters, then we never have to examine a document that is unique and falls into a bucket of its own.

A better approach is to pick some fixed random positions for all documents, and make the hash function depend only on these. This way, we can avoid a problem where there is a common prefix for all or most documents, yet we need not examine entire documents unless they fall into a bucket with another document. One problem with selecting fixed positions is that if some documents are short, they may not have some of the selected positions. However, if we are looking for highly similar documents, we never need to compare two documents that differ significantly in their length. We exploit this idea in Section 3.9.3.

3.9.2 Representing Sets as Strings

Now, let us focus on the harder problem of finding, in a large collection of sets, all pairs that have a high Jaccard similarity, say at least 0.9. We can represent a set by sorting the elements of the universal set in some fixed order, and representing any set by listing its elements in this order. The list is essentially a string of "characters," where the characters are the elements of the universal set. These strings are unusual, however, in that:

1. No character appears more than once in a string, and

2. If two characters appear in two different strings, then they appear in the same order in both strings.

Example 3.24 : Suppose the universal set consists of the 26 lower-case letters, and we use the normal alphabetical order. Then the set $\{d, a, b\}$ is represented by the string `abd`. $\square$

In what follows, we shall assume all strings represent sets in the manner just described. Thus, we shall talk about the Jaccard similarity of strings, when strictly speaking we mean the similarity of the sets that the strings represent. Also, we shall talk of the length of a string, as a surrogate for the number of elements in the set that the string represents.

Note that the documents discussed in Section 3.9.1 do not exactly match this model, even though we can see documents as strings. To fit the model, we would shingle the documents, assign an order to the shingles, and represent each document by its list of shingles in the selected order.

3.9.3 Length-Based Filtering

The simplest way to exploit the string representation of Section 3.9.2 is to sort the strings by length. Then, each string s is compared with those strings t that follow s in the list, but are not too long. Suppose the lower bound on Jaccard similarity between two strings is J. For any string x, denote its length by L_x. Note that $L_s \leq L_t$. The intersection of the sets represented by s and t cannot have more than L_s members, while their union has at least L_t members. Thus, the Jaccard similarity of s and t, which we denote $\text{SIM}(s, t)$, is at most L_s/L_t. That is, in order for s and t to require comparison, it must be that $J \leq L_s/L_t$, or equivalently, $L_t \leq L_s/J$.

Example 3.25 : Suppose that s is a string of length 9, and we are looking for strings with at least 0.9 Jaccard similarity. Then we have only to compare s with strings following it in the length-based sorted order that have length at most $9/0.9 = 10$. That is, we compare s with those strings of length 9 that follow it in order, and all strings of length 10. We have no need to compare s with any other string.

Suppose the length of s were 8 instead. Then s would be compared with following strings of length up to $8/0.9 = 8.89$. That is, a string of length 9 would be too long to have a Jaccard similarity of 0.9 with s, so we only have to compare s with the strings that have length 8 but follow it in the sorted order. $\square$

3.9.4 Prefix Indexing

In addition to length, there are several other features of strings that can be exploited to limit the number of comparisons that must be made to identify all pairs of similar strings. The simplest of these options is to create an index for each symbol; recall a symbol of a string is any one of the elements of the universal set. For each string s, we select a prefix of s consisting of the first p

A Better Ordering for Symbols

Instead of using the obvious order for elements of the universal set, e.g., lexicographic order for shingles, we can order symbols rarest first. That is, determine how many times each element appears in the collection of sets, and order them by this count, lowest first. The advantage of doing so is that the symbols in prefixes will tend to be rare. Thus, they will cause that string to be placed in index buckets that have relatively few members. Then, when we need to examine a string for possible matches, we shall find few other strings that are candidates for comparison.

symbols of s. How large p must be depends on L_s and J, the lower bound on Jaccard similarity. We add string s to the index for each of its first p symbols.

In effect, the index for each symbol becomes a bucket of strings that must be compared. We must be certain that any other string t such that $\text{SIM}(s,t) \geq J$ will have at least one symbol in its prefix that also appears in the prefix of s.

Suppose not; rather $\text{SIM}(s,t) \geq J$, but t has none of the first p symbols of s. Then the highest Jaccard similarity that s and t can have occurs when t is a suffix of s, consisting of everything but the first p symbols of s. The Jaccard similarity of s and t would then be $(L_s - p)/L_s$. To be sure that we do not have to compare s with t, we must be certain that $J > (L_s - p)/L_s$. That is, p must be at least $\lfloor (1-J)L_s \rfloor + 1$. Of course we want p to be as small as possible, so we do not index string s in more buckets than we need to. Thus, we shall hereafter take $p = \lfloor (1-J)L_s \rfloor + 1$ to be the length of the prefix that gets indexed.

Example 3.26 : Suppose $J = 0.9$. If $L_s = 9$, then $p = \lfloor 0.1 \times 9 \rfloor + 1 = \lfloor 0.9 \rfloor + 1 = 1$. That is, we need to index s under only its first symbol. Any string t that does not have the first symbol of s in a position such that t is indexed by that symbol will have Jaccard similarity with s that is less than 0.9. Suppose s is `bcdefghij`. Then s is indexed under `b` only. Suppose t does not begin with `b`. There are two cases to consider.

1. If t begins with `a`, and $\text{SIM}(s,t) \geq 0.9$, then it can only be that t is `abcdefghij`. But if that is the case, t will be indexed under both `a` and `b`. The reason is that $L_t = 10$, so t will be indexed under the symbols of its prefix of length $\lfloor 0.1 \times 10 \rfloor + 1 = 2$.

2. If t begins with `c` or a later letter, then the maximum value of $\text{SIM}(s,t)$ occurs when t is `cdefghij`. But then $\text{SIM}(s,t) = 8/9 < 0.9$.

In general, with $J = 0.9$, strings of length up to 9 are indexed by their first symbol, strings of lengths 10–19 are indexed under their first two symbols, strings of length 20–29 are indexed under their first three symbols, and so on. □

We can use the indexing scheme in two ways, depending on whether we are trying to solve the many-many problem or a many-one problem; recall the distinction was introduced in Section 3.8.4. For the many-one problem, we create the index for the entire database. To query for matches to a new set S, we convert that set to a string s, which we call the *probe* string. Determine the length of the prefix that must be considered, that is, $\lfloor(1-J)L_s\rfloor + 1$. For each symbol appearing in one of the prefix positions of s, we look in the index bucket for that symbol, and we compare s with all the strings appearing in that bucket.

If we want to solve the many-many problem, start with an empty database of strings and indexes. For each set S, we treat S as a new set for the many-one problem. We convert S to a string s, which we treat as a probe string in the many-one problem. However, after we examine an index bucket, we also add s to that bucket, so s will be compared with later strings that could be matches.

3.9.5 Using Position Information

Consider the strings $s =$ `acdefghijk` and $t =$ `bcdefghijk`, and assume $J = 0.9$. Since both strings are of length 10, they are indexed under their first two symbols. Thus, s is indexed under `a` and `c`, while t is indexed under `b` and `c`. Whichever is added last will find the other in the bucket for `c`, and they will be compared. However, since `c` is the second symbol of both, we know there will be two symbols, `a` and `b` in this case, that are in the union of the two sets but not in the intersection. Indeed, even though s and t are identical from `c` to the end, their intersection is 9 symbols and their union is 11; thus $\text{SIM}(s,t) = 9/11$, which is less than 0.9.

If we build our index based not only on the symbol, but on the position of the symbol within the string, we could avoid comparing s and t above. That is, let our index have a bucket for each pair (x, i), containing the strings that have symbol x in position i of their prefix. Given a string s, and assuming J is the minimum desired Jaccard similarity, we look at the prefix of s, that is, the positions 1 through $\lfloor(1-J)L_s\rfloor + 1$. If the symbol in position i of the prefix is x, add s to the index bucket for (x, i).

Now consider s as a probe string. With what buckets must it be compared? We shall visit the symbols of the prefix of s from the left, and we shall take advantage of the fact that we only need to find a possible matching string t if none of the previous buckets we have examined for matches held t. That is, we only need to find a candidate match once. Thus, if we find that the ith symbol of s is x, then we need look in the bucket (x, j) for certain small values of j.

To compute the upper bound on j, suppose t is a string none of whose first $j-1$ symbols matched anything in s, but the ith symbol of s is the same as the jth symbol of t. The highest value of $\text{SIM}(s,t)$ occurs if s and t are identical

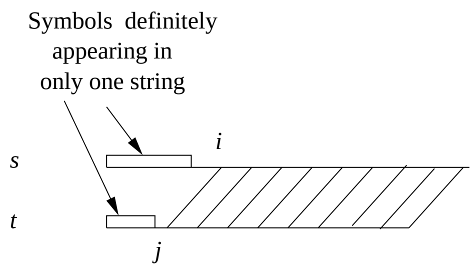


Figure 3.14: Strings s and t begin with $i-1$ and $j-1$ unique symbols, respectively, and then agree beyond that

beyond their ith and jth symbols, respectively, as suggested by Fig. 3.14. If that is the case, the size of their intersection is $L_s - i + 1$, since that is the number of symbols of s that could possibly be in t. The size of their union is at least $L_s + j - 1$. That is, s surely contributes L_s symbols to the union, and there are also at least $j - 1$ symbols of t that are not in s. The ratio of the sizes of the intersection and union must be at least J, so we must have:

$$\frac{L_s - i + 1}{L_s + j - 1} \geq J$$

If we isolate j in this inequality, we have $j \leq \big(L_s(1-J) - i + 1 + J\big)/J$.

Example 3.27: Consider the string $s =$ `acdefghijk` with $J = 0.9$ discussed at the beginning of this section. Suppose s is now a probe string. We already established that we need to consider the first two positions; that is, i can be 1 or 2. Suppose $i = 1$. Then $j \leq (10 \times 0.1 - 1 + 1 + 0.9)/0.9$. That is, we only have to compare the symbol `a` with strings in the bucket for $(\mathtt{a}, j)$ if $j \leq 2.11$. Thus, j can be 1 or 2, but nothing higher.

Now suppose $i = 2$. Then we require $j \leq (10 \times 0.1 - 2 + 1 + 0.9)/0.9$, Or $j \leq 1$. We conclude that we must look in the buckets for $(\mathtt{a}, 1)$, $(\mathtt{a}, 2)$, and $(\mathtt{c}, 1)$, but in no other bucket. In comparison, using the buckets of Section 3.9.4, we would look into the buckets for `a` and `c`, which is equivalent to looking to all buckets $(\mathtt{a}, j)$ and $(\mathtt{c}, j)$ for any j. □

3.9.6 Using Position and Length in Indexes

When we considered the upper limit on j in the previous section, we assumed that what follows positions i and j were as in Fig. 3.14, where what followed these positions in strings s and t matched exactly. We do not want to build an index that involves every symbol in the strings, because that makes the total work excessive. However, we can add to our index a summary of what follows the positions being indexed. Doing so expands the number of buckets, but not beyond reasonable bounds, and yet enables us to eliminate many candidate matches without comparing entire strings. The idea is to use index buckets corresponding to a symbol, a position, and the *suffix length*, that is, the number of symbols following the position in question.

Example 3.28: The string $s = \texttt{acdefghijk}$, with $J = 0.9$, would be indexed in the buckets for $(\texttt{a}, 1, 9)$ and $(\texttt{c}, 2, 8)$. That is, the first position of s has symbol a, and its suffix is of length 9. The second position has symbol c and its suffix is of length 8. □

Figure 3.14 assumes that the suffixes for position i of s and position j of t have the same length. If not, then we can either get a smaller upper bound on the size of the intersection of s and t (if t is shorter) or a larger lower bound on the size of the union (if t is longer). Suppose s has suffix length p and t has suffix length q.

Case 1: $p \geq q$. Here, the maximum size of the intersection is

$$L_s - i + 1 - (p - q)$$

Since $L_s = i + p$, we can write the above expression for the intersection size as $q + 1$. The minimum size of the union is $L_s + j - 1$, as it was when we did not take suffix length into account. Thus, we require

$$\frac{q+1}{L_s + j - 1} \geq J$$

whenever $p \geq q$.

Case 2: $p < q$. Here, the maximum size of the intersection is $L_s - i + 1$, as when suffix length was not considered. However, the minimum size of the union is now $L_s + j - 1 + q - p$. If we again use the relationship $L_s = i + p$, we can replace $L_s - p$ by i and get the formula $i + j - 1 + q$ for the size of the union. If the Jaccard similarity is at least J, then

$$\frac{L_s - i + 1}{i + j - 1 + q} \geq J$$

whenever $p < q$.

Example 3.29: Let us again consider the string $s = \texttt{acdefghijk}$, but to make the example show some details, let us choose $J = 0.8$ instead of 0.9. We know that $L_s = 10$. Since $\lfloor (1 - J)L_s \rfloor + 1 = 3$, we must consider prefix positions $i = 1$, 2, and 3 in what follows. As before, let p be the suffix length of s and q the suffix length of t.

First, consider the case $p \geq q$. The additional constraint we have on q and j is $(q + 1)/(9 + j) \geq 0.8$. We can enumerate the pairs of values of j and q for each i between 1 and 3, as follows.

$i = 1$: Here, $p = 9$, so $q \leq 9$. Let us consider the possible values of q:

- **$q = 9$:** We must have $10/(9 + j) \geq 0.8$. Thus, we can have $j = 1$, $j = 2$, or $j = 3$. Note that for $j = 4$, $10/13 > 0.8$.
- **$q = 8$:** We must have $9/(9 + j) \geq 0.8$. Thus, we can have $j = 1$ or $j = 2$. For $j = 3$, $9/12 > 0.8$.
- **$q = 7$:** We must have $8/(9 + j) \geq 0.8$. Only $j = 1$ satisfies this inequality.
- **$q = 6$:** There are no possible values of j, since $7/(9 + j) > 0.8$ for every positive integer j. The same holds for every smaller value of q.

$i = 2$: Here, $p = 8$, so we require $q \leq 8$. Since the constraint $(q+1)/(9+j) \geq 0.8$ does not depend on i,[7] we can use the analysis from the above case, but exclude the case $q = 9$. Thus, the only possible values of j and q when $i = 2$ are

1. $q = 8$; $j = 1$.
2. $q = 8$; $j = 2$.
3. $q = 7$; $j = 1$.

$i = 3$: Now, $p = 7$ and the constraints are $q \leq 7$ and $(q + 1)/(9 + j) \geq 0.8$. The only option is $q = 7$ and $j = 1$.

Next, we must consider the case $p < q$. The additional constraint is

$$\frac{11 - i}{i + j + q - 1} \geq 0.8$$

Again, consider each possible value of i.

$i = 1$: Then $p = 9$, so we require $q \geq 10$ and $10/(q + j) \geq 0.8$. The possible values of q and j are

1. $q = 10$; $j = 1$.
2. $q = 10$; $j = 2$.
3. $q = 11$; $j = 1$.

$i = 2$: Now, $p = 8$, so we require $q \geq 9$ and $9/(q + j + 1) \geq 0.8$. Since j must be a positive integer, the only solution is $q = 9$ and $j = 1$, a possibility that we already knew about.

$i = 3$: Here, $p = 7$, so we require $q \geq 8$ and $8/(q + j + 2) \geq 0.8$. There are no solutions.

When we accumulate the possible combinations of i, j, and q, we see that the set of index buckets in which we must look forms a pyramid. Figure 3.15 shows the buckets in which we must search. That is, we must look in those buckets (x, j, q) such that the ith symbol of the string s is x, j is the position associated with the bucket and q the suffix length. □

[7] Note that i does influence the value of p, and through p, puts a limit on q.

	q	$j = 1$	$j = 2$	$j = 3$
	7	x		
	8	x	x	
$i = 1$	9	x	x	x
	10	x	x	
	11	x		
	7	x		
$i = 2$	8	x	x	
	9	x		
$i = 3$	7	x		

Figure 3.15: The buckets that must be examined to find possible matches for the string $s =$ `acdefghijk` with $J = 0.8$ are marked with an x

3.9.7 Exercises for Section 3.9

Exercise 3.9.1: Suppose our universal set is the lower-case letters, and the order of elements is taken to be the vowels, in alphabetic order, followed by the consonants in reverse alphabetic order. Represent the following sets as strings.

a $\{q, w, e, r, t, y\}$.

(b) $\{a, s, d, f, g, h, j, u, i\}$.

Exercise 3.9.2: Suppose we filter candidate pairs based only on length, as in Section 3.9.3. If s is a string of length 20, with what strings is s compared when J, the lower bound on Jaccard similarity has the following values: (a) $J = 0.85$ (b) $J = 0.95$ (c) $J = 0.98$?

Exercise 3.9.3: Suppose we have a string s of length 15, and we wish to index its prefix as in Section 3.9.4.

(a) How many positions are in the prefix if $J = 0.85$?

(b) How many positions are in the prefix if $J = 0.95$?

! (c) For what range of values of J will s be indexed under its first four symbols, but no more?

Exercise 3.9.4: Suppose s is a string of length 12. With what symbol-position pairs will s be compared with if we use the indexing approach of Section 3.9.5, and (a) $J = 0.75$ (b) $J = 0.95$?

! Exercise 3.9.5: Suppose we use position information in our index, as in Section 3.9.5. Strings s and t are both chosen at random from a universal set of 100 elements. Assume $J = 0.9$. What is the probability that s and t will be compared if

(a) s and t are both of length 9.

(b) s and t are both of length 10.

Exercise 3.9.6: Suppose we use indexes based on both position and suffix length, as in Section 3.9.6. If s is a string of length 20, with what symbol-position-length triples will s be compared with, if (a) $J = 0.8$ (b) $J = 0.9$?

3.10 Summary of Chapter 3

✦ *Jaccard Similarity*: The Jaccard similarity of sets is the ratio of the size of the intersection of the sets to the size of the union. This measure of similarity is suitable for many applications, including textual similarity of documents and similarity of buying habits of customers.

✦ *Shingling*: A k-shingle is any k characters that appear consecutively in a document. If we represent a document by its set of k-shingles, then the Jaccard similarity of the shingle sets measures the textual similarity of documents. Sometimes, it is useful to hash shingles to bit strings of shorter length, and use sets of hash values to represent documents.

✦ *Minhashing*: A minhash function on sets is based on a permutation of the universal set. Given any such permutation, the minhash value for a set is that element of the set that appears first in the permuted order.

✦ *Minhash Signatures*: We may represent sets by picking some list of permutations and computing for each set its minhash signature, which is the sequence of minhash values obtained by applying each permutation on the list to that set. Given two sets, the expected fraction of the permutations that will yield the same minhash value is exactly the Jaccard similarity of the sets.

✦ *Efficient Minhashing*: Since it is not really possible to generate random permutations, it is normal to simulate a permutation by picking a random hash function and taking the minhash value for a set to be the least hash value of any of the set's members.

✦ *Locality-Sensitive Hashing for Signatures*: This technique allows us to avoid computing the similarity of every pair of sets or their minhash signatures. If we are given signatures for the sets, we may divide them into bands, and only measure the similarity of a pair of sets if they are identical in at least one band. By choosing the size of bands appropriately, we can eliminate from consideration most of the pairs that do not meet our threshold of similarity.

✦ *Distance Measures*: A distance measure is a function on pairs of points in a space that satisfy certain axioms. The distance between two points is 0 if the points are the same, but greater than 0 if the points are different. The distance is symmetric; it does not matter in which order we consider the two points. A distance measure must satisfy the triangle inequality: the distance between two points is never more than the sum of the distances between those points and some third point.

✦ *Euclidean Distance*: The most common notion of distance is the Euclidean distance in an n-dimensional space. This distance, sometimes called the L_2-norm, is the square root of the sum of the squares of the differences between the points in each dimension. Another distance suitable for Euclidean spaces, called Manhattan distance or the L_1-norm is the sum of the magnitudes of the differences between the points in each dimension.

✦ *Jaccard Distance*: One minus the Jaccard similarity is a distance measure, called the Jaccard distance.

✦ *Cosine Distance*: The angle between vectors in a vector space is the cosine distance measure. We can compute the cosine of that angle by taking the dot product of the vectors and dividing by the lengths of the vectors.

✦ *Edit Distance*: This distance measure applies to a space of strings, and is the number of insertions and/or deletions needed to convert one string into the other. The edit distance can also be computed as the sum of the lengths of the strings minus twice the length of the longest common subsequence of the strings.

✦ *Hamming Distance*: This distance measure applies to a space of vectors. The Hamming distance between two vectors is the number of positions in which the vectors differ.

✦ *Generalized Locality-Sensitive Hashing*: We may start with any collection of functions, such as the minhash functions, that can render a decision as to whether or not a pair of items should be candidates for similarity checking. The only constraint on these functions is that they provide a lower bound on the probability of saying "yes" if the distance (according to some distance measure) is below a given limit, and an upper bound on the probability of saying "yes" if the distance is above another given limit. We can then increase the probability of saying "yes" for nearby items and at the same time decrease the probability of saying "yes" for distant items to as great an extent as we wish, by applying an AND construction and an OR construction.

✦ *Random Hyperplanes and LSH for Cosine Distance*: We can get a set of basis functions to start a generalized LSH for the cosine distance measure by identifying each function with a list of randomly chosen vectors. We apply a function to a given vector v by taking the dot product of v with each vector on the list. The result is a sketch consisting of the signs ($+1$ or -1) of the dot products. The fraction of positions in which the sketches of two vectors agree, multiplied by 180, is an estimate of the angle between the two vectors.

✦ *LSH For Euclidean Distance*: A set of basis functions to start LSH for Euclidean distance can be obtained by choosing random lines and projecting points onto those lines. Each line is broken into fixed-length intervals, and the function answers "yes" to a pair of points that fall into the same interval.

✦ *High-Similarity Detection by String Comparison*: An alternative approach to finding similar items, when the threshold of Jaccard similarity is close to 1, avoids using minhashing and LSH. Rather, the universal set is ordered, and sets are represented by strings, consisting their elements in order. The simplest way to avoid comparing all pairs of sets or their strings is to note that highly similar sets will have strings of approximately the same length. If we sort the strings, we can compare each string with only a small number of the immediately following strings.

✦ *Character Indexes*: If we represent sets by strings, and the similarity threshold is close to 1, we can index all strings by their first few characters. The prefix whose characters must be indexed is approximately the length of the string times the maximum Jaccard distance (1 minus the minimum Jaccard similarity).

✦ *Position Indexes*: We can index strings not only on the characters in their prefixes, but on the position of that character within the prefix. We reduce the number of pairs of strings that must be compared, because if two strings share a character that is not in the first position in both strings, then we know that either there are some preceding characters that are in the union but not the intersection, or there is an earlier symbol that appears in both strings.

✦ *Suffix Indexes*: We can also index strings based not only on the characters in their prefixes and the positions of those characters, but on the length of the character's suffix – the number of positions that follow it in the string. This structure further reduces the number of pairs that must be compared, because a common symbol with different suffix lengths implies additional characters that must be in the union but not in the intersection.

3.11 References for Chapter 3

The technique we called shingling is attributed to [10]. The use in the manner we discussed here is from [2]. Minhashing comes from [3]. The original works on locality-sensitive hashing were [9] and [7]. [1] is a useful summary of ideas in this field.

[4] introduces the idea of using random-hyperplanes to summarize items in a way that respects the cosine distance. [8] suggests that random hyperplanes plus LSH can be more accurate at detecting similar documents than minhashing plus LSH.

Techniques for summarizing points in a Euclidean space are covered in [6]. [11] presented the shingling technique based on stop words.

The length and prefix-based indexing schemes for high-similarity matching comes from [5]. The technique involving suffix length is from [12].

1. A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Comm. ACM* **51**:1, pp. 117–122, 2008.

2. A.Z. Broder, "On the resemblance and containment of documents," *Proc. Compression and Complexity of Sequences*, pp. 21–29, Positano Italy, 1997.

3. A.Z. Broder, M. Charikar, A.M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *ACM Symposium on Theory of Computing*, pp. 327–336, 1998.

4. M.S. Charikar, "Similarity estimation techniques from rounding algorithms," *ACM Symposium on Theory of Computing*, pp. 380–388, 2002.

5. S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," *Proc. Intl. Conf. on Data Engineering*, 2006.

6. M. Datar, N. Immorlica, P. Indyk, and V.S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," *Symposium on Computational Geometry* pp. 253–262, 2004.

7. A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," *Proc. Intl. Conf. on Very Large Databases*, pp. 518–529, 1999.

8. M. Henzinger, "Finding near-duplicate web pages: a large-scale evaluation of algorithms," *Proc. 29th SIGIR Conf.*, pp. 284–291, 2006.

9. P. Indyk and R. Motwani. "Approximate nearest neighbor: towards removing the curse of dimensionality," *ACM Symposium on Theory of Computing*, pp. 604–613, 1998.

10. U. Manber, "Finding similar files in a large file system," *Proc. USENIX Conference*, pp. 1–10, 1994.

11. M. Theobald, J. Siddharth, and A. Paepcke, "SpotSigs: robust and efficient near duplicate detection in large web collections," *31st Annual ACM SIGIR Conference*, July, 2008, Singapore.

12. C. Xiao, W. Wang, X. Lin, and J.X. Yu, "Efficient similarity joins for near duplicate detection," *Proc. WWW Conference*, pp. 131-140, 2008.

2 Representation

Learning Objectives

After reading this chapter, you will

- Know that patterns can be represented as
  - Strings
  - Logical descriptions
  - Fuzzy and rough sets
  - Trees and graphs
- Have learnt how to classify patterns using proximity measures like
  - Distance measure
  - Non-metrics which include
    - k-median distance
    - Hausdorff distance
    - Edit distance
    - Mutual neighbourhood distance
    - Conceptual cohesiveness
    - Kernel functions
- Have found out what is involved in abstraction of data
- Have discovered the meaning of feature extraction
- Know the advantages of feature selection and the different approaches to it
- Know the parameters involved in evaluation of classifiers
- Understand the need to evaluate the clustering accomplished

A pattern is a physical object or an abstract notion. If we are talking about the classes of animals, then a description of an animal would be a pattern. If we are talking about various types of balls, then a description of a ball (which may include the size and material of the ball) is a pattern. These patterns are represented by a set of descriptions. Depending on the classification problem, distinguishing features of the patterns are used. These features are called attributes. A pattern is the representation of an object by the values taken by the attributes. In the classification problem, we have a set of objects for which the values of the attributes are known. We have a set of classes and each object belongs to one of these classes. The classes for the case where the patterns are animals could be mammals, reptiles etc. In the case of the patterns being balls, the classes could be football, cricket ball, table tennis ball etc. Given a new pattern, the class of the pattern is to be determined. The choice of attributes and representation of patterns is a very important step in pattern classification. A good representation is one which makes use of discriminating attributes and also reduces the computational burden in pattern classification.

2.1 Data Structures for Pattern Representation

2.1.1 Patterns as Vectors

An obvious representation of a pattern will be a vector. Each element of the vector can represent one attribute of the pattern. The first element of the vector will contain the value of the first attribute for the pattern being considered. For example, if we are representing spherical objects, (30, 1) may represent a spherical object with 30 units of weight and 1 unit diameter. The class label can form part of the vector. If spherical objects belong to class 1, the vector would be (30, 1, 1), where the first element represents the weight of the object, the second element, the diameter of the object and the third element represents the class of the object.

EXAMPLE 1

Using the vector representation, a set of patterns, for example can be represented as

1.0, 1.0, 1 ;	1.0, 2.0, 1
2.0, 1.0, 1 ;	2.0, 2.0, 1
4.0, 1.0, 2 ;	5.0, 1.0, 2
4.0, 2.0, 2 ;	5.0, 2.0, 2
1.0, 4.0, 2 ;	1.0, 5.0, 2
2.0, 4.0, 2 ;	2.0, 5.0, 2
4.0, 4.0, 1 ;	5.0, 5.0, 1
4.0, 5.0, 1 ;	5.0, 4.0, 1

where the first element is the first feature, the second element is the second feature and the third element gives the class of the pattern. This can be represented graphically as shown in Figure 2.1, where patterns of class 1 are represented using the symbol +, patterns of class 2 are represented using X and the square represents a test pattern.

2.1.2 Patterns as Strings

The string may be viewed as a sentence in a language, for example, a DNA sequence or a protein sequence.

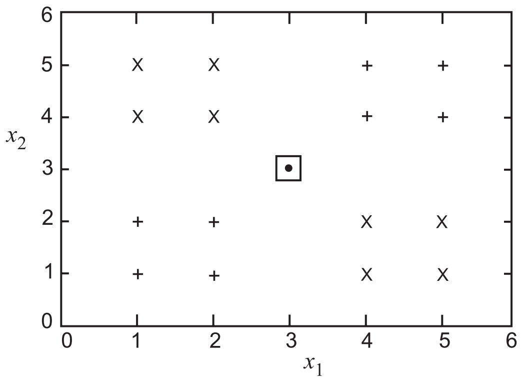


Figure 2.1 Example data set

As an illustration, a gene can be defined as a region of the chromosomal DNA constructed with four nitrogenous bases: adenine, guanine, cytosine and thymine, which are referred to by A, G, C and T respectively. The gene data is arranged in a sequence such as

GAAGTCCAG...

2.1.3 Logical Descriptions

Patterns can be represented as a logical description of the form

$$(x_1 = a_1..a_2) \wedge (x_2 = b_1..b_2) \wedge ...$$

where x_1 and x_2 are the attributes of the pattern and a_i and b_i are the values taken by the attribute.

This description actually consists of a conjunction of logical disjunctions. An example of this could be

$$(\text{colour} = \text{red} \vee \text{white}) \wedge (\text{make} = \text{leather}) \wedge (\text{shape} = \text{sphere})$$

to represent a cricket ball.

2.1.4 Fuzzy and Rough Pattern Sets

Fuzziness is used where it is not possible to make precise statements. It is therefore used to model subjective, incomplete and imprecise data. In a fuzzy set, the objects belong to the set depending on a membership value which varies from 0 to 1.

A rough set is a formal approximation of a crisp set in terms of a pair of sets which give the lower and the upper approximation of the original set. The lower and upper approximation sets themselves are crisp sets. The set X is thus represented by a tuple $\{\underline{X}, \overline{X}\}$ which is composed of the lower and upper approximation. The lower approximation of X is the collection of objects which can be classified with full certainty as members of the set X. Similarly, the upper approximation of X is the collection of objects that may possibly be classified as members of the set X.

The features of the fuzzy pattern may be a mixture of linguistic values, fuzzy numbers, intervals and real numbers. Each pattern X will be a vector consisting of linguistic values, fuzzy numbers, intervals and real values. For example, we may have linguistic knowledge such as "If X_1 is small and X_2 is large, then class 3". This would lead to the pattern (small, large) which has the class label 3. Fuzzy patterns can also be used in cases where there are uncertain or missing values. For example, the pattern maybe $X = (?, 6.2, 7)$. The missing value can be represented as an interval which includes its possible values. If the missing value in the above example lies in the interval [0,1], then the pattern can be represented as

$X = ([0, 1], 6.2, 7)$ with no missing values.

The values of the features may be rough values. Such feature vectors are called rough patterns. A rough value consists of an upper and a lower bound. A rough value can be used to effectively represent a range of values of the feature. For example, power may be represented as (230, 5.2, $(\underline{50}, \overline{49, 51})$), where the three features are voltage, current and frequency (represented by a lower and upper bound).

In some cases, hard class labels do not accurately reflect the nature of the available information. It may be that the pattern categories are ill-defined and best represented as fuzzy sets of patterns. Each training vector x_i is assigned a fuzzy label $u_i \in [0, 1]^c$ whose components are the grades of membership of that pattern to each class.

The classes to which the patterns belong may be fuzzy concepts, for example, the classes considered may be short, medium and tall.

2.1.5 Patterns as Trees and Graphs

Trees and graphs are popular data structures for representing patterns and pattern classes. Each node in the tree or graph may represent one or more patterns. The minimum spanning tree (MST), the Delauney tree (DT), the R-tree and the k-d tree are examples of this. The R-tree represents patterns in a tree structure which splits space into hierarchically nested and possibly overlapping minimum bounding rectangles or bounding boxes. Each node of an R-tree has a number of entries. A non-leaf node stores a way of identifying the node and the bounding box of all entries of nodes which are its descendants. Some of the important operations on an R-tree are appropriate updation (insertion, deletion) of the tree to reflect the necessary changes and searching of the tree to locate the nearest neighbours of a given pattern. Insertion and deletion algorithms use the bounding boxes from the nodes to ensure that the nearby elements are placed in the same leaf node. Search entails using the bounding boxes to decide whether or not to search inside a node. In this way most of the nodes in the tree need not be searched.

A set of patterns can be represented as a graph or a tree where following a path in the tree gives one of the patterns in the set. The whole pattern set is represented as a single tree. An example of this is the frequent pattern (FP) tree.

Minimum Spanning Tree

Each pattern is represented as a point in space. These points can be connected to form a tree called the minimum spanning tree (MST). A tree which covers all the nodes in a graph is called a spanning tree. If $d(X, Y)$ is the distance or dissimilarity between nodes X and Y, an MST is a spanning tree where the sum of the distances of the links (edges in the tree) is the minimum. Figure 2.2 shows an example of a minimum spanning tree.

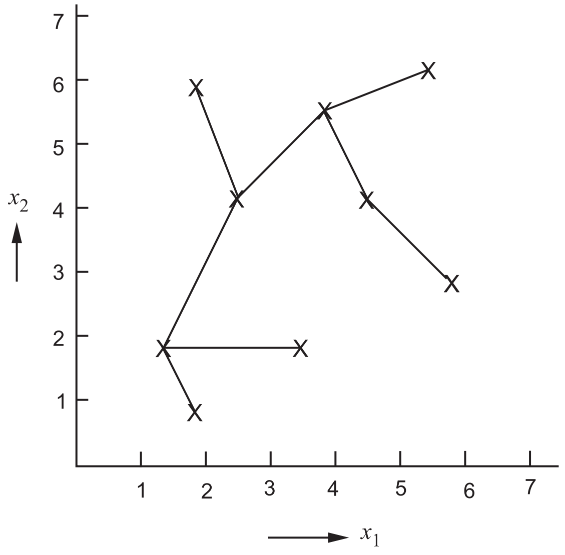


Figure 2.2 Example of a minimum spanning tree

The minimum spanning tree can be used for clustering data points. This is illustrated with the following example.

EXAMPLE 2

In Figure 2.3, 8 patterns are considered. Figure 2.4 gives the minimum spanning tree for the 8 patterns.

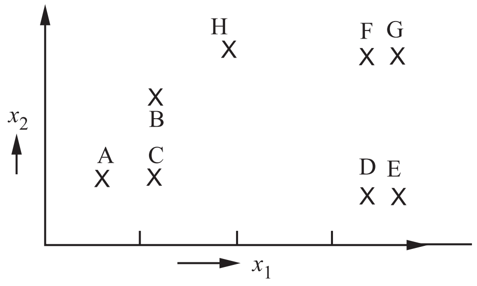


Figure 2.3 Patterns represented in feature space

The minimum spanning tree is used for clustering applications. The largest links in the MST are deleted to obtain the clusters. In Figure 2.4, if the largest link FD is deleted, it results in the formation of two clusters. The first cluster has points A, B, C, H, F and G, and the second cluster has the points D and E. In the first cluster, if the largest link HF is deleted, three clusters are formed. The first cluster has points A, B, C and H; the second cluster has points F and G; and the third cluster has points D and E.

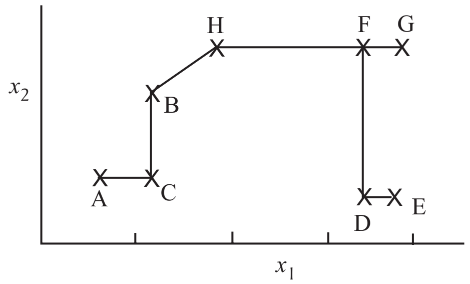


Figure 2.4 The MST for Figure 2.3

Frequent Pattern Trees

This data structure is basically used in transaction databases. The frequent pattern tree (FP tree) is generated from the transactions in the database. It is a compressed tree structure which is useful in finding associations between items in a transaction database. This means that the presence of some items in a transaction will probably imply the presence of other items in the same transaction. The FP growth algorithm used for efficient mining of frequent item sets in large databases uses this data structure.

The first step in constructing this tree is to determine the frequency of every item in the database and sort them from largest to smallest frequencies. Then each entry in the database is ordered so that the order matches the frequency just calculated from largest to smallest. The root of the FP tree is first created and labelled null. The first transaction is taken and the first branch of the FP tree is constructed according to the ordered sequence. The second transaction is then added according to the ordering done. The common prefix shared by this transaction with the previous transaction will follow the existing path, only incrementing the count by one. For the remaining part of the transaction, new nodes are created. This is continued for the whole database. Thus it can be seen that the FP tree is a compressed tree which has information about the frequent items in the original database.

EXAMPLE 3

Consider a 4 $\times$ 4 square, where each of the squares is a pixel to represent a digit. The squares are assigned an alphabet as shown in Figure 2.5. For example, digit 4 would be represented in the 4 $\times$ 4 square as shown in Figure 2.6 and denoted by a, e, g, i, j, k, l, o. The digits 0, 1, 7, 9 and 4 can be represented as shown in Table 2.1.

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

Figure 2.5 Square representing the pixels of a digit

1			
1		1	
1	1	1	1
		1	

Figure 2.6 Square representing the pixels of digit 4

Table 2.1 A transaction database

Digit	Transaction
0	a, d, e, h, i, l, m, p, b, c, n, o
1	d, h, l, p
7	a, b, c, d, h, l, p
9	a, b, c, d, i, j, k, l, p, e, h
4	a, e, g, i, j, k, l, o

A scan of the transaction database in Table 2.1 will yield the frequency of every item which when sorted from largest to smallest gives (l : 5), (a : 4), (d: 4), (p: 4), (h: 3), (i: 3), (c: 3), (e: 3). Only items which have a frequency of 3 and above are listed here. Note that ties are resolved arbitrarily.

Table 2.2 The transaction database ordered according to frequency of items

Sl.No.	Transaction
0	l, a, d, p, h, i, c, e
1	l, d, p, h
7	l, a, d, p, h
9	l, a, d, p, i, e
4	l, a, i

Table 2.2 shows the transaction database ordered according to the frequency of items. Items which have a support below a threshold are weeded out. In this case, items which have support of two or below are left out. Thus, e, m, b, c, j, k, g, f, n and o are removed. The items retained are l, a, d, p, h and i. Using the ordered database shown in Table 2.2, the FP tree given in Figure 2.7 is constructed.

The root node points to items which are the starting items of the transactions. Here, since all transactions start with l, the root node points to l. For the first transaction, the link is made from the root node to l, from l to a, from a to d, from d to p, from p to h and from h to j. A count of 1 is stored for each of these items. The second transaction is then processed. From the root node, it moves to node l, which already exists. Its count is increased by 1. Since node d is not the next node after l, another link is made from l to d, and from d to p and from p to h. The count for l will be 2, the count for d, p and h will be 1. The next transaction moves from root node to l and from l to a and then to d and then to p and then to h along the path which already exists. The count of the items along this path is increased by 1 so that the count for l will be 3 and the count for a, d, p and h will become 2. Taking the transaction for digit 9, there is a path from root node to l, to p passing through a and d. From p, a new link is made to node i. Now the count for l will be 4 and the count for a, d and p will be 3 and the count for i will be 1. For the last transaction, the path is from root node to the already existing l and a, so that the count of l will become 5 and the count for a will be 4. Then a new link is made from a to i, giving a count of 1 to i. As can be seen from Figure 2.7, the header node for i points to all the nodes with item i. Similarly, the header node for d, p and h also point to all the nodes having these items.

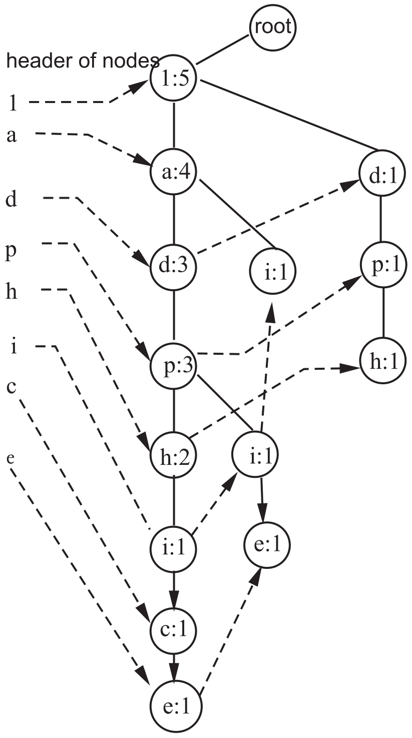


Figure 2.7 FP tree for the transaction database in Table 2.1

2.2 Representation of Clusters

Clustering refers to the process of grouping patterns with similar features together and placing objects with different features in separate groups. There are two data structures here. One is the partition P of the patterns and the other is the set of cluster representatives C.

In problems where the centroid is used as the representative of a group of patterns, P and C depend upon each other. So, if one of them is given, the other can be calculated. If the cluster centroids are given, any pattern can be optimally classified by assigning it to the cluster whose centroid is closest to the pattern. Similarly, if the partition is given, the centroids can be computed. If P is given, then C can be computed in $O(N)$ running time if there are N patterns. If C is given, then P can be generated in $O(NM)$ time if there are M centroids.

Clusters can therefore be represented either by P or C (in the case where the centroid is used as the representative of a group of patterns) or by both P and C.

2.3 Proximity Measures

In order to classify patterns, they need to be compared against each other and against a standard. When a new pattern is present and it is necessary to classify it, the proximity of this pattern to the patterns in the training set is to be found. Even in the case of unsupervised learning, it is required to find some groups in the data so that patterns which are similar are put together. A number of similarity and dissimilarity measures can be used. For example, in the case of the nearest neighbour classifier, a training pattern which is closest to the test pattern is to be found.

2.3.1 Distance Measure

A distance measure is used to find the dissimilarity between pattern representations. Patterns which are more similar should be closer. The distance function could be a metric or a non-metric. A metric is a measure for which the following properties hold :

1. *Positive reflexivity* $d(x, x) = 0$
2. *Symmetry* $d(x, y) = d(y, x)$
3. *Triangular inequality* $d(x, y) \leq d(x, z) + d(z, y)$

The popularly used distance metric called the Minkowski metric is of the form

$$d^m(X, Y) = \left(\sum_{k=1}^{d} \mid x_k - y_k \mid^m \right)^{\frac{1}{m}}$$

When m is 1 it is called the Manhattan distance or the L_1 distance. The most popular is the Euclidean distance or the L_2 distance which occurs when m is assigned the value of 2. We then get

$$d^2(X, Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ...(x_d - y_d)^2}$$

In this way, L_∞ will be

$$d^\infty(X, Y) = \max_{k=1,..,d} \mid x_k - y_k \mid$$

While using the distance measure, it should be ensured that all the features have the same range of values, failing which attributes with larger ranges will be treated as more important. It will be like giving it a larger weightage. To ensure that all features are in the same range, normalisation of the feature values has to be carried out.

The Mahalanobis distance is also a popular distance measure used in classification. It is given by

$$d(X,Y)^2 = (X-Y)^T \Sigma^{-1} (X-Y)$$

where Σ is the covariance matrix.

Example 4

If $X=(4, 1, 3)$ and $Y=(2, 5, 1)$ then the Euclidean distance will be

$$d(X,Y) = \sqrt{(4-2)^2 + (1-5)^2 + (3-1)^2} = 4.9$$

2.3.2 Weighted Distance Measure

When attributes need to treated as more important, a weight can be added to their values. The weighted distance metric is of the form

$$d(X,Y) = \left(\sum_{k=1}^{d} w_k \times (x_k - y_k)^m \right)^{\frac{1}{m}}$$

where w_k is the weight associated with the kth dimension (or feature).

Example 5

If $X=(4, 2, 3)$, $Y=(2, 5, 1)$ and $w_1 = 0.3$, $w_2 = 0.6$ and $w_3 = 0.1$, then

$$d^2(X,Y) = \sqrt{0.3 \times (4-2)^2 + 0.6 \times (1-5)^2 + 0.1 \times (3-1)^2} = 3.35$$

The weights reflects the importance given to each feature. In this example, the second feature is more important than the first and the third feature is the least important.

It is possible to view the Mahalanobis distance as a weighted Euclidean distance, where the weighting is determined by the range of variability of the sample point expressed by the covariance matrix, where σ_i^2 is the variance in the ith feature direction, i = 1, 2. For example, if

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

the Mahalanobis distance gives the Euclidean distance weighted by the inverse of the variance in each dimension.

Another distance metric is the Hausdorff distance. This is used when comparing two shapes as shown in Figure 2.8. In this metric, the points sampled along the shapes boundaries are compared. The Hausdorff distance is the maximum distance between any point in one shape and the point that is closest to it in the other. If there are two point sets (I) and (J), then the Hausdorff distance is

$$\max(\max_{i \in I} \min_{j \in J} \parallel i - j \parallel, \max_{j \in J} \min_{i \in I} \parallel i - j \parallel)$$

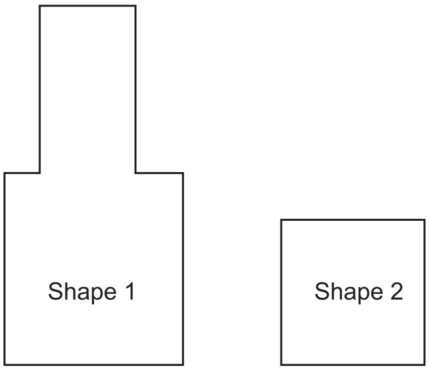


Figure 2.8 Shapes which can be compared by the Hausdorff distance

2.3.3 Non-Metric Similarity Function

Similarity functions which do not obey either the triangular inequality or symmetry come under this category. Usually these similarity functions are useful for images or string data. They are robust to outliers or to extremely noisy data. The squared Euclidean distance is itself an example of a non-metric, but it gives the same ranking as the Euclidean distance which is a metric. One non-metric similarity function is the k-median distance between two vectors. If $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$, then

$$d(X, Y) = k\text{-median}\{|x_1 - y_1|, ..., |x_n - y_n|\}$$

where the k-median operator returns the kth value of the ordered difference vector.

EXAMPLE 6

If X = (50, 3, 100, 29, 62, 140) and Y = (55, 15, 80, 50, 70, 170), then

Difference vector = {5, 12, 20, 21, 8, 30}

$$d(X, Y) = k\text{-median}\{5, 8, 12, 20, 21, 30\}$$

If $k = 3$, then $d(X, Y) = 12$

Another measure of similarity between two patterns X and Y is

$$S(X, Y) = \frac{X^t Y}{||X||\,||Y||}$$

This corresponds to the cosine of the angle between X and Y. $S(X, Y)$ is the similarity between X and Y. If we view $1 - S(X, Y)$ as the distance, $d(X, Y)$, between X and Y, then $d(X, Y)$ does not satisfy the triangular inequality, it is not a metric. However, it is symmetric, because $\cos(\theta) = \cos(-\theta)$.

Example 7

If X, Y, and Z are two vectors in a 2-d space such that the angle between X and Y is 45 and that between Y and Z is 45, then

$$d(X, Z) = 1 - 0 = 1$$

whereas $d(X, Y) + d(Y, Z) = 2 - \sqrt{(2)} = 0.586$

So, triangular inequality is violated.

A non-metric which is non-symmetric is the Kullback–Leibler distance (KL distance). This is the natural distance function from a ‘‘true’’ probability distribution p, to a ‘‘target’’ probability distribution q. For discrete probability distributions if $p = \{p_1, ..., p_n\}$ and $q = \{q_1, ..., q_n\}$, then the KL distance is defined as

$$\text{KL}(p, q) = \Sigma_i p_i \log_2 \left(\frac{p_i}{q_i}\right)$$

For continuous probability densities, the sum is replaced by an integral.

2.3.4 Edit Distance

Edit distance measures the distance between two strings. It is also called the Levenshtein distance. The edit distance between two strings s_1 and s_2 is defined as the minimum number of point mutations required to change s_1 to s_2. A point mutation involves any one of the following operations.

1. Changing a letter
2. Inserting a letter
3. Deleting a letter

The following recurrence relation defines the edit distance between two strings

$$\begin{array}{lll} d(\text{“ ”},\text{“ ”}) & = & 0 \\ d(s,\text{“ ”}) & = & d(\text{“ ”}, s) = \|s\| \end{array}$$

$$d(s_1 + ch_1, s_2 + ch_2) = \min(d(s_1, s_2) + \{\text{if } ch_1 = ch_2 \text{ then } 0 \text{ else } 1\},$$
$$d(s_1 + ch_1, s_2) + 1, d(s_1, s_2 + ch_2) + 1)$$

If the last characters of the two strings ch_1 and ch_2 are identical, they can be matched for no penalty and the overall edit distance is $d(s_1, s_2)$. If ch_1 and ch_2 are different, then ch_1 can be changed into ch_2, giving an overall cost of $d(s_1, s_2) + 1$. Another possibility is to delete ch_1 and edit s_1 into $s_2 + ch_2$, i.e., $d(s_1, s_2 + ch_2) + 1$. The other possibility is $d(s_1 + ch_1, s_2) + 1$. The minimum value of these possibilities gives the edit distance.

EXAMPLE 8

1. If s = “TRAIN” and t = “BRAIN”, then edit distance = 1 because using the recurrence relation defined earlier, this requires a change of just one letter.
2. If s = “TRAIN” and t = “CRANE”, then edit distance = 3. We can write the edit distance from s to t to be the edit distance between “TRAI” and “CRAN” + 1 (since N and E are not the same). It would then be the edit distance between “TRA” and “CRA” + 2 (since I and N are not the same). Proceeding in this way, we get the edit distance to be 3.

2.3.5 Mutual Neighbourhood Distance (MND)

The similarity between two patterns A and B is

$$S(A, B) = f(A, B, \epsilon)$$

where ϵ is the set of neighbouring patterns. ϵ is called the *context* and corresponds to the surrounding points. With respect to each data point, all other data points are numbered from 1 to $N - 1$ in increasing order of some distance measure, such that the nearest neighbour is assigned value 1 and the farthest point is assigned the value $N - 1$. If we denote by NN(u,v), the number of data point v with respect to u, the mutual neighbourhood distance (MND), is defined as

$$\text{MND(u,v)} = \text{NN(u,v)} + \text{NN(v,u)}$$

This is symmetric and by defining NN(u,u)= 0, it is also reflexive. However, the triangle inequality is not satisfied and therefore MND is not a metric.

Example 9

Consider Figure 2.9. In Figure 2.9(a), the ranking of the points A, B and C can be represented as

	1	2
A	B	C
B	A	C
C	B	A

MND(A, B) = 2
MND(B, C) = 3
MND(A, C) = 4

In Figure 2.9(b), the ranking of the points A, B, C, D, E and F can be represented as

	1	2	3	4	5
A	D	E	F	B	C
B	A	C	D	E	F
C	B	A	D	E	F

MND(A, B) = 5
MND(B, C) = 3
MND(A, C) = 7

It can be seen that in the first case, the least MND distance is between A and B, whereas in the second case, it is between B and C. This happens by changing the context.

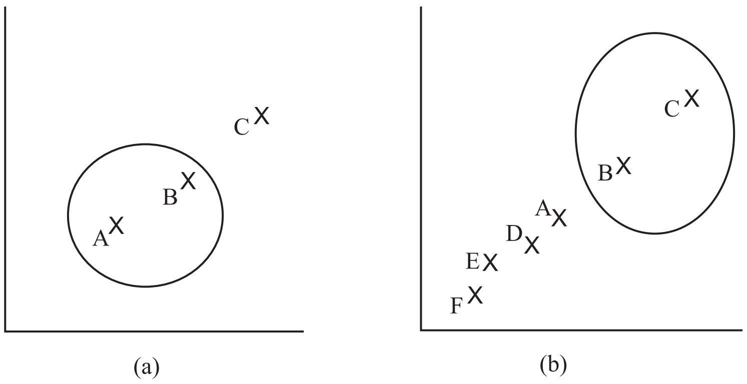


Figure 2.9 Mutual neighbourhood distance

2.3.6 Conceptual Cohesiveness

In this case, distance is applied to pairs of objects based on a set of concepts. A concept is a description of a class of objects sharing some attribute value. For example, (colour = blue) is a concept that represents a collection of blue objects. Conceptual cohesiveness uses the domain knowledge in the form of a set of concepts to characterise the similarity. The conceptual cohesiveness (similarity function) between A and B is characterised by

$S(A, B) = f(A, B, \epsilon, \mathcal{C})$, where $\mathcal{C}$ is a set of pre-defined concepts.

The notion of conceptual distance combines both symbolic and numerical methods. To find the conceptual distance between patterns A and B, A and B are generalised and the similar and dissimilar predicates will give the similarity $S(A, B, G)$ and $D(A, B, G)$. This depends on the generalisation $G(A, B)$. The distance function $f(A, B, G)$ is given by

$$f(A, B, G) = \frac{D(A, B, G)}{S(A, B, G)}$$

The generalisation is not unique—there may be other generalisations. So we can get $S(A, B, G')$ and $D(A, B, G')$ for another generalisation $G'(A, B)$, giving the distance function

$$f(A, B, G') = \frac{D(A, B, G')}{S(A, B, G')}$$

The minimum of these distance functions gives the conceptual distance. The reciprocal of the conceptual distance is called the *conceptual cohesiveness*.

2.3.7 Kernel Functions

The kernel function can be used to characterise the distance between two patterns x and y.

1. *Polynomial kernel* The similarity between x and y can be represented using the polynomial kernel function

   $$K(x, y) = \phi(x)^t \phi(y) = (x^t y + 1)^2$$

   where $\phi(x) = (x_1^2, x_2^2, 1, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2)$

   By using this, linearly dependent vectors in the input space get transformed to independent vectors in kernel space.

2. *Radial basis function kernel* Using the RBF kernel,

$$K(x,y) = \exp^{\frac{-||x-y||^2}{2\sigma^2}}$$

The output of this kernel depends on the Euclidean distance between x and y. Either x or y will be the centre of the radial basis function and σ will determine the area of influence over the data space.

2.4 Size of Patterns

The size of a pattern depends on the attributes being considered. In some cases, the length of the patterns may be a variable. For example, in document retrieval, documents may be of different lengths. This can be handled in different ways.

2.4.1 Normalisation of Data

This process entails normalisation so that all patterns have the same dimensionality. For example, in the case of document retrieval, a fixed number of keywords can be used to represent the document. Normalisation can also be done to give the same importance to every feature in a data set of patterns.

EXAMPLE 10

Consider a data set of patterns with two features as shown below :

X_1	:	(2, 120)
X_2	:	(8, 533)
X_3	:	(1, 987)
X_4	:	(15, 1121)
X_5	:	(18, 1023)

Here, each line corresponds to a pattern. The first value represents the first feature and the second value represents the second feature. The first feature has values below 18, whereas the second feature is much larger. If these values are used in this way for computing distances, the first feature will be insignificant and will not have any bearing on the classification. Normalisation gives equal importance to every feature. Dividing every value of the first feature by its maximum value, which is 18, and dividing every value of the second feature by its maximum value, which is 1121, will make all the values lie between 0 and 1 as shown below :

$X_1^{'}$	:	(0.11, 0.11)
$X_2^{'}$	:	(0.44, 0.48)

$X_3^{'}$: (0.06, 0.88)
$X_4^{'}$: (0.83, 1.0)
$X_5^{'}$: (1.0, 0.91)

2.4.2 Use of Appropriate Similarity Measures

Similarity measures can deal with unequal lengths. One such similarity measure is the edit distance.

2.5 Abstractions of the Data Set

In supervised learning, a set of training patterns where the class label for each pattern is given, is used for classification. The complete training set may not be used because the processing time may be too long but an abstraction of the training set can be used. For example, when using the nearest neighbour classifier on a test pattern with n training patterns, the effort involved will be $O(n)$. If m patterns are used and $(m << n)$, $O(m)$ effort will be sufficient. Depending on the abstraction, different classifiers are used.

1. *No abstraction of patterns* All the training patterns are used and abstraction of the data is not carried out. Examples of classifiers used here are the nearest neighbour (NN), the k-nearest neighbour (kNN) and the modified kNN (MkNN) classifiers. These methods use the neighbour(s) of the test pattern from all the training patterns.
2. *Single representative per class* All the patterns belonging to a class are represented by a single representative pattern. This single representative can be obtained in many ways. One option is to take the mean of all the patterns in the class. One could also take the medoid of all the samples. Here by medoid, we mean the most centrally located pattern. Other methods of obtaining a single representative, which may be typical of the domain to which the patterns belong, can also be used. A test pattern would be compared to the centroid of the patterns belonging to each class and classified as belonging to the class of the mean closest to it. This is the minimum distance classifier (MDC) .
3. *Multiple representatives per class*

   (a) *Cluster representatives as abstractions* The training patterns in the class are clustered and the cluster centre of each cluster is the representative of all the patterns in the cluster. The set of cluster centres is an abstraction of the whole training set.

(b) *Support vectors as representatives* Support vectors are determined for the class and used to represent the class. Support vector machines (SVMs) are described in Chapter 7.

(c) *Frequent item set based abstraction* In the case of transaction data bases, each pattern represents a transaction. An example of this is the departmental store where each transaction is the set of items bought by one customer. These transactions or patterns are of variable length. Item sets which occur frequently are called frequent item sets. If we have a threshold α, item sets which occur more than α times in the data set are the frequent item sets. Frequent item sets are an abstraction of the transaction database. An important observation is that any discrete-valued pattern may be viewed as a transaction.

EXAMPLE 11

In Figure 2.10, the cluster of points can be represented by its centroid or its medoid. Centroid stands for the sample mean of the points in the cluster. It need not coincide with one of the points in the cluster. The medoid is the most centrally located point in the cluster. Use of the centroid or medoid to represent the cluster is an example of using a single representative per class. It is possible to have more than one representative per cluster. For example, four extreme

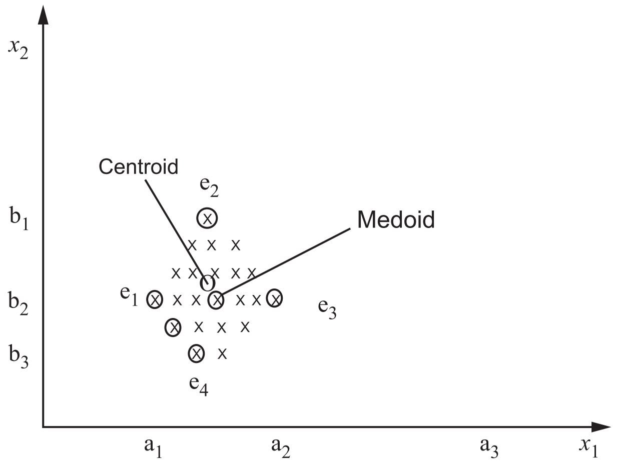


Figure 2.10 A set of data points

points labelled e_1, e_2, e_3, e_4 can represent the cluster as shown in Figure 2.10. When one representative point is used to represent the cluster, in the case of the nearest neighbour classification, only one distance needs to be computed from a test point instead of, say in our example, 22 distances. In the case of there being more than one representative pattern, if four representative patterns are there, only four distances need to be computed instead of 22 distances.

2.6 Feature Extraction

Feature extraction involves detecting and isolating various desired features of patterns. It is the operation of extracting features for identifying or interpreting meaningful information from the data. This is especially relevant in the case of image data where feature extraction involves automatic recognition of various features. Feature extraction is an essential pre-processing step in pattern recognition.

2.6.1 Fisher's Linear Discriminant

Fisher's linear discriminant projects high-dimensional data onto a line and performs classification in this space. If there are two classes, the projection maximises the distance between the means and minimises the variance within each class. Fisher's criterion which is maximised over all linear projections V can be defined as :

$$J(\mathrm{V}) = \frac{|\,\mathrm{mean}_1 - \mathrm{mean}_2|^2}{s_1^2 + s_2^2}$$

where mean_1 and mean_2 represent the mean of Class 1 patterns and Class 2 patterns respectively and s^2 is proportional to the variance. Maximising this criterion yields a closed form solution that involves the inverse of a covariance matrix.

In general, if x_i is a set of N column vectors of dimension D, the mean of the data set is

$$\mathrm{mean} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

In case of multi-dimensional data, the mean is a vector of length D, where D is the dimension of the data.

If there are K classes $\{C_1, C_2, ..., C_K\}$, the mean of class C_k containing N_k members is

$$\text{mean}_k = \frac{1}{N_k} \sum_{x_i \in C_k} x_i$$

The between class scatter matrix is

$$\sigma_B = \sum_{k=1}^{K} N_k(\text{mean}_k - \text{mean})(\text{mean}_k - \text{mean})^T$$

The within class scatter matrix is

$$\sigma_W = \sum_{k=1}^{K} \sum_{x_i \in C_k} (x_i - \text{mean}_k)(x_i - \text{mean}_k)^T$$

The transformation matrix that re-positions the data to be most separable is

$$J(V) = \frac{V^T \sigma_B V}{V^T \sigma_W V}$$

$J(V)$ is the criterion function to be maximised. The vector V that maximises $J(V)$ can be shown to satisfy

$$\sigma_B V = \lambda \sigma_W V$$

Let $\{v_1, v_2, ..., v_D\}$ be the generalised eigenvectors of σ_B and σ_W.

This gives a projection space of dimension D. A projection space of dimension $d < D$ can be defined using the generalised eigenvectors with the largest d eigenvalues to give $V_d = [v_1, v_2, ..., v_d]$.

The projection of vector x_i into a sub-space of dimension d is $y = V_d^T x$. In the case of the two-class problem,

$$\text{mean}_1 = \frac{1}{N_1} \sum_{x_i \in C_1} x_i$$

$$\text{mean}_2 = \frac{1}{N_2} \sum_{x_i \in C_2} x_i$$

$$\sigma_B = N_1(\text{mean}_1 - \text{mean})(\text{mean}_1 - \text{mean})^T + N_2(\text{mean}_2 - \text{mean})(\text{mean}_2 - \text{mean})^T$$

$$\sigma_W = \sum_{x_i \in C_1} (x_i - \text{mean}_1)(x_i - \text{mean}_1)^T + \sum_{x_i \in C_2} (x_i - \text{mean}_2)(x_i - \text{mean}_2)^T$$

$$\sigma_B V = \lambda \sigma_W V$$

This means that

$$\sigma_W^{-1}\sigma_B V = \lambda V$$

Since $\sigma_B V$ is always in the direction of $\text{mean}_1 - \text{mean}_2$, the solution for V is :

$$V = \sigma_W^{-1}(\text{mean}_1 - \text{mean}_2)$$

The intention here is to convert a d-dimensional problem to a one-dimensional one.

Example 12

If there are six points namely $(2,2)^t$, $(4,3)^t$ and $(5,1)^t$ belonging to Class 1 and $(1,3)^t$, $(5,5)^t$ and $(3,6)^t$ belonging to Class 2, the means will be

$$m_{x1} = \begin{bmatrix} 3.66 \\ 2 \end{bmatrix}$$

$$m_{x2} = \begin{bmatrix} 3 \\ 4.66 \end{bmatrix}$$

The within class scatter matrix will be

$$\sigma_W = \begin{bmatrix} -1.66 \\ 0 \end{bmatrix} \times \begin{bmatrix} -1.66 & 0 \end{bmatrix} + \begin{bmatrix} 0.33 \\ 1 \end{bmatrix} \times \begin{bmatrix} 0.33 & -1 \end{bmatrix} + \begin{bmatrix} 1.33 \\ -1 \end{bmatrix}$$
$$\times \begin{bmatrix} 1.33 & -1 \end{bmatrix} + \begin{bmatrix} -2 \\ -1.66 \end{bmatrix} \times \begin{bmatrix} -2 & -1.66 \end{bmatrix} + \begin{bmatrix} 2 \\ 0.33 \end{bmatrix}$$
$$\times \begin{bmatrix} 2 & 0.33 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.33 \end{bmatrix} \times \begin{bmatrix} 0 & 1.33 \end{bmatrix} = \begin{bmatrix} 12.63 & 2.98 \\ 2.98 & 6.63 \end{bmatrix}$$

$$S_W^{-1} = \frac{1}{74.88}\begin{bmatrix} 6.63 & -2.98 \\ -2.98 & 12.63 \end{bmatrix}$$

The direction is given by

$$V = \sigma_W^{-1}(\text{mean}_1 - \text{mean}_2) = \frac{1}{74.88}\begin{bmatrix} 6.63 & -2.98 \\ -2.98 & 12.63 \end{bmatrix} \times \begin{bmatrix} 0.66 \\ -2.66 \end{bmatrix}$$

$$V = \frac{1}{74.88}\begin{bmatrix} 12.30 \\ -34.2 \end{bmatrix} = \begin{bmatrix} 0.164 \\ -0.457 \end{bmatrix}$$

Note that $v^t x \geq -0.586$ if $x \in \text{Class 1}$ and $v^t x \leq -1.207$ if $x \in \text{Class 2}$.

2.6.2 Principal Component Analysis (PCA)

PCA involves a mathematical procedure that transforms a number of correlated variables into a smaller number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. PCA finds the most accurate data representation in a lower dimensional space. The data is projected in the direction of maximum variance.

If x is a set of N column vectors of dimension D, the mean of the data set is

$$m_x = E\{x\}$$

The covariance matrix is

$$C_x = E\{(x - m_x)(x - m_x)^T\}$$

The components of C_x denoted by c_{ij} represent the covariances between the random variable components x_i and x_j. The component c_{ii} is the variance of the component x_i.

This is a symmetric matrix from which the orthogonal basis can be calculated by finding its eigenvalues and eigenvectors. The eigenvectors e_i and the corresponding eigenvalues λ_i are solutions of the equation

$$C_x e_i = \lambda_i e_i, i = 1, ..., n$$

By ordering the eigenvectors in the order of descending eigenvalues, an ordered orthogonal basis can be created with the first eigenvector having the direction of the largest variance of the data. In this way, we can find directions in which the data set has the most significant amounts of energy.

If A is the matrix consisting of eigenvectors of the covariance matrix as the row vectors formed by transforming a data vector x, we get

$$y = A(x - m_x)$$

The original data vector x can be reconstructed from y by

$$x = A^T y + m_x$$

Instead of using all the eigenvectors, we can represent the data in terms of only a few basis vectors of the orthogonal basis. If we denote the matrix having the K first eigenvectors as A_K, we get

$$y = A_K(x - m_x)$$

and

$$x = A_K^T y + m_x$$

The original data vector is projected on the coordinate axes having the dimension K and the vector is transformed back by a linear combination of the basis vectors. This minimises the mean-square error with the given number of eigenvectors used. By picking the eigenvectors having the largest eigenvalues, as little information as possible is lost. This provides a way of simplifying the representation by reducing the dimension of the representation.

EXAMPLE 13

A data set contains four patterns in two dimensions. The patterns belonging to Class 1 are (1, 1) and (1, 2). The patterns belonging to Class 2 are (4, 4) and (5, 4). With these four patterns, the covariance matrix is

$$C_x = \begin{bmatrix} 4.25 & 2.92 \\ 2.92 & 2.25 \end{bmatrix}$$

The eigenvalues of C_x are

$$\lambda = \begin{bmatrix} 6.336 \\ 0.1635 \end{bmatrix}$$

where the two eigenvalues are represented as a column vector.

Since the second eigenvalue is very small compared to the first eigenvalue, the second eigenvector can be left out. The eigenvector which is most significant is

$$e_1 = \begin{bmatrix} 0.814 \\ 0.581 \end{bmatrix}$$

To transform the patterns on to the eigenvector, the pattern (1, 1) gets transformed to

$$\begin{bmatrix} 0.814 & 0.581 \end{bmatrix} \times \begin{bmatrix} -1.75 \\ -1.75 \end{bmatrix} = -2.44$$

Similarly, the patterns (1, 2), (4, 4) and (5, 4) get transformed to –1.86, 1.74 and 2.56.

When we try to get the original data using the transformed data, some information is lost. For pattern (1, 1), using the transformed data, we get

$$\begin{bmatrix} 0.814 & 0.581 \end{bmatrix}^T \times (-2.44) + \begin{bmatrix} 2.75 \\ 2.75 \end{bmatrix} = \begin{bmatrix} -1.99 & -1.418 \end{bmatrix}^T + \begin{bmatrix} 2.75 \\ 2.75 \end{bmatrix}$$

$$= \begin{bmatrix} 0.76 \\ 1.332 \end{bmatrix}$$

2.7 Feature Selection

The features used for classification may not always be meaningful. Removal of some of the features may give a better classification accuracy. Features which are useless for classification are found and left out. Feature selection can also be used to speed up the process of classification, at the same time, ensuring that the classification accuracy is optimal. Feature selection ensures the following:

1. *Reduction in cost of pattern classification and design of the classifier* Dimensionality reduction, i.e., using a limited feature set simplifies both the representation of patterns and the complexity of the classifiers. Consequently, the resulting classifier will be faster and use less memory.
2. *Improvement of classification accuracy* The improvement in classification is due to the following reasons:
   (a) The performance of a classifier depends on the inter-relationship between sample sizes, number of features, and classifier complexity. To obtain good classification accuracy, the number of training samples must increase as the number of features increase. The probability of misclassification does not increase as the number of features increases, as long as the number of training patterns is arbitrarily large. Beyond a certain point, inclusion of additional features leads to high probabilities of error due to the finite number of training samples. If the number of training patterns used to train the classifier is small, adding features may actually degrade the performance of a classifier. This is called the peaking phenomena and is illustrated in Figure 2.11. Thus it can be seen that a small number of features can alleviate the curse of dimensionality when the number of training patterns is limited.
   (b) It is found that under a broad set of conditions, as dimensionality increases, the distance to the nearest data point approaches the distance to the furthest data point. This affects the results of the nearest neighbour problem. Reducing the dimensionality is meaningful in such cases to get better results.

All feature selection algorithms basically involve searching through different feature sub-sets. Since feature selection algorithms are basically search procedures, if the number of features is large (or even above, say, 30 features), the number of feature sub-sets become prohibitively large. For optimal algorithms such as the exhaustive search and branch and bound technique , the computational efficiency comes down rather steeply and it is necessary to use sub-optimal procedures which are essentially faster.

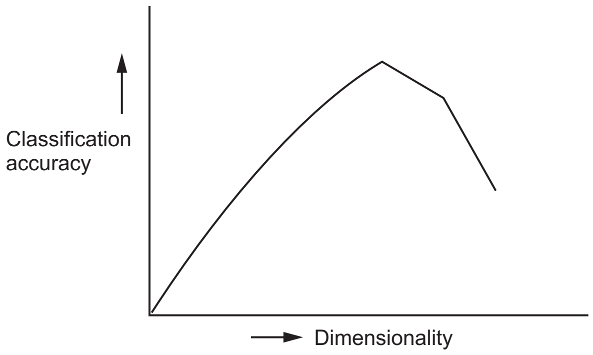


Figure 2.11 Peaking phenomenon or the curse of dimensionality

Feature sub-sets that are newly discovered have to be evaluated by using a criterion function. For a feature sub-set X, we have to find $J(X)$. The criterion function usually used is the classification error on the training set. Here $J = (1 - P_e)$, where $P(e)$ is the probability of classification error. This suggests that a higher value of J gives a better feature sub-set.

2.7.1 Exhaustive Search

The most straightforward approach to the problem of feature selection is to search through all the feature sub-sets and find the best sub-set. If the patterns consist of d features, and a sub-set of size m features is to be found with the smallest classification error, it entails searching all $\binom{d}{m}$ possible sub-sets of size m and selecting the sub-set with the highest criterion function $J(.)$, where $J = (1 - P_e)$. Table 2.3 shows the various sub-sets for a data set with 5 features. This gives sub-sets of three features. A 0 means that the corresponding feature is left out and a 1 means that the feature is included.

Table 2.3 Features selected in exhaustive search

Sl. No.	f1	f2	f3	f4	f5
1.	0	0	1	1	1
2.	0	1	0	1	1
3.	0	1	1	0	1
4.	0	1	1	1	0
5.	1	0	0	1	1
6.	1	0	1	0	1
7.	1	0	1	1	0
8.	1	1	0	0	1
9.	1	1	0	1	0
10.	1	1	1	0	0

This technique is impractical to use even for moderate values of d and m. Even when d is 24 and m is 12, approximately 2.7 million feature sub-sets must be evaluated.

2.7.2 Branch and Bound Search

The branch and bound scheme avoids exhaustive search by using intermediate results for obtaining bounds on the final criterion value. This search method assumes monotonicity as described below.

Let $(Z_1, Z_2, ..., Z_l)$ be the $l = d - m$ features to be discarded to obtain an m feature sub-set. Each variable Z_i can take on values in $\{1,\ 2, ...,\ d\}$. The order of the Z_is is immaterial and they are distinct, so we consider only sequences of Z_i such that $Z_1 < Z_2 < ... < Z_l$. The feature selection criterion is $J_l(Z_1, \ldots, Z_l)$. The feature sub-set selection problem is to find the optimum sub-set $Z_1^*, \ldots, Z_l^*$ such that

$$J_l(Z_1^*, ..., Z_l^*) = \max J_l(Z_1, ..., Z_l)$$

If the criterion J satisfies monotonicity, then

$$J_1(Z_1) \geq J_2(Z_1, Z_2) \geq ... \geq J_l(Z_1, ..., Z_l)$$

This means that a sub-set of features should not be better than any larger set that contains the sub-set.

Let B be a lower bound on the optimum value of the criterion $J_l(Z_1^*, ..., Z_l^*)$. That is

$$B \leq J_l(Z_1^*, ..., Z_l^*)$$

If $J_k(Z_1, ..., Z_k)(k \leq l)$ were less than B, then

$$J_l(Z_1, ..., Z_k, Z_{k+1}, ..., Z_l) \leq B$$

for all possible $Z_{k+1}, ..., Z_l$.

This means that whenever the criterion evaluated for any node is less than the bound B, all nodes that are successors of that node also have criterion values less than B, and therefore cannot lead to the optimum solution. This principle is used in the branch and bound algorithm. The branch and bound algorithm successively generates portions of the solution tree and computes the criterion. Whenever a sub-optimal partial sequence or node is found to have a criterion lower than the bound at that point in time, the sub-tree under that node is implicitly rejected and other partial sequences are explored.

Starting from the root of the tree, the successors of the current node are enumerated in the ordered list LIST(i). The successor for which the partial criterion $J_i(Z_1, ..., Z_i)$ is maximum is picked as the new current node and the algorithm moves on to the next higher level. The lists LIST(i) at each level i keeps track of the nodes that have been explored. The SUCCESSOR variables determine the number of successor nodes the current node will have at the next level. AVAIL keeps track of the feature values that can be enumerated at any level. Whenever the partial criterion is found to be less than the bound, the algorithm backtracks to the previous level and selects a hitherto unexplored node for expansion. When the algorithm reaches the last level l, the lower bound B is updated to be the current value of $J_l(Z_1, ..., Z_l)$ and the current sequence $(Z_1, ..., Z_l)$ is saved as $(Z_1^*, ..., Z_l^*)$. When all the nodes in LIST(i) for a given i are exhausted, the algorithm backtracks to the previous level. When the algorithm backtracks to level 0, it terminates. At the conclusion of the algorithm, $(Z_1^*, ..., Z_l^*)$ will give the complement of the best feature set.

The branch and bound algorithm is as follows:

Step 1: Take root node as the current node.

Step 2: Find successors of the current node.

Step 3: If successors exist, select the successor i not already searched which has the maximum J_i and make it the current node.

Step 4: If it is the last level, update the bound B to the current value of $J_l(Z_1, ..., Z_l)$ and store the current path $(Z_1, ..., Z_l)$ as the best path $(Z_1^*, ..., Z_l^*)$. Backtrack to previous levels to find the current node.

Step 5: If the current node is not the root, go to Step 2.

Step 6: The complement of the best path at this point $(Z_1^*, ..., Z_l^*)$ gives the best feature set.

This algorithm assumes monotonicity of the criterion function $J(.)$. This means that for two feature sub-sets χ_1 and χ_2, where $\chi_1 \subset \chi_2$, $J(\chi_1) < J(\chi_2)$. This is not always true.

The modified branch and bound algorithm (BAB$^+$) gives an algorithmic improvement on the BAB. Whenever the criterion evaluated for any node is not larger than the bound B, all nodes that are its successors also have criterion values not larger than B and therefore cannot be the optimal solution. BAB$^+$ does not generate these nodes and replaces the current bound with the criterion value which is larger than it and is held by the terminal node in the search procedure. The bound reflecting the criterion value held by the globally optimal solution node will never be replaced. This algorithm implicitly skips over the intermediate nodes and “short-traverses”, thus improving the efficiency of the algorithm.

The relaxed branch and bound (RBAB) can be used even if there is a violation of monotonicity. Here we have a margin b and even if J exceeds the bound by an amount less than b, the search is continued. b is called the margin. On the other hand, the modified relaxed branch and bound algorithm gets rid of the margin and cuts branches below a node Z only if both Z and a parent of Z have a criterion value less than the bound.

Example 14

Consider a data set having four features f1, f2, f3 and f4. Figure 2.12 shows the tree that results when one feature at a time is removed. The numbering of the nodes shows the order in which the tree is formed. When a leaf node is reached, the criterion value of the node is evaluated. When node 3 is reached, since its criterion value is 25, the bound is set as 25. Since the bound of the next node 4 is 45 which is greater than 25, the bound is revised to 45. Node 5 has a smaller criterion than the bound and therefore the bound remains as 45. When node 6 is evaluated, it is found to have a criterion of 37. Since this is smaller than the bound, this node is not expanded further. Since monotonicity is assumed, nodes 7 and 8 are estimated to have criteria which are smaller than 37. Similarly node 9 which has a criterion of 43 is also not expanded further since its criterion is less than the bound. The features removed would therefore be f1 and f3. This is shown in Figure 2.13. Using relaxed branch and bound, if the margin b is 5, considering node 9, since the difference between its criterion 43 and the bound which is 45 is less than the margin, node 9 is further expanded to give node 10. If monotonicity is violated and node 10 has a criterion greater than 45, it will be chosen as the one with the best criterion which makes the two features to be removed as f3 and f4. This is shown in Figure 2.13 where the criterion value is 47.

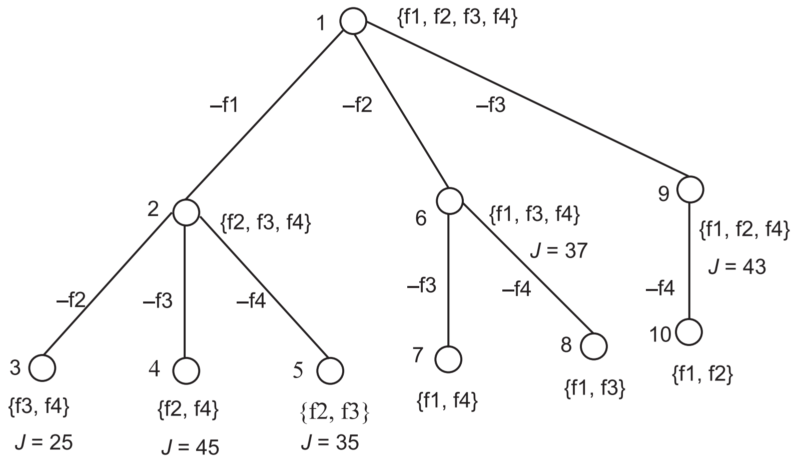


Figure 2.12 The solution tree for d = 4 and m = 2

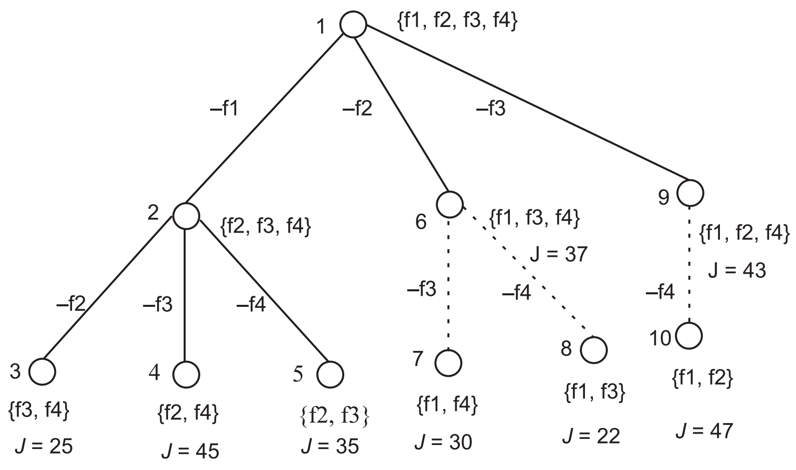


Figure 2.13 Using branch and bound algorithm

2.7.3 Selection of Best Individual Features

This is a very simple method where only the best features are selected. All the individual features are evaluated independently. The best m features are then selected. This method, though computationally simple, is very likely to fail since the dependencies between features also have to be taken into account.

2.7.4 Sequential Selection

These methods operate either by evaluating growing feature sets or shrinking feature sets. They either start with the empty set and add features one after the other; or they start with the full set and delete features. Depending on whether we start with an empty set or a full one, we have the sequential forward selection (SFS) and the sequential backward selection (SBS) respectively. Since these methods do not examine all possible sub-sets, the algorithms do not guarantee the optimal result (unlike the branch and bound and the exhaustive search techniques). Besides, these methods suffer from the ‘‘nesting effect’’ explained below.

Sequential Forward Selection (SFS)

The sequential method adds one feature at a time, each time checking on the performance. It is also called the ‘‘bottom-up approach’’ since the search starts with an empty set and successively builds up the feature sub-set. The method suffers from the ‘‘nesting effect’’ since features once selected cannot be later discarded. The number of feature sub-sets searched will be

$$\sum_{i=1}^{m}(d-i+1) = m[d - \frac{(m-1)}{2}]$$

The features are selected according to their significance. The most significant feature is selected to be added to the feature sub-set at every stage. The most significant feature is the feature that gives the highest increase in criterion function value as compared to that of the others before adding the feature. If U_0 is the feature added to the set X_k consisting of k features then the significance of this is

$$S_{k+0}(U_0) = J(X_k \bigcup U_0) - J(X_k)$$

The most significant feature with respect to the set X_k is

$$S_{k+0}(U_0^r) = \max_{1 \le i \le \phi} S_{k+0}(U_0^i)$$

i.e., $$J(X_k \bigcup U_0^r) = \max_{1 \le i \le \phi} J(X_k \bigcup U_0^i)$$

where ϕ is the number of all the possible 0-tuples.

The least significant feature with respect to set X_k is

$$S_{k+0}(U_0^r) = \min_{1 \le i \le \phi} S_{k+0}(U_0^i)$$

i.e., $$J(X_k \bigcup U_0^r) = \min_{1 \le i \le \phi} J(X_k \bigcup U_0^i)$$

Sequential Backward Selection (SBS)

This method first uses all the features and then deletes one feature at a time. It is also called the "top-down" approach, since the search starts with the complete set of features and successively discards features. The disadvantage is that features, once discarded, cannot be re-selected. Features are discarded successively by finding the least significant feature at that stage.

2.7.5 Sequential Floating Search

To take care of the "nesting effect" of SFS and SBS, "plus l take away r" selection was introduced, where the feature sub-set is first enhanced by l features using forward selection and then r features are deleted using backward selection. The main drawback of this method is that there is no theoretical way of predicting the values of l and r to achieve the best feature sub-set. The floating search method is an improvement on this, since there is no need to specify the parameters l and r. The number of forward and backward steps is determined dynamically while the method is running so as to maximise the criterion function. At each step, only a single feature is added or removed. The values of l and r is kept ‘‘floating’’, i.e., they are kept flexible so as to approximate the optimal solution as much as possible. Consequently, the dimensionality of the feature sub-set does not change monotonically but is actually ‘‘floating’’ up and down.

Sequential Floating Forward Search (SFFS)

The principle of SFFS is as follows:

Step 1: Let $k = 0$.

Step 2: Add the most significant feature to the current sub-set of size k. Let $k = k+1$.

Step 3: Conditionally remove the least significant feature from the current sub-set.

Step 4: If the current sub-set is the best sub-set of size $k - 1$ found so far, let $k = k - 1$ and go to Step 3. Else return the conditionally removed feature and go to Step 2.

Example 15

Consider a data set of patterns with four features f1, f2, f3 and f4. The steps of the algorithm are as given below.

Step 1: Let the sub-set of the features chosen be empty, i.e., $F = \phi$.
Step 2: The most significant feature is found. Let it be f3. Then F = {f3}.
Step 3: The least significant feature is found, i.e., it is seen if f3 can be removed.
Step 4: The removal of f3 does not improve the criterion value and f3 is restored.
Step 5: The most significant feature is found. Let it be f2. Then F = {f3, f2}
Step 6: The least significant feature is found. It is f2. Condition fails.
Step 7: The most significant feature is found. Let it be f1. Then F = {f3, f2, f1}
Step 8: The least significant feature is found. Let it be f3. If it gives a better sub-set, then F = {f1, f2}.

The optimal set would be F = {f1, f2} if $m = 2$. It is noted that if in the last step, the least significant feature was found to be f1, F = {f2, f3} as in Step 5. This leads to looping.

Sequential Floating Backward Search (SBFS)

This method is similar to the SFFS except that backward search is carried out first and then the forward search. The principle of SBFS is as follows:

Step 1: $k = 0$.

Step 2: Remove the least significant feature from the current sub-set of size k. Let $k = k - 1$.

Step 3: Conditionally add the most significant feature from the features not in the current sub-set.

Step 4: If the current sub-set is the best sub-set of size $k - 1$ found so far, let $k = k + 1$ and go to Step 3. Else remove the conditionally added feature and go to Step 2.

The algorithm starts with all the features and then removes them one by one. Due to this, the method does not always give good results. If the dimensionality is large, i.e., d is large and if m is very small, it will be time-consuming and it is better to use the SFFS.

2.7.6 Max–Min Approach to Feature Selection

This method has a computational advantage over the other well-known methods due to the fact that instead of computationally time-consuming calculations in a multi-dimensional space, the max–min method requires calculations in two-dimensional space only. However, the results achieved by this method are rather unsatisfactory. It works as follows.

Let us denote

f_i = feature from the selected feature set $F_k = \{f_1, ..., f_k\}$ acquired in the *i*th step of the selection procedure.
g_j = *j*th feature from the set of features not selected.
$\delta J(y_j, x_i)$ = the absolute value of the difference between $J(g_j, f_i)$ and $J(f_i)$.

In the max–min method, the new feature y_j is chosen as the next feature if it yields

$$\max_{\forall y_j} \min_{\forall x_i} \Delta J(y_j, x_i)$$

The poor results of this method confirm that it is not possible to select a set of features in a high-dimensional space based on two-dimensional information without a substantial information loss.

Example 16

Consider a data set with patterns having four features f1, f2, f3 and f4. The criterion function value using up to two features at a time is shown in Table 2.4.

Table 2.4 Criterion function using a sub-set of features

Feature	f1	f2	f3	f4
f1	10	30	35	55
f2	30	20	40	53
f3	35	40	30	42
f4	55	53	42	40

If $J(\text{f1})$ represents the criterion function using only feature f1, then

$$J\ (\text{f1}) = 10;\ J\ (\text{f2}) = 20;\ J\ (\text{f3}) = 30;\ J\ (\text{f4}) = 40$$

This is depicted by the diagonal entries in Table 2.4.

First considering f1, if J (f1, f2) $= 30$, J(f1, f3) $= 35$ and J (f1, f4) $= 55$, then

$$\Delta\ J\ (\text{f1, f2}) = 10;\ \Delta\ J\ (\text{f1, f3}) = 5;\ \Delta\ J\ (\text{f1, f4}) = 15$$

The minimum of these is $\Delta\ J$ (f1, f3) $= 5$

Next considering f2, if J (f2, f1)$=30$, J (f2, f3)$= 40$ and J (f2,f4) $= 53$, then

$$\Delta\ J\ (\text{f2, f1}) = 20;\ \Delta\ J\ (\text{f2, f3}) = 10;\ \Delta\ J\ (\text{f2, f4}) = 13$$

The minimum of these is $\Delta\ J$ (f2, f3) $= 10$

Next considering f3, if J (f3, f1) $= 35$, J (f3, f2) $= 40$ and J (f3, f4) $= 42$, then

$$\Delta\ J\ (\text{f3, f1}) = 25;\ \Delta\ J\ (\text{f3, f2}) = 20;\ \Delta\ J\ (\text{f3, f4}) = 2$$

The minimum of these is $\Delta\ J$ (f3, f4) $= 2$

Next considering f4, if J (f4, f1) $= 55$, J (f4, f2) $= 53$ and J (f4, f3) $= 42$, then

$$\Delta\ J\ (\text{f4, f1}) = 45;\ \Delta\ J\ (\text{f4, f2}) = 33;\ \Delta\ J\ (\text{f4, f3}) = 12$$

The minimum of these is $\Delta\ J$ (f4, f3) $= 12$

Finding the maximum of the four minimum values, we get f4 which is chosen as the next feature.

It can be seen that only two features are considered at a time. The features chosen could have been different if more number of features are considered at a time.

2.7.7 Stochastic Search Techniques

Genetic algorithm is a stochastic search technique which is often used for feature selection. The population in the GA consists of strings which are binary in nature. Each string (or chromosome) is of length d, with each position i being zero or one depending on the absence or presence of feature i in the set. This means that each feature sub-set is coded as a d-element bit string or binary valued vector. Each string in the population is a feature selection vector α, where each $\alpha = \alpha_1, ..., \alpha_d$ and α_i assumes a value 0 if the ith feature is excluded and 1 if it is present in the sub-set. To compute its fitness, each chromosome is evaluated by determining its performance on the training set.

2.7.8 Artificial Neural Networks

A multilayer feed-forward network with a back-propagation learning algorithm is used in this method. The approach considered here is to take a larger than necessary network and then remove unnecessary nodes. Pruning is carried out by eliminating the least salient nodes. It is based on the idea of iteratively eliminating units and adjusting the remaining weights in such a way that the network performance does not become worse over the entire training set. The pruning problem is formulated in terms of solving a system of linear equations using the optimisation technique.

The pruning of nodes corresponds to removing the corresponding features from the feature set. The saliency of a node is defined as the sum of the increase in error over all the training patterns caused by the removal of that node. The node pruning based feature selection first trains a network and then removes the least salient node. The reduced network is trained again, followed by the removal of the least salient node. This procedure is repeated to get the least classification error.

2.8 Evaluation of Classifiers

Before using a classifier to carry out a classification task, it is necessary to evaluate its performance. The various parameters of the classifier which requires to be taken into account are

1. *Accuracy of the classifier* The main aim of using a classifier is to correctly classify unknown patterns. Classification accuracy is an important parameter in evaluating a classifier.

2. *Design time and classification time* Design time is the time taken to build the classifier from the training data while classification time is the time taken to classify a pattern using the designed classifier. The nearest neighbour classifier does not require any design time. However, the classification time will be high since each test pattern has to be compared to all the patterns in the training set. On the other hand, the neural network classifier requires a high design time to train the weights in the network. But, the classification time will be low as it is only necessary to run the pattern through the trained network to get the class of the pattern.
3. *Space required* If an abstraction of the training set is carried out, the space required will be less. If no abstraction is carried out and the entire training data is required for classification, the space requirement is high. The nearest neighbour classifier requires the entire training data to be stored and therefore, the space required will be more. The neural network, decision tree and the rule-based classifier requires only the abstraction of the training set and therefore requires less space. For instance, the neural network classifier requires only the trained neural network to carry out the classification task. The training data set is not required.
4. *Explanation ability* If the reason for the classifier in choosing the class of a pattern is clear to the user, then its explanation ability is good. For instance, in the decision tree classifier, following the path from the root of the tree to the leaf node for the values of the features in the pattern will give the class of the pattern. Similarly, the user understands why a rule based system chooses a particular class for a pattern. On the other hand, the neural network classifier has a trained neural network and it is not clear to the user what the network is doing.
5. *Noise tolerance* This refers to the ability of the classifier to take care of outliers and patterns wrongly classified.

The accuracy of the classifier is the parameter usually taken into account. However, if the classification time is too large and it is required to carry out the classification task quickly, it may be better to sometimes sacrifice the accuracy and use a classifier which is faster. In places where there is a space limitation like in hand-held devices, it is good to use a classifier requiring little space. In crucial applications like medical diagnostics, explanation ability may be necessary in a classifier so that the doctors are sure that the classifier is doing the right thing.

To estimate how good a classifier is, an estimate can be made using the training set itself. This is known as re-substitution estimate. It assumes that the training data is a good representative of the data. Sometimes, a part of the training data is used as a measure of the performance of the classifier. Usually the training set is divided into smaller sub-sets. One of the sub-sets is used for training while the other is used for validation. The different methods of validation are as follows:

1. *Holdout method* The training set is divided into two sub-sets. Typically two-thirds of the data is used for training and one-thirds is used for validation. It could also be one-half for training and one-half for testing or any other proportion.

2. *Random sub-sampling* In this method, the holdout method is repeated a number of times with different training and validation data each time. If the process is repeated k times, the overall accuracy will be

$$\text{acc}_{\text{overall}} = \frac{1}{k}\Sigma_{i=1}^{k}\text{acc}_i$$

3. *Cross-validation* In cross-validation, each pattern is used the same number of times for training and exactly once for testing. An example of cross-validation is the two-fold cross-validation. Here the data set is divided into two equal sub-sets. First, one set is used for training and the other for testing. Then the roles of the sub-sets are swapped—the set for training becomes the set for validation and vice versa.

   In k-fold cross-validation, the data is divided into k equal sub-sets. During each run, one sub-set is used for testing and the rest of the sub-sets are used for training. This procedure is carried out k times so that each sub-set is used for testing once. A special case of the k-fold cross-validation is when $k = n$, where n is the number of patterns in the data set. This is called the *leave-one-out* approach, where each test set contains only one pattern. The procedure is repeated n times.

4. *Bootstrap procedure* Here a pattern is chosen randomly from the data set, while not deleting it from the data set. Another pattern is then randomly chosen. This is repeated till we have N patterns. These are used for training and the patterns not chosen are used for testing.

2.9 Evaluation of Clustering

It is also necessary to evaluate the clustering carried out by any method. Usually, if the dimensions are low in number, a manual inspection of the clustering graph gives a good idea if the clustering is okay. The quality of the clustering can be measured by examining the similarity between patterns. Patterns belonging to the same cluster should have high similarity and patterns belonging to different clusters should have low similarity.

Discussion

The representation of patterns is very important as a good representation makes use of the discriminating attributes of the objects. Use of good representation reduces the computation burden in pattern classification. There are a variety of data structures which can be used for representing patterns. The method chosen would depend on the problem. Feature selection is the process of selecting a sub-set of the features which is found to be more relevant and weeding out the features which really do not contribute much to the classification process. All feature selection methods depend on trying out different sub-sets of features and finding the best sub-set. Before deciding on the classifier to be used for a particular task, it is necessary to evaluate the classifier. The criteria to be used for evaluation of classifiers and how to go about evaluating classifiers has been discussed.

Further Reading

Huttenlocher (1993) describes the Hausdorff distance which is used as a similarity measure for patterns with images. The frequent pattern (FP) tree used to represent patterns in a compact way is given by Han et al. (2004).

A number of papers explain the different methods of feature selection. Narendra and Fukunaga (1977) explain the branch and bound algorithm for feature selection. An improvement on this algorithm is given Yu and Yuan (1993). Floating search methods for feature selection are described by Pudil et al. (1994) and Somol et al. (1999). Kuncheva and Jain (1999) and Siedlecki and Sklansky (1989) show how genetic algorithms can be used for feature selection. Kuncheva and Jain (1999) show how we can combine the process of feature selection and prototype selection.

Exercises

1. Find the centroid and medoid (most centrally located pattern) for the following set of patterns:

   (1, 1), (1, 3), (1, 4), (2, 2), (2, 3), (3, 1), (3, 4), (4, 2).

   Under what conditions will the medoid and the centroid be identical?
2. Under what conditions will a set of points give positive minimum variance value? Why is the centroid a good representative of a set of points?
3. Find the edit distance between the two strings "HOUSE" and "MOUND".
4. Show that the squared Euclidean distance is not a metric.
5. For non-binary data, show that $L_1 > L_2 > L_\infty$.
6. The similarity function $D(X, Y)$ between X and Y as given below is a non-metric.

$S(X,Y) = \frac{X^tY}{||X||\,||Y||}$

$D(X,Y) = 1 - S(X,Y)$

Show how this can be converted into a metric.

7. The Hausdorff distance between two points I and J is given by

   $\max(\max_{i \in I} \min_{j \in J} \parallel i - j \parallel, \max_{j \in J} \min_{i \in I} \parallel i - j \parallel)$

   This distance is not a metric. Which property is violated here, making it a non-metric?

8. Find the mutual neighbourhood distance (MND) between D and E given the following set of points:

   A = (1, 1); B = (2, 2); C = (2, 1); D = (3, 1);

   E = (4, 4); F = (5, 4); G = (6, 4); H = (6, 5)

9. Give the conditions under which the FP tree built on N d-dimensional binary patterns has the minimum number of nodes where each pattern has at most l $(< d)$ 1s.

10. A data set consists of the following patterns:

    (1, 1, 1), (2, 2, 1), (1.5, 0.5, 1), (1, 3, 1), (4, 4, 2), (5, 5, 2), (4, 5, 2), (4, 6, 2)

    where each pattern consists of the x-coordinate, y-coordinate and the class. Find the direction of the W vector associated with Fisher's linear discriminant.

11. Given n d-dimensional patterns from two classes and $n < d$, comment on whether S_B and S_W used in Fisher's linear discriminant are singular or not.

12. Find the direction of W using Fisher's linear discriminant when $m_1 = m_2$ and $s_1 = s_2 = 0.5I$, where I is the identity matrix.

13. Given below is a set of patterns with three features X, Y and Z.

X	Y	Z	Output
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Are any of the features redundant? Explain.

14. If there are ten features and it is necessary to reduce the number of features to six so that the best set of six features is chosen, what is the number of feature sub-sets to be evaluated to find the optimal set of six features in the exhaustive search?

Computer Exercises

1. Write a program to generate the FP tree of a set of patterns. Use the program to generate the FP tree for the data given in Table 2.1.
2. Extend the program of Computer Exercise 1 so as to get the nearest neighbour of any transaction using the FP tree and use it on the data given in Table 2.1.
3. Write a program to obtain the Fisher's linear discriminant for a training data set. Obtain the Fisher's linear discriminant for the two-dimensional data set :

   (1, 1, 1), (1, 2, 1), (1, 3, 1), (2, 1, 1), (2, 2, 1), (2, 3, 1), (2, 3.5, 1),
   (2.5, 2,1), (3.5, 1, 1), (3.5, 2, 1), (3.5, 3, 2), (3.5, 4,2), (4.5, 1, 2),
   (4.5, 2, 2), (4.5, 3, 2), (5, 4, 2), (5, 5, 2), (6, 3, 2), (6, 4, 2), (6, 5, 2)

   where each pattern is represented by feature 1, feature 2 and the class.
4. Write a program to implement the principal component analysis and use it on the data given in Computer Exercise 3.
5. Implement the branch and bound search for feature selection. Apply it to a data set having a number of features and find the set of features found by the program.
6. Implement the sequential floating forward search and apply it to a data set used for Computer Exercise 5. Compare the features found with that found using the branch and bound algorithm.
7. Implement the sequential floating backward search and apply it to the data set used for Computer Exercise 5. Compare the feature set found with that found using SFFS and branch and bound algorithm. Which algorithm gives the best set of features?

Bibliography

1. Han, Jaiwei, Jian Pei, Yiwen Yin and Runying Mao. Mining frequent patterns without candidate generation: A frequent pattern tree approach. *Data Mining and Knowledge Discovery* 8(1): 53–87. 2004.

2. Huttenlocher, D. P., G. A. Klanderman and W. A. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15(9): 850–863. 1993.

3. Jain, A. K. and D. Zongker. Feature selection: Evaluation, application and small sample performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 19:153–157. 1997.

4. Kuncheva, L. and L. C. Jain. Nearest neighbor classifier: Simultaneous editing and feature selection. *Pattern Recognition Letters* 20:1149–1156. 1999.

5. Narendra, P. M. and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Trans. Computers* 26(9): 917–922. 1977.

6. Pudil, P., J. Novovicova and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters* 15: 1119–1125. 1994.

7. Siedlecki, W. and J. Sklansky. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters* 10: 335–347. 1989.

8. Somol, P., P. Pudil, J. Novovicova, P. Paclik. Adaptive floating search methods in feature selection. *Pattern Recognition Letters* 20: 1157–1163. 1999.

9. Yu, Bin and Baozong Yuan. A more efficient branch and bound algorithm for feature selection. *Pattern Recognition* 26(6): 883–889. 1993.