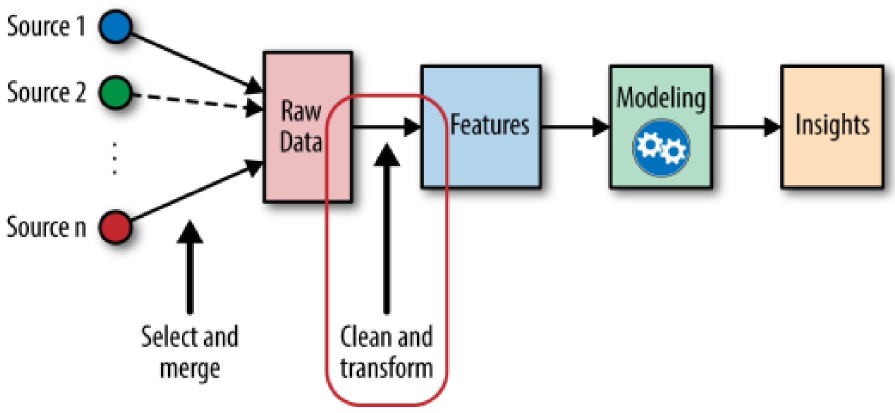
Lecture 1

Feature extraction

A feature is a numeric representation of raw data. There are many ways to turn raw data into numeric measurements, which is why features can end up looking like a lot of things. Naturally, features must derive from the type of data that is available. Perhaps less obvious is the fact that they are also tied to the model; some models are more appropriate for some types of features, and vice versa. The right features are relevant to the task at hand and should be easy for the model to ingest. Feature engineering is the process of formulating the most appropriate features given the data, the model, and the task.Feature extraction addresses the problem of finding the most compact and informative set of features, to improve the efficiency or data storage and processing.



The number of features is also important. If there are not enough informative features, then the model will be unable to perform the ultimate task. If there are too many features, or if most of them are irrelevant, then the model will be more expensive and tricky to train. Something might go awry in the training process that impacts the model?s performance.

Feature extraction is highly domain-specific and depend on the data type e.g numeric data, text data, image data. Good features should not only represent salient aspects of the data, but also conform to the assumptions of the model. Human expertise, which is often required to convert ?raw? data into a set of useful features, can be complemented by automatic feature construction methods. In some approaches, feature construction is integrated in the modeling process. For examples the ?hidden units? of artificial neural networks compute internal representations analogous to constructed features. In other approaches, feature construction is a preprocessing.

Hence, transformations are often necessary. Numeric feature engineering techniques are fundamental. They can be applied whenever raw data is converted into numeric features.

Let x be a pattern vector of dimension n, $\mathbf{x} = [x_1, x_2, ...x_n]$. The components x_i of this vector are the original features. We call $\mathbf{x}'$ a vector of transformed features of dimension n'. Preprocessing transformations may include:

Standardization: Features can have different scales although they refer to comparable objects. Consider for instance, a pattern $x = [x_1, x_2]$ where x_1 is a width measured in meters and x_2 is a height measured in centimeters. Both can be compared, added or subtracted but it would be unreasonable to do it before appropriate normalization. The following classical centering and scaling of the data is often used: $x'_i = \frac{(x_i ? \mu_i)}{\sigma_i}$, where μ_i and σ_i are the mean and the standard deviation of feature x_i over training examples.

Normalization: Consider for example the case where $\mathbf{x}$ is an image and the x_i's to normalize x by dividing it by the total number of counts in order to encode the distribution and remove the dependence on the size of the image. This translates into the formula: $\mathbf{x}' = \mathbf{x}/||\mathbf{x}||$

Signal enhancement The signal-to-noise ratio may be improved by applying signal or image-processing filters. These operations include baseline or background removal, de-noising, smoothing, or sharpening. The Fourier transform and wavelet transforms are popular methods.

Extraction of local features: For sequential, spatial or other structured data, specific techniques like convolutional methods using hand-crafted kernels or syntactic and structural methods are used. These techniques encode problem specific knowledge into the features.

Linear and non-linear space embeddin methods: When the dimensionality of the data is very high, some techniques might be used to project or embed the data into a lower dimensional space while retaining as much information as possible. Classical examples are Principal Component Analysis (PCA) and Multidimensional Scaling (MDS) The coordinates of the data points in the lower dimension space might be used as features or simply as a means of data visualization.

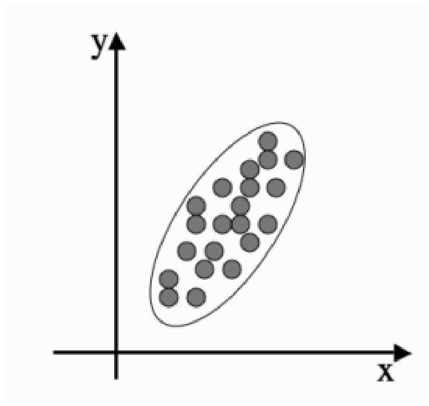
Non-linear expansions: Althoug dimensionality reduction is often summoned when speaking about complex data, it is sometimes better to increase the dimensionality. This happens when the problem is very complex and first order interactions are not enough to derive good results. This consists for instance in computing products of the original features x_i to create monomials $x_{k_1}, x_{k_2}...x_{k_p}$.

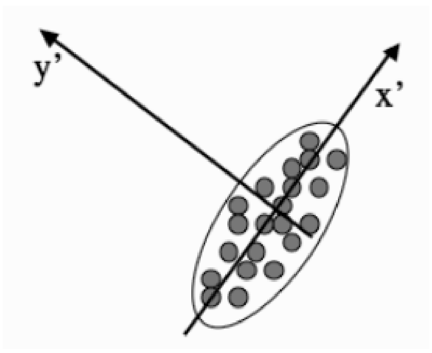
Feature discretization. Some algorithms do no handle well continuous data. It makes sense then to discretize continuous values into a finite discrete set. This step not only facilitates the use of certain algorithms, it may simplify the data description and improve data understanding

Lecture 2

Principal component analysis

Dimensionality reduction is about getting rid of ”uninformative information” while retaining the crucial bits. There are many ways to define ”uninformative.” PCA focuses on the notion of linear dependency. We describe the column space of a data matrix as the span of all feature vectors. If the column space is small compared to the total number of features, then most of the features are linear combinations of a few key features. Linearly dependent features are a waste of space and computation power because the information could have been encoded in much fewer features. To avoid this situation, principal component analysis tries to reduce such ”fluff” by squashing the data into a much lower-dimensional linear subspace. The key idea here is to replace redundant features with a few new features that adequately summarize information contained in the original feature space. It?s easy to tell what the new feature should be when there are only two features. It’s much harder when the original feature space has hundreds or thousands of dimensions.





The main purpose of principal component analysis (PCA) is to reduce the dimensionality of a high-dimensional data set consisting of a large number of interrelated variables and at the same time to retain as much as possible of the variation present in the data set. The principal components (PCs) are new variables that are uncorrelated and ordered such that the first few retain most of the variation present in all of the original variables.

The PCs are defined as follows. Let $\mathbf{v} = (v_1, v_2, \ldots, v_d)'$ be a vector of d random variables, where $'$ is the transpose operation. The first step is to find a linear function 1v of the elements of v that maximizes the variance, where a_1 is a d-dimensional vector $(a_{11}, a_{12}, ..., a_{1d})'$, so

$$\mathbf{a}_1'\mathbf{v} = \sum_{i=1}^{d} a_{1i} v_i$$

After finding $\mathbf{a}_1'\mathbf{v}, \mathbf{a}_2'\mathbf{v}, \ldots, \mathbf{a}_{j-1}'\mathbf{v}$ we look for a linear function $\mathbf{a}_j'\mathbf{v}$ that is uncorrelated with $\mathbf{a}_1'\mathbf{v}, \mathbf{a}_2'\mathbf{v}, \ldots, \mathbf{a}_{j-1}'\mathbf{v}$ and has maximum variance. Then we will find d such linear functions after d steps. The jth derived variable $\mathbf{a}_{j-1}'\mathbf{v}$ is the jth PC. In general, most of the variation in v will be accounted for by the first few PCs.

To find the form of the PCs, we need to know the covariance matrix Σ of $\mathbf{v}$. In most realistic cases, the covariance matrix Σ is unknown, and it will be replaced by a sample covariance matrix. For $j = 1, 2, ..., d$, it can be shown that the jth PC is given by $z_j = \mathbf{a}_j\mathbf{v}$, where a_j is an eigenvector of corresponding to the jth largest eigenvalue λ_j.

$$\text{maximize } \operatorname{var}(\mathbf{a}_1'\mathbf{v}) \text{ subject to } \mathbf{a}_1'\mathbf{a} = 1$$

where var($\mathbf{a}_1\prime\mathbf{v}$) can be found by solving the following optimization problem:

$$\operatorname{var}(\mathbf{a}_1'\mathbf{v}) = \mathbf{a}_1'\Sigma\mathbf{a}_1$$

To solve the above optimization problem, the technique of Lagrange multipliers can be used. Let λ be a Lagrange multiplier. We want to maximize

$$\mathbf{a}_1'\Sigma\mathbf{a}_1 - \lambda\left(\mathbf{a}_1'\mathbf{a} - 1\right)$$

Differentiating equation with respect to a_1, we have

$$\Sigma\mathbf{a}_1 - \lambda\mathbf{a}_1 = 0$$

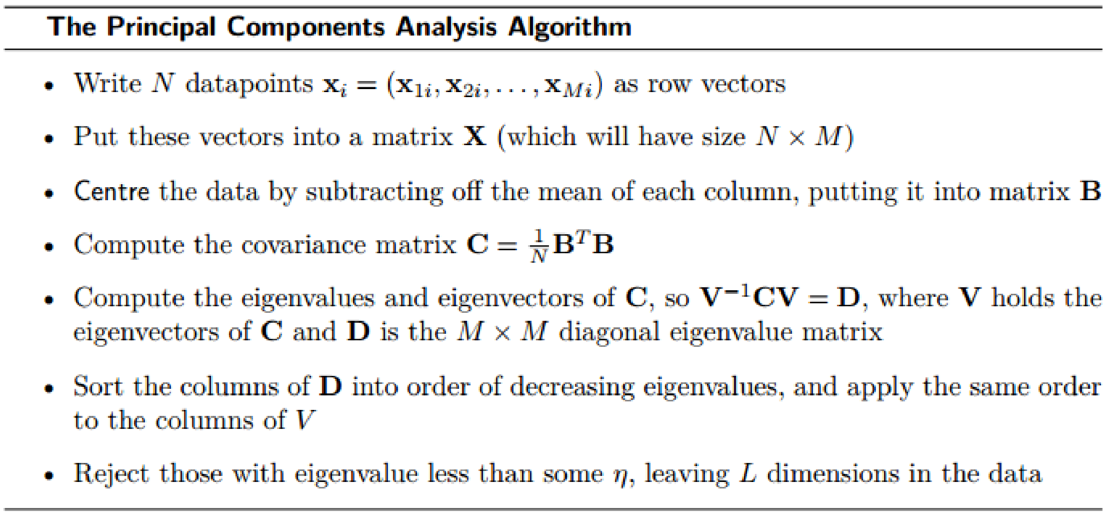
or

where I_d is $d \times d$ identity matrix.

Thus λ is an eigenvalue of Σ and $\mathbf{a}_1$ is the corresponding eigenvector. Since

$$\mathbf{a}_1'\Sigma\mathbf{a}_1 = \mathbf{a}_1'\lambda\mathbf{a}_1 = \lambda$$

$\mathbf{a}_1$ is the eigenvector corresponding to the largest eigenvalue of Σ. In fact, it can be shown that the jth PC is $\mathbf{a}_j'\sqsubseteq$ where $\mathbf{a}_j$ is an eigenvector of Σ corresponding to its jth largest eigenvalue λ_j



The Principal Components Analysis Algorithm

- Write N datapoints $\mathbf{x}_i = (\mathbf{x}_{1i}, \mathbf{x}_{2i}, \ldots, \mathbf{x}_{Mi})$ as row vectors
- Put these vectors into a matrix $\mathbf{X}$ (which will have size $N \times M$)
- Centre the data by subtracting off the mean of each column, putting it into matrix $\mathbf{B}$
- Compute the covariance matrix $\mathbf{C} = \frac{1}{N}\mathbf{B}^T\mathbf{B}$
- Compute the eigenvalues and eigenvectors of $\mathbf{C}$, so $\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{D}$, where $\mathbf{V}$ holds the eigenvectors of $\mathbf{C}$ and $\mathbf{D}$ is the $M \times M$ diagonal eigenvalue matrix
- Sort the columns of $\mathbf{D}$ into order of decreasing eigenvalues, and apply the same order to the columns of V
- Reject those with eigenvalue less than some η, leaving L dimensions in the data

Limitations of PCA

When using PCA for dimensionality reduction, one must address the question of how many principal components (k) to use. Like all hyperparameters, this number can be tuned based on the quality of the resulting model. But there are also heuristics that do not involve expensive computational methods. One possibility is to pick k to account for a desired proportion of total variance. (This option is available in the scikit-learn package PCA.) The variance of the projection onto the kth component is:

which is the square of the kth-largest singular value of X. The ordered list of singular values of a matrix is called its spectrum. Thus, to determine how many components to use, one can perform a simple spectral analysis of the data matrix and pick the threshold that retains enough variance.

Another method for picking k involves the intrinsic dimensionality of a dataset. This is a hazier concept, but can also be determined from the spectrum. Basically, if the spectrum contains a few large singular values and a number of tiny ones, then one can probably just harvest the largest singular values and discard the rest. Sometimes the rest of the spectrum is not tiny, but there?s a large gap between the head and the tail values. That would also be a reasonable cutoff. This method is requires visual inspection of the spectrum and hence cannot be performed as part of an automated pipeline.

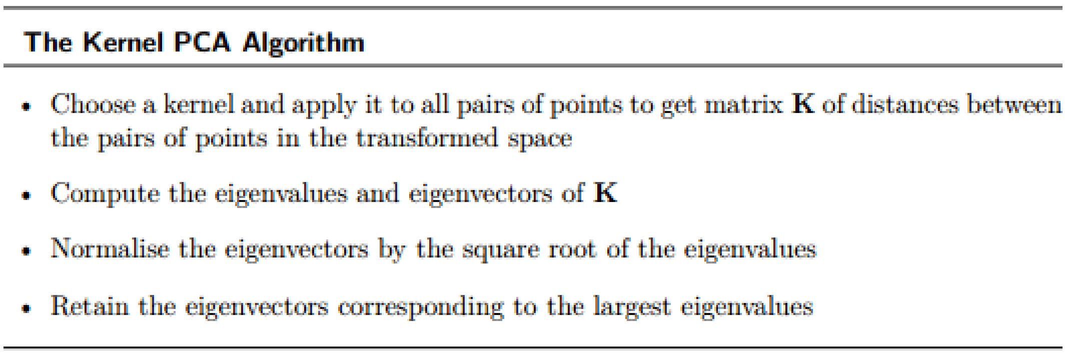
One key criticism of PCA is that the transformation is fairly complex, and the results are therefore hard to interpret. The principal components and the projected vectors are real-valued and could be positive or negative. The principal components are essentially linear combinations of the (centered) rows, and the projection values are linear combinations of the columns. In a stock returns application, for instance, each factor is a linear combination of time slices of stock returns. What does that mean? It is hard to express a human-understandable reason for the learned factors. Therefore, it is hard for analysts to trust the results. If you can?t explain why you should be putting billions of other people?s money into particular stocks, you probably won?t choose to use that model.

It is difficult to perform PCA in a streaming fashion, in batch updates, or from a sam? ple of the full data. Streaming computation of the SVD, updating the SVD, and computing the SVD from a subsample are all difficult research problems. Algorithms exist, but at the cost of reduced accuracy. One implication is that one should expect lower representational accuracy when projecting test data onto principal components found in the training set. As the distribution of the data changes, one would have to recompute the principal components in the current dataset.

Lastly, it is best not to apply PCA to raw counts (word counts, music play counts, movie viewing counts, etc.). The reason for this is that such counts often contain large outliers. (The probability is pretty high that there is a fan out there who watched The Lord of the Rings 314,582 times, which dwarfs the rest of the counts.) As we know, PCA looks for linear correlations within the features. Correlation and variance statistics are very sensitive to large outliers; a single large number could change the statistics a lot. So, it is a good idea to first trim the data of large values

Kernel PCA

One problem with PCA is that it assumes that the directions of variation are all straight lines. This is often not true. We can use the auto-associator with multiple hidden layers as just discussed, but there is a very nice extension to PCA that uses the kernel trick to get around this problem, just as the SVM got around it for the Perceptron. Just as is done there, we apply a (possibly non-linear) function Φ to each datapoint x that transforms the data into the kernel space, and then perform normal linear PCA in that space.



The Kernel PCA Algorithm

- Choose a kernel and apply it to all pairs of points to get matrix $\mathbf{K}$ of distances between the pairs of points in the transformed space
- Compute the eigenvalues and eigenvectors of $\mathbf{K}$
- Normalise the eigenvectors by the square root of the eigenvalues
- Retain the eigenvectors corresponding to the largest eigenvalues

Use Cases

- PCA reduces feature space dimensionality by looking for linear correlation patterns between features. Since it involves the SVD, PCA is expensive to compute for more than a few thousand features. But for small numbers of real-valued features, it is very much worth trying.

- PCA transformation discards information from the data. Thus, the downstream model may be cheaper to train, but less accurate. On the MNIST dataset, some have observed that using reduced-dimensionality data from PCA results in less accurate classification models. In these cases, there is both an upside and a downside to using PCA.

- One of the coolest applications of PCA is in anomaly detection of time series. Lakhina et al. (2004) used PCA to detect and diagnose anomalies in internet traffic. They focused on volume anomalies, i.e., when there is a surge or a dip in the amount of traffic going from one network region to another. These sudden changes may be indicative of a misconfigured network or coordinated denial-of-service attacks. Either way, knowing when and where such changes occur is valuable to internet operators.

- Since there is so much total traffic over the internet, isolated surges in small regions are hard to detect. A relatively small set of backbone links handle much of the traffic. Their key insight is that volume anomalies affect multiple links at the same time (because network packets need to hop through multiple nodes to reach their destination). Treat each of the links as a feature, and the amount of traffic at each time step as the measurement. A data point is a time slice of traffic measurements across all links on the network. The principal components of this matrix indicate the overall traffic trends on the network. The rest of the components represent the residual signal, which contains the anomalies.

- PCA is also often used in financial modeling. In those use cases, it works as a type of factor analysis, a term that describes a family of statistical methods that aim to describe observed variability in data using a small number of unobserved factors. In factor analysis applications, the goal is to find the explanatory components, not the transformed data.

- Financial quantities like stock returns are often correlated with each other. Stocks may move up and down at the same time (positive correlation), or move in opposite directions (negative correlation). In order to balance volatility and reduce risk, an investment portfolio needs a diverse set of stocks that are not correlated with each other. (Don?t put all your eggs in one basket if that basket is going to sink.) Finding strong correlation patterns is helpful for deciding on an investment strategy

- Stock correlation patterns can be industry-wide. For example, tech stocks may go up and down together, while airline stocks tend to go down when oil prices are high. But industry may not be the best way to explain the outcome. Analysts also look for unexpected correlations in observed statistics. In particular, the statistical factor model runs PCA on the matrix of time series of individual stock returns to find commonly covarying stocks. In this use case, the end goal is the principal components themselves, not the transformed data.

Lecture 3

Singular value decomposition

SVD is known under many different names. In the early days, as the above passage implies, it was called, ”factor analysis.” Other terms include principal component (PC) decomposition and empirical orthogonal function (EOF) analysis. All these are mathematically equivalent, although the way they are treated in the literature is often quite different.

The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler.

For the case of simplicity we will focus on the SVD for real-valued matrices and ignore the case for complex numbers.

Today, singular value decomposition has spread through many branches of science, in particular psychology and sociology, climate and atmospheric science, and astronomy. It is also extremely useful in machine learning and in both descriptive and predictive statistics.

SVD is a powerful technique in matrix computation and analysis, such as solving systems of linear equations and matrix approximation. SVD is also a well-known linear projection technique and has been widely used in data compression and visualization

Let D $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be a numerical data set in a d-dimensional space. Then D can be represented by an $n \times n$ matrix X as

$$X = (x_{ij})_{n \times d}$$

where x_{ij} is the j-component value of $\mathbf{x}_i$.

Let $\bar{\mu} = (\bar{\mu}_1, \bar{\mu}_2, \ldots, \bar{\mu}_d)$ be the column mean of X,

$$\bar{\mu}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}, \quad j = 1, 2, \ldots, d$$

and let $\mathbf{e}_n$ be a column vector of length n with all elements equal to one. Then SVD expresses $X - \mathbf{e}_n\bar{\mu}$ as

$$X - \mathbf{e}_n\bar{\mu} = USV^T$$

where U is an $n \times n$ column orthonormal matrix, i.e $V^H V = I$, where V^H is the conjugate transpose of V.

The columns of the matrix V are the eigenvalues of the covariance matrix C of X; precisely,

$$C = \frac{1}{n}X^T X - \bar{\mu}^T\bar{\mu} = V\Lambda V^T$$

Since C is a $d \times d$ positive semidefinite matrix, it has d nonnegative of C be ordered in decreasing order. $\lambda_1 \geq \lambda_2 \ldots \geq \lambda_d$. Let $\sigma_j (j = 1, 2, \ldots, d)$ be the standard deviation of the jth column of X, i.e,

$$\sigma_j = \left(\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{\mu}_j)^2\right)^{\frac{1}{2}}$$

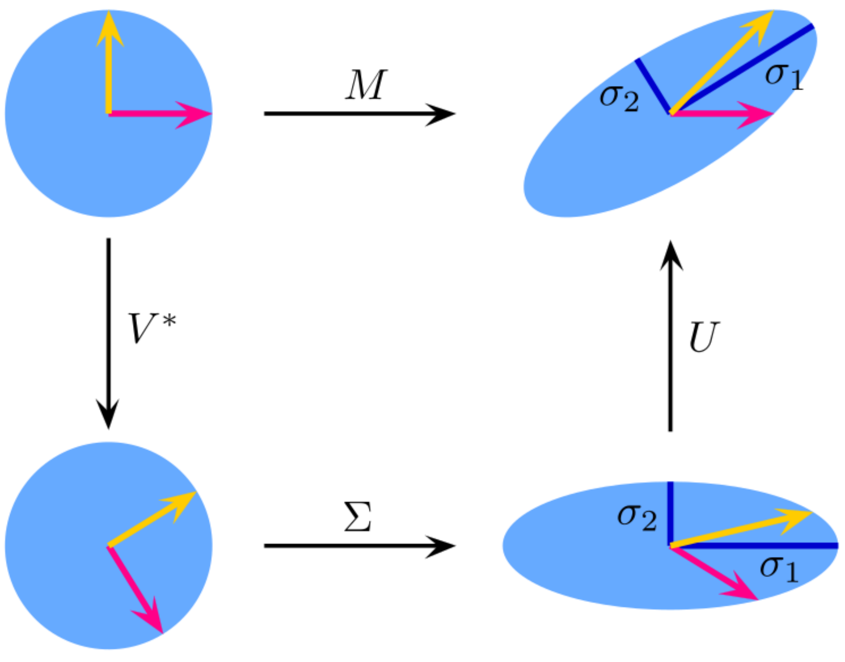
The trace Σ of C is invariant under rotation, i.e.,

$$\Sigma = \sum_{j=1}^{d}\sigma_j^2 = \sum_{j=1}^{d}\lambda_j$$

Noting that $\mathbf{e}_n^T X = n\bar{\mu}$ and $\mathbf{e}_n^T \mathbf{e}_n = n$ from equations,we have

$$\begin{aligned} VS^TSV^T &= VS^TU^TUSV^T \\ &= (X - \mathbf{e}_n\bar{\mu})^T (X - \mathbf{e}_n\bar{\mu}) \\ &= X^TX - \bar{\mu}^T\mathbf{e}_n^T X - X^T\mathbf{e}_n\bar{\mu} + \bar{\mu}^T\mathbf{e}_n^T\mathbf{e}_n\bar{\mu} \\ &= X^TX - n\bar{\mu}^T\bar{\mu} \\ &= nV\Lambda V^T \end{aligned}$$

$$s_j^2 = n\lambda_j, \quad j = 1, 2, \ldots, d$$



$$M = U \cdot \Sigma \cdot V^*$$

Application of SVD

The sum of the squares of the singular values should be equal to the total variance in A. Thus, the size of each tells you how much of the total variance is accounted for by each singular vector. You can create a truncated SVD containing, for instance, 99% of the variance:

$$a_{ij} \approx \sum_{k=1}^{p} u_{ik} s_k v_{jk}$$

where $p < n$ is the number of singular values that we?ve decided to keep. Typically, this will be fewer than the top ten (p=10) singular values. It is in this facility of singular value decomposition to exclude the least significant components of A that much of its power lies.

Solving matrix equations

Some more rearrangement of shows that SVD can be used for solving systems of linear equations:

$$A\boldsymbol{x} = \boldsymbol{b}$$

$$\boldsymbol{x} = VS^{-1}U^T\boldsymbol{b}$$

or, in summation notation:

$$x_i = \sum_j \frac{v_{ij}}{s_j} \sum_k u_{kj} b_k$$

If this was all there was to it, there would be little to recommend SVD over simpler matrix solvers, such as QR decomposition or even Gaussian elimination. In many cases, however, the matrix will be ill-conditioned, making the solution unstable so that it blows up or produces floating point overflow. This will show up in the singular values: the smallest ones will be very close to zero as measured relative to the largest. To produce a stable solution, we just throw these components away as in Equation above

It also generalizes to non-square matrices. Since an $m \times n$ matrix, where $m > n$, will have only n singular values, in SVD this is equivalent to solving an $m \times m$ matrix using only n singular values. For non-square matrices, matrix inversion using singular value decomposition is equivalent to solving the normal equation:

$$A^T A\boldsymbol{x} = A^T \boldsymbol{b}$$

and produces the solution for x that is closest to the origin, that is, of minimal length. The normal equation is the solution to the following minimization problem:

$$\min_x |Ax - b|$$

Thus, they are both generalized, linear, least squares fitting techniques.

Data reduction

A typical machine learning problem might have several hundred or more variables, while many machine learning algorithms will break down if presented with more than a few dozen. This makes singular value decomposition indispensable in ML for variable reduction.

We have already seen in Equation how an SVD with a reduced number of singular values can closely approximate a matrix. This can be used for data compression by storing the truncated forms of U, S, and V in place of A and for variable reduction by replacing A with U. Results will need to be transformed back to the original coordinate system by multiplying with S and V in accordance with Equation.

Lecture 4

Feature selection

Dimensionality reduction through the choice of an appropriate feature subset selection, results in multiple uses including performance upgrading, reducing the curse of dimensionality, promoting generalization abilities, speedup by depreciating computational power, growing model strength and lesser costs by avoiding "expensive" features.

Feature selection or variable selection is taken as the process employed for selecting the finest subset from the original features, primarily to solve the high dimensionality problem and avoid redundant and irrelevant features. There are plenty of advantages in using feature selection: reduction in training time, growing generalization performance, better classification results and excellent model interpretability. Although best classifiers are used for classification process there is no assurance in getting better results. So, Feature selection when combined with good classifiers there is more probability in getting good results

Feature selection varies from Feature extraction that generates new features by merging the original features. While Feature selection, is responsible in maintaining prototypes of the selected features, which is wellsuited in many fields. It is imperative and usually used Pre-Processing techniques, applied to the normalized training dataset and results in reduced feature set selection.

Adding all those features seems reasonable but it comes at a price: it increases the dimensionality of the patterns and thereby immerses the relevant information into a sea of possibly irrelevant, noisy or redundant features. How do we know when a feature is relevant or informative? This is what ”feature selection” is about

We are decomposing the problem of feature extraction in two steps: feature construction, briefly reviewed in the previous section, and feature selection, to which we are now directing our attention. Although feature selection is primarily performed to select relevant and informative features, it can have other motivations, including:

1. general data reduction, to limit storage requirements and increase algorithm speed;
2. feature set reduction, to save resources in the next round of data collection or during utilization;
3. performance improvement, to gain in predictive accuracy;
4. data understanding, to gain knowledge about the process that generated the data or simply visualize the data

Feature selection techniques prune away nonuseful features in order to reduce the complexity of the resulting model. The end goal is a parsimonious model that is quicker to compute, with little or no degradation in predictive accuracy. In order to arrive at such a model, some feature selection techniques require training more than one candidate model. In other words, feature selection is not about reducing training time in fact, some techniques increase overall training time?but about reducing model scoring time.

There are different feature selections algorithms which have achieved good results in many applications. They have the following objectives

1. Identify the feature subset that is basic and satisfies the target concept

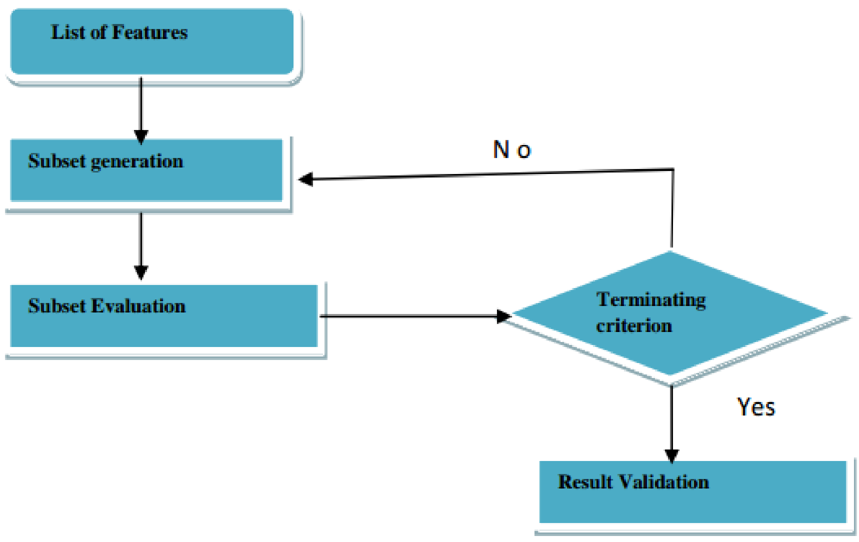
2. Choose a subset of L features from a set a K features, $L < M$, in such a way that the criterion function is maximized.

3. Select a subset of features for increasing prediction accuracy or reducing the size of the structure without considerably decreasing prediction accuracy of the classifier constructed using only the selected features

Filter Selection Procedure

Generally feature selection procedure encompass four essential steps:

1. subset generation;
2. subset evaluation;
3. terminating criterion
4. result validation



The approach starts with subset creation which uses scrupulous exploration trick to generate candidate feature subsets. Furthermore every candidate subset is assessed based on a precise estimation principle and differentiated with the earlier finest ones. If it is stronger among all others, then it substitutes the best preceding one. This procedure of subset establishment and assessment is performed repetitively, and stopped when a specified terminating condition is fulfilled. Lastly the chosen top feature subset is confirmed by earlier information. Eventually, search pattern and estimation criterion are two key issues in the research of feature selection. The General Feature Selection Procedure is shown

Lecture 5

Feature Ranking and Subset Selection

Feature Ranking

There are several feature selection algorithms, among them feature ranking gains more interest in research because of their ease in application, simplicity and finest empirical results. These approaches accord score or rank to the features based on certain criteria and exploit the ranks or scores for the selection mechanism. In terms of computation, feature ranking is more efficient than remaining feature selection techniques, as they compute M scores and sort all the scores accordingly. Then subsets of features are selected and applied to the respective classifier or predictor.

Learning to rank or machine-learned ranking (MLR) is the application of machine learning, typically supervised, semi-supervised or reinforcement learning, in the construction of ranking models for information retrieval systems.[2] Training data consists of lists of items with some partial order specified between items in each list. This order is typically induced by giving a numerical or ordinal score or a binary judgment (e.g. "relevant" or "not relevant") for each item. The ranking model's purpose is to rank, i.e. produce a permutation of items in new, unseen lists in a way which is "similar" to rankings in the training data in some sense.

As the research grows in the feature selection perspective, the researchers introduced varying feature selection criteria. In a related study Hsu hu has given association between rank phenomena and score phenomena through setting up a technique known as rank/score graph. They have demonstrated that subjected to definite circumstances, rank combination exceeds score combination. In another study the authors [33] successfully exploited rank combination in merging with different feature selection techniques. Hence, rankings obtained for features are merged with a weighted summation gained with each of the entry scores achieved by separate feature selection methods. Hence the combined approach ameliorates well than individual feature selection methods in several applications.

Subset Selection

Subset selection evaluates a subset of features as a group for suitability. Subset selection algorithms can be broken up into wrappers, filters, and embedded methods. Wrappers use a search algorithm to search through the space of possible features and evaluate each subset by running a model on the subset. Wrappers can be computationally expensive and have a risk of over fitting to the model. Filters are similar to wrappers in the search approach, but instead of evaluating against a model, a simpler filter is evaluated. Embedded techniques are embedded in, and specific to, a model.

Many popular search approaches use greedy hill climbing, which iteratively evaluates a candidate subset of features, then modifies the subset and evaluates if the new subset is an improvement over the old. Evaluation of the subsets requires a scoring metric that grades a subset of features. Exhaustive search is generally impractical, so at some implementor (or operator) defined stopping point, the subset of features with the highest score discovered up to that point is selected as the satisfactory feature subset. The stopping criterion varies by algorithm; possible criteria include: a subset score exceeds a threshold, a program's maximum allowed run time has been surpassed, etc.

Alternative search-based techniques are based on targeted projection pursuit which finds low-dimensional projections of the data that score highly: the features that have the largest projections in the lower-dimensional space are then selected.

Subset generation is done in two steps. First step is finding the search direction and the other one is identifying search strategy. Search direction can be forward, backward and random. If the search begins with empty set and adding features sequentially it is called forward search. If the search begins with a full set and removing features consecutively it is called backward search. The search may also begin by selecting features randomly so that local optima can be avoided, it is called random search.

In the second step, there are different search strategies for discovering an optimal or suboptimal feature subset. For a data set with N features, there exist 2N candidate subsets. An exhaustive search approach considers the entire 2N feature subsets to trace most excellent ones. It has the complexity of O (2N) given in terms of the data set dimensionality. There are other search strategies like sequential,complete, and exponential search strategies.

Subset Evaluation

After subset generation is completed, there is a need for evaluating each generated subset. Evaluation criteria are broadly classified into two types with respect to their reliance on learning algorithms that are accordingly pertained on the resultant feature subset. They are either independent criteria or dependent criteria. A few examples of prevailing independent criteria are information, dependency, distance, consistency measures. Usually an independent principle is mainly applied in algorithms with the filter model. Therefore a dependent principle is applied in the wrapper model, needs a preplanned learning algorithm in feature selection. It utilizes behavior of the learning algorithm applied for the chosen subset for adjudicating the features to be picked. Then considering whether they rely on the learning algorithms or not, feature selection algorithms are basically partitioned into three groups:filter, wrapper, and hybrid methods.

Lecture 6

Filter, Wrapper and Embedded methods

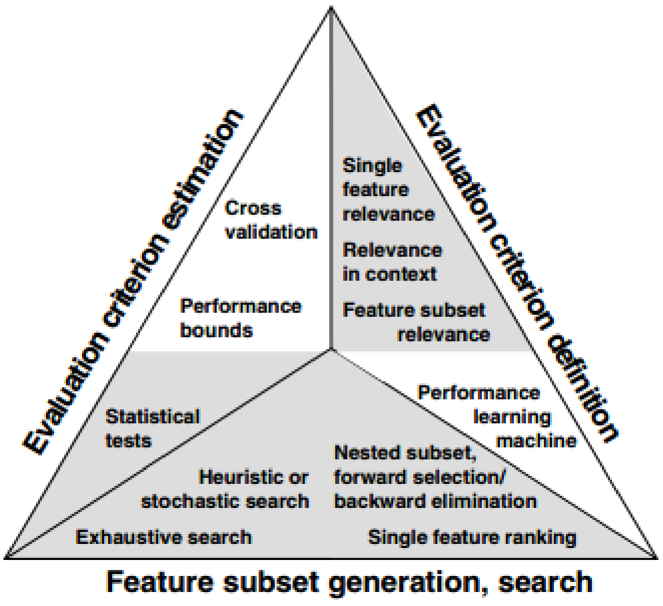
Feature selection techniques prune away nonuseful features in order to reduce the complexity of the resulting model. The end goal is a parsimonious model that is quicker to compute, with little or no degradation in predictive accuracy. In order to arrive at such a model, some feature selection techniques require training more than one candidate model. In other words, feature selection is not about reducing training timein fact, some techniques increase overall training timebut about reducing model scoring time.

Roughly speaking, feature selection techniques fall into three classes:

Filtering

Filtering techniques preprocess features to remove ones that are unlikely to be useful for the model. For example, one could compute the correlation or mutual information between each feature and the response variable, and filter out the features that fall below a threshold. Filtering techniques are much cheaper than the wrapper techniques described next, but they do not take into account the model being employed. Hence, they may not be able to select the right features for the model. It is best to do prefiltering conservatively, so as not to inadvertently eliminate useful features before they even make it to the model training step.

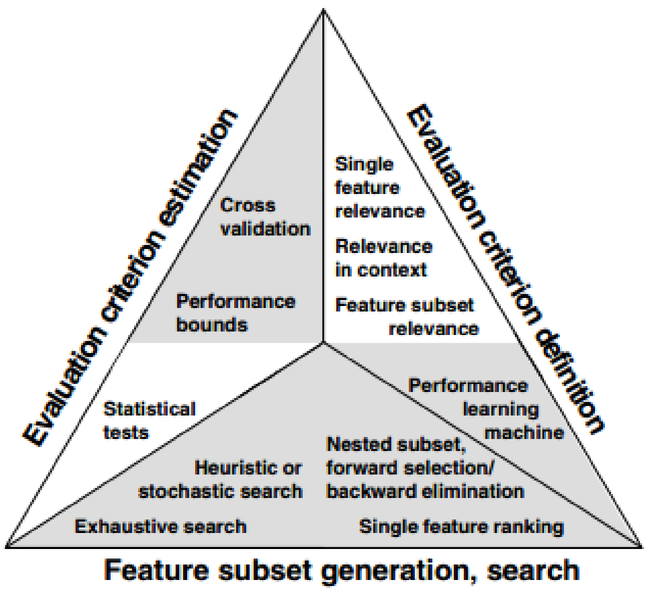
Filter methods work irrespective of any classifier; they filter out unimportant, unnecessary and noisy features using preprocessing steps before induction process starts. They give ranks to the features or feature subsets. They make use of intrinsic properties to detect feature subsets. functionalities with regard to target classes. It has many advantages over wrappers; they have excellent simplification characteristics since they are selfdetermining without any scrupulous learning method. Filters generally evaluate feature subsets by their data content using correlation measures, distance measures, gain ratio, principal component analysis (PCA), information gain etc. Filter techniques chuck features upon their evaluation, using data characteristics or using several kinds of statistical analysis, irrespective of any learning method involved.


Evaluation criterion estimation
Cross validation
Performance bounds
Statistical tests
Evaluation criterion definition
Single feature relevance
Relevance in context
Feature subset relevance
Performance learning machine
Heuristic or stochastic search
Nested subset, forward selection/ backward elimination
Exhaustive search
Single feature ranking
Feature subset generation, search

(a) Filters

Wrapper methods

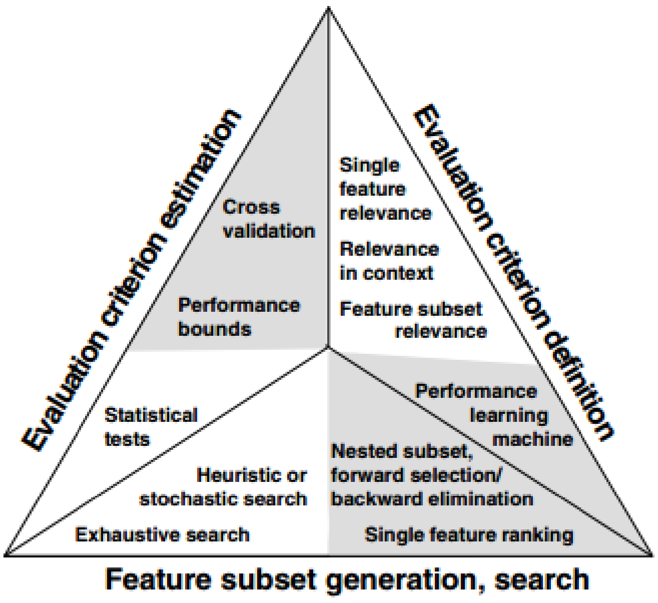
Wrapper Method exploits a learning algorithm for assessing features or feature subsets through their predictive accuracy. They need more computation time and slower because of the repeating process; however they give more accurate results than filter procedure. This is the disadvantage for high dimensional data but it could be overwhelmed by using a fast learning algorithm. Additionally, the subset assessment is strongly connected with a classifier used; wrappers must be therefore has the capability in repeating the process, whenever another classifier is employed for feature evaluation. These techniques are expensive, but they allow you to try out subsets of features, which means you won't accidentally prune away features that are uninformative by themselves but useful when taken in combination. The wrapper method treats the model as a black box that provides a quality score of a proposed subset for features. There is a separate method that iteratively refines the subset.


Evaluation criterion estimation
Evaluation criterion definition
Cross validation
Performance bounds
Statistical tests
Single feature relevance
Relevance in context
Feature subset relevance
Performance learning machine
Nested subset, forward selection/ backward elimination
Heuristic or stochastic search
Exhaustive search
Single feature ranking
Feature subset generation, search

(b) Wrappers

Embedded methods

These methods perform feature selection as part of the model training process. For example, a decision tree inherently performs feature selection because it selects one feature on which to split the tree at each training step. Another example is the l^1 regularizer, which can be added to the training objective of any linear model. The l^1 regularizer encourages models that use a few features as opposed to a lot of features, so it?s also known as a sparsity constraint on the model. Embedded methods incorporate feature selection as part of the model training process. They are not as powerful as wrapper methods, but they are nowhere near as expensive. Compared to filtering, embedded methods select features that are specific to the model. In this sense, embedded methods strike a balance between computational expense and quality of results.


Evaluation criterion estimation
Evaluation criterion definition
Cross validation
Performance bounds
Single feature relevance
Relevance in context
Feature subset relevance
Performance learning machine
Statistical tests
Heuristic or stochastic search
Exhaustive search
Nested subset, forward selection/ backward elimination
Single feature ranking
Feature subset generation, search

(c) Embedded methods

Lecture 7

Evaluating Machine Learning algorithms

We want to learn a hypothesis that fits the future data best. To make that precise we need to define "future" and "best" We make the **stationarity assumption**: that there is a probability distribution over examples that remains stationary over time. Each example data point (before we see it) is a random variable E_j whose observed value $e_j = (x_j, y_j)$ is sampled from that distribution, and is independent of the previous examples:

$$\mathbf{P}(E_j|E_{j-1}, E_{j-2}, \ldots) = \mathbf{P}(E_j)$$

and each example has an identical prior probability distribution:

$$\mathbf{P}(E_j) = \mathbf{P}(E_{j-1}) = \mathbf{P}(E_{j-2}) = \ldots$$

Examples that satisfy these assumptions are called independent and identically distributed or I.I.D. i.i.d.. An i.i.d. assumption connects the past to the future; without some such connection, all bets are off?the future could be anything. (We will see later that learning can still occur if there are slow changes in the distribution.) ERROR RATE The next step is to define ?best fit.? We define the error rate of a hypothesis as the proportion of mistakes it makes?the proportion of times that h(x) $h(x) \neq y$ for an (x, y) example. Now, just because a hypothesis h has a low error rate on the training set does not mean that it will generalize well. A professor knows that an exam will not accurately evaluate students if they have already seen the exam questions. Similarly, to get an accurate evaluation of a hypothesis, we need to test it on a set of examples it has not seen yet. The simplest approach is the one we have seen already: randomly split the available data into a training set from which the learning algorithm produces h and a test set on which the accuracy of h is evaluated. This method, sometimes called **holdout cross-validation**, has the disadvantage that it fails to use all the available data; if we use half the data for the test set, then we are only training on half the data, and we may get a poor hypothesis. On the other hand, if we reserve only 10% of the data for the test set, then we may, by statistical chance, get a poor estimate of the actual accuracy. We can squeeze more out of the data and still get an accurate estimate using a technique called **k-fold cross-validation**. The idea is that each example serves double duty?as training data and test data. First we split the data into k equal subsets. We then perform k rounds of learning; on each round $1/k$ of the data is held out as a test set and the remaining examples are used as training data. The average test set score of the k rounds should then be a better estimate than a single score. Popular values for k are 5 and 10?enough to give an estimate that is statistically likely to be accurate, at a cost of 5 to 10 times longer computation time. The extreme is k = n, also known as **leave-one-out cross-validation** . **LOOCV**. Despite the best efforts of statistical methodologists, users frequently invalidate their results by inadvertently peeking at the test data. Peeking can happen like this: A learning algorithm has various ?knobs? that can be twiddled to tune its behavior?for example, various different criteria for choosing the next attribute in decision tree learning. The researcher generates hypotheses for various different settings of the knobs, measures their error rates on the test set, and reports the error rate of the best hypothesis. The reason is that the hypothesis was selected on the basis of its test set error rate, so information about the test set has leaked into the learning algorithm.

Peeking is a consequence of using test-set performance to both choose a hypothesis and evaluate it. The way to avoid this is to really hold the test set out-lock it away until you are completely done with learning and simply wish to obtain an independent evaluation of the final hypothesis. (And then, if you don?t like the results ... you have to obtain, and lock away, a completely new test set if you want to go back and find a better hypothesis.) If the test set is locked away, but you still want to measure performance on unseen data as a way of selecting a good hypothesis, then divide the available data (without the test set) into a training set and a **validation set**. The next section shows how to use validation sets to find a good tradeoff between hypothesis complexity and goodness of fit.

Error rates to loss

Consider the problem of classifying email messages as spam or non-spam. It is worse to classify non-spam as spam (and thus potentially miss an important message) then to classify spam as non-spam (and thus suffer a few seconds of annoyance). So a classifier with a 1% error rate, where almost all the errors were classifying spam as non-spam, would be better than a classifier with only a 0.5% error rate, if most of those errors were classifying non-spam as spam. We saw that decision-makers should maximize expected utility, and utility is what learners should maximize as well. In machine learning it is traditional to express utilities by means of a **loss function**. The loss function $L(x, y, \hat{y})$ is defined as the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$:

$$
\begin{aligned}
L(x, y, \hat{h}) = \; & \mathit{Utility}(\text{result of using } y \text{ given an input}) \\
& - \mathit{Utility}(\text{result of using } \hat{y} \text{ given an input } x)
\end{aligned}
$$

This is the most general formulation of the loss function. Often a simplified version is used, $L(y, \hat{y})$, that is independent of x. We will use the simplified version for the rest of this chapter, which means we can?t say that it is worse to misclassify a letter from Mom than it is to misclassify a letter from our annoying cousin, but we can say it is 10 times worse to classify non-spam as spam than vice-versa:

$$L(\mathit{spam}, \mathit{nospam}) = 1, \quad L(\mathit{nospam}, \mathit{spam}) = 10$$

Note that $L(y, y)$ is always zero; by definition there is no loss when you guess exactly right. For functions with discrete outputs, we can enumerate a loss value for each possible misclassification, but we can?t enumerate all the possibilities for real-valued data. If $f(x)$ is 137.035999, we would be fairly happy with $h(x) = 137.036$, but just how happy should we be? In general small errors are better than large ones; two functions that implement that idea are the absolute value of the difference (called the L_1 loss), and the square of the difference (called the L_2 loss). If we are content with the idea of minimizing error rate, we can use the $L_{0/1}$ loss function, which has a loss of 1 for an incorrect answer and is appropriate for discrete-valued outputs:

$$
\begin{aligned}
\text{Absolute value loss:} \quad & L_1(y, \hat{y}) = |y - \hat{y}| \\
\text{Squared error loss:} \quad & L_2(y, \hat{y}) = (y - \hat{y})^2 \\
\text{0/1 loss:} \quad & L_{0/1}(y, \hat{y}) = 0 \quad \text{if} \quad y = \hat{y}, \text{else} 1
\end{aligned}
$$

The learning agent can theoretically maximize its expected utility by choosing the hypothesis that minimizes expected loss over all input-output pairs it will see. It is meaningless to talk about this expectation without defining a prior probability distribution, $P(X, Y)$ over examples. Let E be the set of all possible input?output examples. Then the expected generalization loss for a hypothesis h (with respect to loss function L) is

There are four reasons why $\hat{y}*$ may differ from the true function, f: unrealizability, variance, noise, and computational complexity. First, f may not be realizable-may not be in may be present in such a way that other hypotheses are preferred. Second, a learning algorithm will return different hypotheses for different sets of examples, even if those sets are drawn from the same true function f, and those hypotheses will make different predictions on new examples. The higher the variance among the predictions, the higher the probability of significant error. Note that even when the problem is realizable, there will still be random variance, but that variance decreases towards zero as the number of training examples inNOISE creases. Third, f may be nondeterministic or noisy-it may return different values for $f(x)$ each time x occurs. By definition, noise cannot be predicted; in many cases, it arises because the observed labels y are the result of attributes of the environment not listed in x. And finally, when H is complex, it can be computationally intractable to systematically search the whole hypothesis space. The best we can do is a local search (hill climbing or greedy search) that explores only part of the space. That gives us an approximation error. Combining the sources of error, we?re left with an estimation of an approximation of the true function f.

Traditional methods in statistics and the early years of machine learning concentrated on small-scale learning, where the number of training examples ranged from dozens to the low thousands. Here the generalization error mostly comes from the approximation error of not having the true f in the hypothesis space, and from estimation error of not having enough training examples to limit variance. In recent years there has been more emphasis on large scale learning, often with millions of examples. Here the generalization error is dominated by limits of computation: there is enough data and a rich enough model that we could find an h that is very close to the true f, but the computation to find it is too complex, so we settle for a sub-optimal approximation.

Regularization

An alternative approach is to search for a hypothesis that directly minimizes the weighted sum of empirical loss and the complexity of the hypothesis, which we will call the total cost:

$$Cost(h) = EmpLoss(h) + \lambda\, Complexity(h)$$
$$\hat{h}^{*} = argmin\, Cost(h)$$

Here λ is a parameter, a positive number that serves as a conversion rate between loss and hypothesis complexity (which after all are not measured on the same scale). This approach combines loss and complexity into one metric, allowing us to find the best hypothesis all at once. Unfortunately we still need to do a cross-validation search to find the hypothesis that generalizes best, but this time it is with different values of λ rather than size. We select the value of λ that gives us the best validation set score. This process of explicitly penalizing complex hypotheses is called regularization (because it looks for a function that is more regular, or less complex). Note that the cost function requires us to make two choices: the loss function and the complexity measure, which is called a regularization function. The choice of regularization function depends on the hypothesis space. For example, a good regularization function for polynomials is the sum of the squares of the coefficients-keeping the sum small would guide us away from the wiggle polynomials.

Another way to simplify models is to reduce the dimensions that the models work with. A process of **feature selection** can be performed to discard attributes that appear to be irrelevant. χ^2 pruning is a kind of feature selection.

Lecture 8

Model Selection

Features and models sit between raw data and the desired insights. In a machine learning workflow, we pick not only the model, but also the features. This is a double-jointed lever, and the choice of on affects the other. Good features make the subsequent modeling step easy and the resulting model more capable of completing the desired task. Bad features may require a much more complicated model to achieve the same level of performance

When we have a variety of models of different complexity (e.g., linear or logistic regression models with different degree polynomials, or KNN classifiers with different values of K), how should we pick the right one? A natural approach is to compute the **misclassification rate** on the training set for each method. This is defined as follows:

$$\mathrm{err}(f, \mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(f(\mathbf{x} \neq y_i))$$

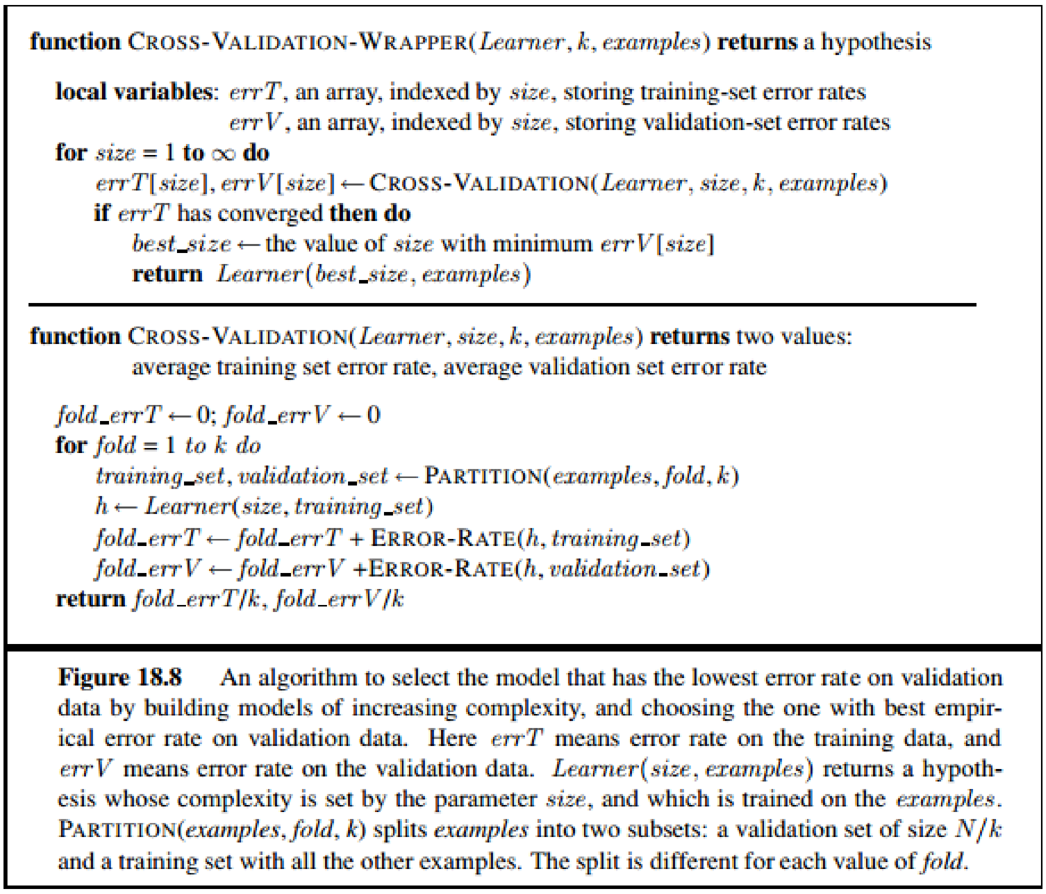
where $f(x)$ is our classifier. In Figure 1.21(a), we plot this error rate vs K for a KNN classifier (dotted blue line). We see that increasing K increases our error rate on the training set, because we are over-smoothing. As we said above, we can get minimal error on the training set by using K = 1, since this model is just memorizing the data.

However, what we care about is generalization error, which is the expected value of the misclassification rate when averaged over future data (see Section 6.3 for details). This can be approximated by computing the misclassification rate on a large independent test set, not used during model training. We plot the test error vs K in solid red (upper curve). Now we see a U-shaped curve: for complex models (small K), the method overfits, and for simple models (big K), the method underfits. Therefore, an obvious way to pick K is to pick the value with the minimum error on the test set (in this example, any value between 10 and 100 should be fine).

Unfortunately, when training the model, we don?t have access to the test set (by assumption), so we cannot use the test set to pick the model of the right complexity.11 However, we can create a test set by partitioning the training set into two: the part used for training the model, and a second part, called the validation set, used for selecting the model complexity. We then fit all the models on the training set, and evaluate their performance on the validation set, and pick the best. Once we have picked the best, we can refit it to all the available data. If we have a separate test set, we can evaluate performance on this, in order to estimate the accuracy of our method.

Often we use about 80% of the data for the training set, and 20% for the validation set. But if the number of training cases is small, this technique runs into problems, because the model won?t have enough data to train on, and we won?t have enough data to make a reliable estimate of the future performance.

A simple but popular solution to this is to use **cross validation (CV)**. The idea is simple: we split the training data into K folds; then, for each fold $k \in \{1, \ldots K\}$, we train on all the folds but the k?th, and test on the k?th, in a round-robin fashion, as sketched in figure. We then compute the error averaged over all the folds, and use this as a proxy for the test error. (Note that each point gets predicted only once, although it will be used for training $K?1$ times.) It is common to use K = 5; this is called 5-fold CV. If we set K = N, then we get a method called **leave-one out cross validation**, or **LOOCV**, since in fold i, we train on all the data cases except for i, and then test on i. Choosing K for a KNN classifier is a special case of a more general problem known as model selection, where we have to choose between models with different degrees of flexibility. Crossvalidation is widely used for solving such problems, although we will discuss other approaches later in the book.



```
function CROSS-VALIDATION-WRAPPER($Learner, k, examples$) returns a hypothesis
  local variables: $errT$, an array, indexed by $size$, storing training-set error rates
                   $errV$, an array, indexed by $size$, storing validation-set error rates
  for $size$ = 1 to $\infty$ do
    $errT[size], errV[size] \leftarrow$ CROSS-VALIDATION($Learner, size, k, examples$)
    if $errT$ has converged then do
      $best\_size \leftarrow$ the value of $size$ with minimum $errV[size]$
      return $Learner(best\_size, examples)$

function CROSS-VALIDATION($Learner, size, k, examples$) returns two values:
    average training set error rate, average validation set error rate
  $fold\_errT \leftarrow 0$; $fold\_errV \leftarrow 0$
  for $fold$ = 1 $to$ $k$ $do$
    $training\_set, validation\_set \leftarrow$ PARTITION($examples, fold, k$)
    $h \leftarrow Learner(size, training\_set)$
    $fold\_errT \leftarrow fold\_errT$ + ERROR-RATE($h, training\_set$)
    $fold\_errV \leftarrow fold\_errV$ + ERROR-RATE($h, validation\_set$)
  return $fold\_errT/k, fold\_errV/k$
```

Figure 18.8 An algorithm to select the model that has the lowest error rate on validation data by building models of increasing complexity, and choosing the one with best empirical error rate on validation data. Here $errT$ means error rate on the training data, and $errV$ means error rate on the validation data. $Learner(size, examples)$ returns a hypothesis whose complexity is set by the parameter $size$, and which is trained on the $examples$. PARTITION($examples, fold, k$) splits $examples$ into two subsets: a validation set of size N/k and a training set with all the other examples. The split is different for each value of $fold$.

No free lunch theorem

All models are wrong, but some models are useful. ? George Box

Much of machine learning is concerned with devising different models, and different algorithms to fit them. We can use methods such as cross validation to empirically choose the best method for our particular problem. However, there is no universally best model - this is sometimes called the no free lunch theorem (Wolpert 1996). The reason for this is that a set of assumptions that works well in one domain may work poorly in another. As a consequence of the no free lunch theorem, we need to develop many different types of models, to cover the wide variety of data that occurs in the real world. And for each model, there may be many different algorithms we can use to train the model, which make different speed-accuracy-complexity tradeoffs. It is this combination of data, models and algorithms that we will be studying in the subsequent chapters.