Data cleansing, data cleaning, or data scrubbing is the process of **detecting and correcting (or removing) corrupt or inaccurate records** from a record saet, table, or database and refers to **identifying incomplete, incorrect, inaccurate or irrelevant parts of the data** and then **replacing, modifying, or deleting the dirty or coarse data.**

Reintroduction to stats

Sample space S is the set of all possible events we might observe. An event is a subset of the sample space.

A *random variable* is a mapping from the event space to a number (or vector.) (X, realization in lower case)

Probability Mass Function (PMF) - Requirement: P*x*2X *pX*(*x*) = 1
Cumulative Distribution Function (CDF)

E[X] = Expected value or mean

X and *Y* have a **joint distribution** if their realizations come together as a pair.

Supervised Learning

Training experience: a set of ***labeled examples*** of the form <*x*1*, x*2*, . . . xp, y*> where xj are feature values and y is the output

input variables or ***features*** or *attributes,* ***labels*** or *output variables* or *targets*

The ith training example has the form: $\langle x_{1,i}, \ldots x_{p,i}, y_i \rangle$ where p is the number of features (32 in our case).

Notation $\mathbf{x}_i$ denotes a **column** vector with elements $x_{1,i}, \ldots x_{p,i}$.

The training set D consists of n training examples

We denote the $n \times p$ matrix of features by X and the size-n column vector of outputs from the data set by $\mathbf{y}$.

In statistics, X is called the data matrix or the *design matrix.*

$\mathcal{X}$ denotes space of input values

$\mathcal{Y}$ denotes space of output values

h is called a predictive model or hypothesis

Steps to solving a supervised learning problem
1. Decide what the input-output pairs are.
2. Decide how to encode inputs and outputs.
This defines the input space X, and the output space Y.
(We will discuss this in detail later)
3. Choose model space/hypothesis class H .

Linear hypothesis - *h***w**(**x**) = *w*0 + *w*1*x*1 + *w*2*x*2 + · · ·,

Error minimization - use error function or cost function

- Least mean squares (LMS)
- **Least Squares Solution**

Performance Evaluation

A “statistic” is the result of applying a function (summary) to the data
(E.g. sample median)

A ***parameter*** is any summary of the distribution of a random variable. (E.g. *μX*, median.) A parameter is the result of a ***function*** applied to a distribution.

Estimation uses a ***statistic*** (e.g. ¯*xn*) to estimate a ***parameter*** (e.g. *μX*) of the ***distribution*** of a ***random variable***.

Bias – The **expected difference** between estimator and parameter. (0 = unbiased)

Variance - The **expected squared difference** between estimator and its mean

Choosing Performance Measures for Regression: Mean Errors
- (Root) Mean Squared Error
- Mean absolute error (easier to interpret)

Choosing Performance Measures for Regression: Mean Relative Error

Model Selection

Performance: We would like to estimate the **generalization error** of our resulting predictor.
Model selection: We would like to choose the best model space (e.g. linear, quadratic, …) **for the data we have**

Small training error, large generalization error is known as **overfitting**

Model Selection Strategy 1: A Validation Set
• A separate **validation set** can be used for model selection.
– Train on the training set using each proposed model space
– Evaluate each on the validation set, identify the one with lowest *validation* error
– Choose the simplest model with performance < 1 std. error worse than the best.

Single-Partition Approach - The data is randomly partitioned into three disjoint subsets:
– A *training set* used only to find the parameters **w**
– A *validation set* used to find the right model space (e.g., the degree of the polynomial)
– A *test set* used to estimate the generalization error of the resulting model
Cons:
– Smaller effective training sets make performance and performance estimates more variable.
– Small validation sets can give poor model selection
– Small test sets can give poor estimates of performance

k-fold cross-validation
Loop through the partitions $i = 1...k$:
– Partition i is for evaluation (i.e., estimating the performance of the algorithm after learning is done)
– The rest are used for training (i.e., choosing the specific model within the space)
"Cross-Validation Error" is the average error on the evaluation partitions.
Extra loop through model spaces for model selection

Classification

By linear models, we mean that the hypothesis function $h\mathbf{w}(\mathbf{x})$ is a (transformed) *linear function of the parameters* **w**.

Linear Methods for Classification
• Classification tasks
• Loss functions for classification
• Logistic Regression
• Support Vector Machines

Large Margin Classifiers:
Linear Support Vector Machines
• Linear classifiers that focus on learning the *decision boundary* rather than the conditional distribution $P(Y = y|\mathbf{X} = \mathbf{x})$
– Perceptrons
* output a **class**, not a probability
– "Margin" idea and max margin classifiers
– (Linear) support vector machines
* Formulation as optimization problem

Perceptron convergence theorem - *If* classes are linearly separable ***then*** the perceptron learning rule will find a separater after some finite number of updates.

Support Vector Machines
• Support vector machines (SVMs) for binary classification can be viewed as a way of training perceptrons
• Three main new ideas:
– A optimization criterion (the "margin") guarantees uniqueness and has theoretical advantages
– Natural handling nonseparable data by allowing mistakes
– An efficient way of operating in expanded feature spaces: "kernel trick"
• SVMs can also be used for multiclass classification and regression.

For a given separating hyperplane, the ***margin*** is two times the (Euclidean) distance from the hyperplane to the nearest training example.

SVMs are a state-of-the-art for classification when you don't need probability estimates

Nonlinear models

Kernel functions - Whenever a learning algorithm (such as SVMs) can be written in terms of dot-products, it can be generalized to kernels. A *kernel* is any function $K : \mathrm{R}n \times \mathrm{R}n$ *7!* R which corresponds to a dot product for some feature mapping ϕ

Polynomial kernels
• More generally, $K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})d$ is a kernel, for any positive integer *d*.
• If we expanded the product above, we get terms for all degrees up to and including *d* (in *xi* and *zi*).
• If we use the primal form of the SVM, each of these will have a weight associated with it!
• *Curse of dimensionality:* it is very expensive both to optimize and to predict with an SVM in primal form with many features.
Many kernels are available:
– Information diffusion kernels (Lafferty and Lebanon, 2002)
– Diffusion kernels on graphs (Kondor and Jebara 2003)
– String kernels for text classification (Lodhi et al, 2002)
– String kernels for protein classification (e.g., Leslie et al, 2002)
… and others!

Artificial Neural Networks - Functional form + Training algorithm

Traditional ANN training is done by looping over examples (much like the perceptron) and taking a small step down the single-example gradient.

Backpropagation - Adaptive learning rate

Deep learning tricks – Pre-training, dropout, ReLUs

Decision Trees
• Non-parametric learning
• *k*-nearest neighbour
• Efficient implementations
• Variations

Parametric supervised learning - summarize the data using the parameters
Non-parametric (memory-based) learning methods – just store all training examples

One-nearest neighbor
• Given: Training data $\{(\mathbf{x}i, yi)\}n\ i$=1, distance metric *d* on *X*.
• Training: Nothing to do! (just store data)
• Prediction: for $\mathbf{x}$ *2 X*
– Find nearest training sample to $\mathbf{x}$

• Nearest-neighbor does not explicitly compute decision boundaries
• But the effective decision boundaries are a subset of the *Voronoi diagram* for the training data

What kind of distance metric?
• Euclidian distance

• Maximum/minimum difference along any axis
• Weighted Euclidian distance (with weights based on domain knowledge)

Distance metric tricks
• You may need to do preprocessing:
– *Scale* the input dimensions (or normalize them)
– Remove noisy inputs
– Determine weights for attributes based on cross-validation (or information-theoretic methods)
• Distance metric is often domain-specific
– E.g. string edit distance in bioinformatics
– E.g. trajectory distance in time series models for walking data
• Distance metric can be learned sometimes (more on this later)

***k*-nearest neighbor**
Bias-variance trade-off
• If *k* is low, very non-linear functions can be approximated, but we also capture the noise in the data Bias is low, variance is high
• If *k* is high, the output is much smoother, less sensitive to data variation
High bias, low variance
• A validation set can be used to pick the best *k*

Locally-weighted regression
• Weighted regression: different weights in the error function for different points
• Locally weighted regression: weights *depend on the distance to the query point*

LOESS Smoothing
• Quadratic Regression
• Uses the closest _ percent of the training set to make each prediction, called “span”

Generalized Additive Models
• Also smooth functions of the input variables; appearance similar to LOESS but with deeper theory.
• Based on regression splines

Lazy and eager learning
• *Lazy*: wait for query before generalizing
E.g. Nearest Neighbor
• *Eager*: generalize before seeing query
E.g. SVM, Linear regression

Pros and cons of lazy and eager learning
• Eager learners must create global approximation
• Lazy learners can create many local approximations
• An eager learner does the work off-line, summarizes lots of data with few parameters
• A lazy learner has to do lots of work sifting through the data at query time
• Typically lazy learners take longer time to answer queries and require more space

When to consider nonparametric methods
• When you have: instances that map to points in R*p*, not too many attributes per instance (< 20), lots of data

Decision trees, more formally
• Each internal node contains a *test*, on the value of one (typically) or more feature values
• A test produces discrete outcomes

How do we learn decision trees?
• Usually, decision trees are constructed in two phases:
1. An recursive, top-down procedure “grows” a tree
(possibly until the training data is completely fit)
2. The tree is “pruned” back to avoid overfitting
• Both typically use *greedy heuristics*

The further *P* is from uniform, the lower the entropy

Information gain – How much does the entropy of *Y* go down, on average, if I am told the value of *X*? (Determine best test)

Unsupervised Learning

Regulation **-** Ask for a weight vector that has low training error but is also small

Dimensionality Reduction Techniques
Axis-aligned: Remove features that are well-predicted by other features
Linear: **Principal components analysis** creates smaller set of new features that are weighted sums of existing features
Non-linear: Create small set of new features that are non-linear functions of existing features

- Kernel PCA
- Independent components analysis
- Self-organizing maps
- Multi-dimensional scaling
- **t-SNE: t-distributed Stochastic Neighbour Embedding**

Principal Component Analysis

Given: instances, each being a length- real vector.
Suppose we want a 1-dimensional representation of that data, instead of -dimensional.
Specifically, we will:

- Choose a line in that "best represents" the data.
- Assign each data object to a point along that line.

Identifying a point on a line just requires a scalar

Uses of PCA

- Pre-processing for a supervised learning algorithm, e.g. for image data, robotic sensor data
- Used with great success in image and speech processing
- Visualization
- Exploratory data analysis
- Removing the linear component of a signal (before fancier non-linear models are applied)

Reconstruction error

Singular Value Decomposition – b, the eigenvalues , the vj, and the projections of the instances can all be computing in polynomial time, e.g. using (thin) Singular Value Decomposition.

Clustering is grouping similar objects together.

K-means clustering

- One of the most commonly-used clustering algorithms, because it is easy to implement and quick to
- run.
- Assumes the objects (instances) to be clustered are -dimensional vectors, xi.
- Uses a distance measure between the instances (typically Euclidean distance)
- The goal is to *partition* the data into K disjoint subsets

Single-linkage - Favors spatially-extended / filamentous clusters