1 Introduction to Data Mining

If we look at the change in processing speed and the speed of data collection, we can deduce that every 3 years the processing speed increases by a factor 4 and the collected data by a factor 8.

Definition 1.1 (Moore's Law)**.** Processing speed doubles every 18 months

Definition 1.2 (Kryder's Law)**.** Hard disk capacity doubles every 12 months

Definition 1.3 (Lyman & Varian)**.** The amount of collected data doubles every year

Meaning, every 3 years, the overflow of data (or flood) is doubling. Besides the total amount of data, the shape of the data has also changed over time. This new data is also called Big Data, generally characterized by the "Three Vs of Big Data" (IBM):

Definition 1.4 (Three Vs of Big Data (IBM))**.**

- **Volume** - Large size of data (terabytes, petabytes).
- **Velocity** - Speed of added data is fast, and needs to be processed fast.
- **Variety** - Structured and unstructured data, such as text, sensor data, audio, video, click streams, log files and more.

To be able to process such big data, we need to do fuzzy queries and we want to be able to run the processes in a streaming fashion. Traditionally these are easy to execute, but complex in this context. The trade-off therein is accuracy for speed (a rough answer in real-time is sufficient).

1.1 Examples

1.1.1 Recommender Systems

Given petabytes of sales data in the form *<user_id, item_id>* we want to do recommendations to the *current_user_id* that just clicked or bought *current_item_id* and we have only milliseconds to do this.

This is a fairly simplified scenario, as you generally have a lot more information on users to do these recommendations.

1.1.2 Data Stream Mining

In some situations, the incoming stream of data is too large or too fast to be stored, and needs to be analyzed on-the-fly with limited memory.

Examples of this are to detect fraud in electronic transactions, showing the correct advertisements, prognostic health monitoring of a fighter jet, detection of DoS attacks, or returning rankings from current interests or click streams from users.

1.2 Data Mining Paradigms

We can group data mining activities into two separate paradigms: supervised and unsupervised learning.

Definition 1.5 (Supervised Learning)**.** We have a set of defined inputs and defined outputs and try to build a model that converts input into these outputs

Definition 1.6 (Unsupervised Learning)**.** The outputs are characteristics and generally tries to learn similarity from the data.

1.3 Overfitting

If we would train our model on all the data that we have, we have the risk to overfit on the data. This means that it follows the data closely, but will not work with unseen data. Thus, we separate the training data from test and evaluation data, and use approaches to avoid overfitting on the training data.

For example, we can use k-Fold Cross-Validation where we split the training data (stratified) in to k folds and use $(k-1)$ folds for training and 1 for testing. Repeat this k times and average the results. A downside of this is that it is computationally expensive (but parallelizable).

2 Recommender Systems

There are different approaches to recommender system, the idea is basically that you have a mapping of users onto items and use this mapping to learn the value of an item to a user that is not in the mapping.

Generally, we have a matrix of the size users $\times$ items and it is generally filled with ratings, interest, history or some other value that represents the relation. Such matrices are very large and usually sparse.

2.1 Naive Approach

The baseline approach is to take the mean of the entire matrix, item or user. A bit more advanced we can do a linear regression modeled as:

Definition 2.1 (Naive Linear Recommender)**.**

$$R_{\text{user-item}}(u, i) = \alpha \times R_{\text{user}}(u, i) + \beta \times R_{\text{item}}(u, i)$$

The linear regression returns good results, is easy to calculate and update, and allows for simple interpretation of the model (good movie, harsh user, crowd follower).

2.2 Content-Based Approach

The intuition for Content-based Approaches is to construct a profile for every item and a profile of every user, and see which items are closest to the user profile. The profile generally contains labels such as budget, genre, origin, cast, et cetera. The baseline would be to take the mean of each dimension of the profile for a user.

2.2.1 Model-Based Approach

The content-based approach assumes that you can fill in these parameters for users yourself (or source it from the user themselves). However, we can also train a model per user that predicts rating from the item profiles. Pitfalls here are that it is expensive to build and maintain, the accuracy is low and it doesn't work with new users.

2.3 Collaborative Filtering

An alternative to item-specific profiles is to recommend items to users based on what similar users have liked. You can either do user-to-user collaborative filtering or item-to-item collaborative filtering.

User-User Recommendations Each user has a vector with ratings for every item that they have rated. Users with similar vectors are selected as neighbours. We then take the top L neighbours and aggregate their ratings to predict the rating.

Item-Item Recommendations Each item has a vector with ratings for every user that have rated it. Items with similar users are selected as neighbours. We then take the top L neighbours and aggregate their ratings to predict the rating.

The main differences here are that for either category insertion of new items or users creates a cold start problem respective of the type of collaborative filtering. Furthermore, user-to-user filtering has a higher personalization, suffers more from sparsity, is less scalable with larger user bases and may fluctuate with user behaviour. However, as it takes the user as the comparison base, the personalization is better.

2.4 Singular Value Decomposition

A large motivator for changes in recommendation algorithms was the Netflix Challenge. Before, as discussed in the introduction we can do recommendation by matching pairs of users and ratings to estimate ratings for new users (based on other ratings).

A possible approach to recommending movies by using the user ratings is the CINEMATCH system, which was state-of-the-art in 2006. While this approach worked well, Netflix decided to set up a challenge for scientists to improve the recommendation system. For this challenge you would predict ratings based on a training set and submit them. The improvement was measured using the RMSE.

Definition 2.2 (RMSE)**.**

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{predicted} - \text{true})^2}$$

2.5 Matrix Factorisation - UV Decomposition

One approach for recommendation is the use of matrix factorisation, where the users and ratings are combined into a utility matrix. This matrix is then split into a user matrix (with a fixed dimension) and a vector matrix (with a fixed dimension). Ratings can then be approximated by multiplying a specific user vector with the transpose of an item vector and summing the output. The dimension lengths are fixed before doing the decomposition but their semantic "meaning" and values are learned from the data.

If we encode a loss function to be a polynomial that measures the distance between the predicted values and the true values (like the MSE) we can do learn the correct values for U and V (where the MSE is minimal).

For example, we can do a simple line search, this initializes all variables randomly, chooses a random parameter (with the rest frozen) and then finds the optimal value for that parameter. It does that by calculating the derivative of that parameter and chooses the optimum.

The gradient descent algorithm is a better alternative, which takes the loss function and chooses an arbitrary point in the multi-dimensional space. Then, it finds a direction in which the loss function is decreasing the most rapidly using partial derivatives and makes a small step in that direction. It repeats this until a local minimum is reached.

Finally, an alternative is to use alternating least squares, which reduces the distance in an alternating fashion between the user and item matrices. First, freeze the user features and treat all item features as variables, solve the resulting least squares problem. Then, freeze the item features and treat the user features as variables and solve resulting least squares.

2.6 Boosting Accuracy - Blending Models

Better results were ultimately achieved by blending different algorithms together. For example, first you can train the initial values using a regression model instead of choosing random parameters.

3 Mining Data Streams

Instead of stored data that is used for off-line analysis we might sometimes need instant action on results or signals that we receive. We can use trained models (rules, decision trees, neural networks, et cetera) in real time. Specifically, we might be interested in queries over the last N records (sliding window) or a sample of everything that we have seen so far (i.e. the last N days).

3.1 Data Stream Sampling

Generally, high velocity data streams (such as global-scale events) are too large to mine, thus we need to sample events from these data streams. Naively, we can take a modulo, for example $(n mod 100) < c$ which will take c% samples from the stream. Alternatively, we could take random selections from a probability $c/100$.

However, there are problems with this as we run the risk of representing items from some groups more than others, and it might not be representative of the entire stream. We could also create groups, or a list of clients, and take a sample from them. Whenever an event comes by, we check if it is from a client that is in our hash map. This might become problematic if we have a large number of users, as the worst case hase to represent all users in the table. Moreover, the list of clients is a snapshot and might not reflect the current active users.

Alternatively, we could alter the hash function such that we don't store the value in a hashmap but rather map the unique clients to integers between 1 to 100 and use this to sample the first n integers. However, when the amount of data grows, the memory usage also grows if we keep the same sampling ratio.

3.1.1 Reservoir Sampling

Suppose the memory can store s records, initially we store all the records in the memory and the probability of an element entering is $\frac{s}{n} = 1$. When the $(i+1)$th element arrives, decide with probability $\frac{s}{i+1}$ to keep the record in

RAM. Otherwise, ignore it. If you choose to keep it, throw one of the previously stored records out, selected with equal probability, and use the free space for the new record.

Definition 3.1 (Reservoir Sampling)**.**

- Suppose we can store s records
- Store all incoming records until i is s
- For every subsequent record, decide with probability $\frac{s}{i+1}$ to keep the record.
- If the record is to be kept, replace a random record in the store.

3.2 Bloom Filters

Bloom Filters allow to classify entries as positive or negative, where we are certain that a real positives are never classified as negative, but have no problems with a small number of false positives. Generally, the idea is that we filter out the majority cases where we are sure that an (expensive) computation will be negative or not required.

First, decide on a memory space that you have and write a hash function that encodes the search input to a one-hot encoding in this memory space. Finally, set all bits to 1 for the positive values and whenever an input points to a positive bit, consider it positive.

To improve, we can use multiple hash functions simultaneously and in the case of a positive number, set all bits to one. In that case, all k positions in the memory need to be set to one to indicate a positive input.

Definition 3.2 (Bloom Filter - False Positives)**.**

$$\left(1-\frac{1}{N}\right)^k n = \left[\left(1-\frac{1}{N}\right)\right]^{\frac{kn}{N}} \approx e^{-\frac{kn}{N}}$$

$$\lim_{x\to\infty}\left(1-\frac{1}{x}\right)^x = e^{-1}$$

$$k_{opt} = \frac{N}{n}\times \ln 2$$

3.3 Probablistic Counting

Given a data stream of a size n we want to count how many distinct items we have seen. Generally, you would count n using a hash map, however if the size of n gets larger storing the hash map might not be feasible. If we accept that we have some loss of accuracy, we can use probablistic counting approaches.

3.3.1 MinTopK Estimate

Hash incoming objects into doubles in the interval $[0, 1]$ and count them, shrinking the interval if needed. Maintain only the K biggest values, say, $K = 1000$. If s is the minimum value of the tracked set, the number of distinct elements is approximately $K/(1 - s)$.

Definition 3.3 (MinTopK Estimate)**.**

- Pick a hash function h that maps m elements to a float in the interval $[0, 1]$.
- For each stream element a, calculate $h(a)$ and store the hash if it is in the top K hashes.
- The number of distinct elements is approximately $K/(1 - s)$.

3.3.2 Flajolet-Martin

The intuition is that you hash passing elements into short bitstrings, store only the length of the longest tail of zeroes, and the more distinct elements, the longer the longest tail of zeroes. To estimate the number of distinct elements, use $2^R/\Phi$:

Definition 3.4 (Flajolet-Martin)**.**

- Pick a hash function h that maps m elements to lgm bits.
- For each stream element a, let $r(a)$ be the number of trailing 0s in $h(a)$
- Record R = the maximum $r(a)$ seen
- Estimate the number of distinct elements as $\frac{1}{\Phi}2^R$ where $\Phi = 0.77351$

The probability that $h(a)$ ends in at least r zeroes is 2^{-r}. If there are m differente elements, the probability that none of them have r zeroes (so $R \geq r$) is $1 - (1 - 2^{-r})^m$. Pitfalls are that 2^R is a power of 2, meaning that there are larger jumps as we come across larger items. Bad luck can return in huge errors, but we can work around that by running parallel copies using different hash functions and average the results.

3.3.3 LogLog

An update to the Flajolet-Martin algorithm by Durand and Flajolet is the LogLog algorithm. This uses stochastic overaging and calibration, where the samples are partitioned into $n = 2^l$ groups using the first l bits of the hash function as a selector. If $n = 1024$, the relative error is around 3% to 4%

Definition 3.5 (LogLog)**.**

- Partition the samples into $n = 2^l$ groups, with the first l bits of the hash as selector
- Calculate $R_1, \ldots, R_n$
- Return $a_n * n * 2^{\text{mean}(R_1,\ldots,R_n)}$

This results in bit strings of length $logn$ and we maintain the length of the longest tail of zeroes: $loglogn$.

4 Similarity Search

In the task of determining the similarity between users U_1 and U_2 based on the sets of movies they have watched, a naive approach involves comparing all possible user pairs. This results in a computational complexity of $O(n^2)$, where n is the number of users. To address this challenge efficiently, we focus on identifying pairs of users with high similarity.

4.1 Jaccard Similarity

One commonly used metric for measuring the similarity between two sets is the Jaccard Similarity. This metric quantifies the overlap between two sets by comparing the size of their intersection to the size of their union.

Definition 4.1 (Jaccard Similarity)**.**

$$\text{Similarity}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

The assumption here is that we don't have to bother about all the movies that we didn't watch together, it only takes into account the movies that we did see.

4.2 Candidate Selection for Similar Pairs

In this case we would still have to execute this similarity calculation on all possible user pairs, even those that have a very small overlap. To select a set of users that we will run similarity tests on, we can use min-hashing to generate signatures and use locality-sensitive hashing to generate candidate pairs to test for similarity.

First, we run a min-hashing algorithm to generate signatures, which are short integer vectors that represent the sets and reflects their similarity. Then, we run locality-sensitive hashing to get pairs of signatures that we need to test for similarity.

4.2.1 Shingles

If we don't look at user-movie selection, but rather document-text selection to do for example plagiarism checks, we can use shingles (or grams) as a measure of similarity. For example, if we have a document $abcab$ and we take $2-grams$ we get the set of ab, bc, ca. Alternatively, instead of characters, we can use a sequence of k consecutive words. For example, $childinhiseyes$ will result in $childin, inhis, hiseyes$. Finally, we represent the document by a binary column of the set of k-shingles. Documents with a lot of shingles in common have similar text, even if the text appears in a different order. However, you need to pick k large enough, or most document will have most shingles. To compress long shingles, we can hash them to uints as we can easily compare and store such values.

4.2.2 Minhashing

We can take the column of the user-movie, or document-shingles binary matrix and replace the binary columns with a short signature. The goal is that if the columns are similar, the signatures should be similar. We can't sample from the rows and let the signature be those bits only, as the input matrix is likely to be very sparse. Rather, we use the notion that the ordering of the matrix is not relevant to the similarity calculation.

Definition 4.2 (Minhashing)**.**

- Permute (swap) the rows randomly (or generate a random sequence of row indices)
- Define hash function $h(C)$ as the position of the first permuted row in C that has a 1.
- Run this several (i.e. 100) times independently and concatenate to create a signature
- The similarity of two signatures is then the fraction of signature elements that match

We see that the probability that $h(C_1) = h(C_2)$ is very close to $\text{Sim}(C_1, C_2)$. Consider that if we check two hashes and they are not the same, we note that down as a type S row. If they are equal we note that down as a type B row. The overall probability is then $\frac{B}{B+S}$ or, the equal over the total rows, which is the same as the Jaccard Similarity (intersection over union).

The main challenge of minhashing arises when dealing with a very large number of entries, as generating and storing a true random permutation of rows becomes computationally expensive and memory-intensive. Furthermore, accessing rows in the permuted order can lead to inefficient memory access patterns, such as thrashing or excessive swapping. To address this, instead of explicitly permuting rows, we use $n \to n$ hash functions to simulate the effect of permutations. By processing the matrix top-down row by row, we update the signature values for each column and each hash function in parallel, keeping track of the minimum hash values. This approach avoids the need for explicit permutations.

Definition 4.3 (Minhashing without Permutation)**.**

- Define a family of $n \to n$ hash functions $h_1, h_2, \ldots, h_k$ to simulate the effect of row permutations.
- Initialize the signature matrix Signature$[i][C]$ with ∞ for all hash functions i and columns C.
- Process the binary matrix row by row:
  - For each row r:
    - Compute $h_i(r)$ for all hash functions $h_1, h_2, \ldots, h_k$.
    - For each column C in that row where the matrix has a 1:
      - For each hash function h_i, update the signature value:

$$\text{Signature}[i][C] = \min(\text{Signature}[i][C], h_i(r))$$

- The resulting signature matrix stores the smallest hash values observed for each column and each hash function, effectively simulating the effect of random row permutations.

4.2.3 Locality-Sensitive Hashing

Now that we have converted the large and sparse binary matrix to a smaller matrix of similarity hashes, the memory constraint is removed. However, calculating the real similarities between the pairs is still an expensive task. The idea behind locality-sensitive hashing is to split the columns of the signature matrix into bands of the same size. If two columns are very similar, then it is likely that at least one band will be identical.

Instead of first calculating the bands, we can determine candidate pairs from columns that hash at least once to the same bucket. Split the rows up into bands and calculate a number of hashes for all columns in the bands. When two columns hit the same bucket, consider them similar and record the pair.

Definition 4.4 (Locality-Sensitive Hashing)**.**

- Split the signature matrix in b bands
- For over all the columns c in all bands calculate hashes that map to k buckets.
- Whenever a column c in a band maps to the same bucket as another column, consider them similar and record the pair.

Summarizing, the probability that the signatures agree on one row is s (the Jaccard similarity) and $P_{\text{band}} = s^k$ that they agree on one particular band (or $1 - s^k$ that they are not identical). The probability that they are not identical in any of the b bands is $(1 - P_{\text{band}})^b$. Finally, the probability of becoming a candidate pair is $t \approx \frac{1}{b}^{\frac{1}{r}} = \sqrt[r]{\frac{1}{b}}$

5 Data Visualisation and Dimensionality Reduction

Visualization of distributions (one-dimensional) is be generally done with histograms.

Definition 5.1 (Histograms)**.** Estimate density by defining cut points and count occurrences between cut points (bins). Use equal-width bins or equal-height bins. Using an incorrect amount of equal-width bins can create artefacts (gaps or sharp peaks).

A problem with histograms, and the driving idea with kernel density estimation is that there is an error involved in the answers, i.e. the age is not a precise integer because someone might have their birthday next week.

Moreover, as the KDE gives a single continuous line, it is easy to draw multiple lines from the KDE. In this case, you could use it for showing differences of distribution between different subgroups.

Definition 5.2 (Kernel Density Estimation)**.** Draw a equal-width histogram, smooth the peaks (i.e. gaussian) and sum each distribution.

Before we smooth out the kernels that we calculated, we need to estimate the correct bandwidth h where σ is the standard deviation.

Definition 5.3 (Scott's Rule)**.**

$$h = \sigma \times n^{\frac{1}{5}}$$

Definition 5.4 (Silverman's Rule)**.**

$$h = \left(\frac{3n}{4}\right)^{\frac{1}{5}}$$

Finally, we could also change the h ourselves dynamically to see which value represents the data properly. A problem with KDE is that it smooths out over the minimum and maximum as well, so it shows that that might be out of our domain.

5.1 Dimensionality Reduction

Histograms and KDEs work for data with a single attribute, however, if we have multi-dimensional data that we want to visualize, we can use different techniques.

5.1.1 Principal Component Analysis

PCA reduces the dimensionality of multi-dimensional data to two-dimensional data in a linear way that allows us to plot data. The idea is that it finds the most interesting vectors in the data and plots those, with the other vector being orthogonal. After finding an orthogonal combination of vectors, it plots those. In essence, it rotates the data such that you look at one slice or side of the data.

The principal components are orthogonal and are ordered by the largest variance. The dimensionality is then reduced by selecting the first k (2 for 2D) components. This is unsupervised, and only works based on variance.

A challenge with PCA is that some attributes might be on a different scale (i.e. one in kilometers, one in millimeters). In that case, the variance of larger-scaled featuers will be more principal. In that case, it helps to first scale the dimensions.

Definition 5.5 (Principal Component Analysis)**.**

- Standardize the data
- Compute the covariance matrix C ($C = \frac{1}{n-1}X^TX$)
- Decompose C into the eigenvectors and eigenvalues $C = V\Lambda V^T$
- Sort the eigenvalues and select the top components

Definition 5.6 (PCA Scree Plot)**.** A plot that shows the (ordered) variance for the top k components, and can be used to determine how many components need to be shown.

5.1.2 t-SNE

An alternative to Principal Component Analysis when linear models do not suffice. For example, the data might have a linear subspace or in other words a 2D-manifold inside the multi-dimensional data. The idea here is that we want to draw points that are close to eachother in a high-dimensional space as neighbours in a low-dimensional space as well.

First, compute the distances between every pair of datapoints in the high-dimensional space and the low-dimensional space. Then, compute the divergence between the two and finally move the points in the low-dimensional space to lower the divergence.

The base of this algorithm is the SNE (Stochastic Neighbor Embedding), which expresses the similarity between two points as probabilities. This is done by calculating a scaled gaussian and finally making it symmetric by using both pairs and taking the average. This captures the notion of a neighbourhood, nearby points get a high probability and far away points deminish to zero.

Definition 5.7 (Stochastic Neighbor Embedding)**.**

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}$$
$$p_{i|i} = 0$$
$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

Instead of using the gaussian distribution the t-SNE uses the Student's t-distribution, as it has a thicker tail.

Definition 5.8 (t Stochastic Neighbor Embedding)**.**

$$p_{j|i} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_k \sum_{l\neq k}\left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

To compare two probability distributions, we can use Kullback-Leibler divergences between the two conditional probabilities as a cost function. P is fixed in this case, and Q is variable and represents the low-dimensional space.

Definition 5.9 (Kullback-Leibler)**.**

$$KL(P\|Q) = \sum_{i\neq j} p_{ij}\log\frac{p_{ij}}{q_{ij}}$$

Finally, there is a parameter that can be tuned named perplexity which affects the number of effective neighbours that needs to be extracted. A smaller perplexity will try to find local clusters and might miss larger global structures and larger perplexities will consider more points as neighbours, and might smooth out finer details.

Note. *Although perlexity, gradient descent and covariances are used here, don't mistake that t-SNE trains a model. It runs on one dataset and gives the outputs only for that dataset. It is not a trained model of sorts.*

6 Subgroup Discovery

Compared to a global model, we are sometimes interested in training local patterns, which are models that cover only part of the data.

For example, a prediction rule where we have a model that only holds for the left-hand side could be:

$$\text{status} = \text{married} \cap \text{age} >= 29 \rightarrow \text{salary} \geq 50000$$

The complement can be the same or different. The main idea is that we can do exploratory data analysis and find dependencies in attributes and create a global model that has little overlap and resolves conflicts.

Definition 6.1 (Subgroup)**.** Subset of the data with unusual charactersitics (antecedent of a rule).

Subgroup Discovery is a supervised paradigm, where we can do classification where the target is binary or nominal or regression where the target is numeric. Alternatively, there's EMM, where we have different kinds of targets.

For example, if we have a binary target such as whether something is in a subgroup, we can draw a confusion matrix.

Definition 6.2 (Confusion Matrix)**.**

	T	**F**	
T	TP	FP	$TP+FP$
F	FN	TN	$FN+TN$
	$TP+FN$	$FP+TN$	$Total$

Significance of Diagonal Values: High values on the True Positive and True Negative means there is a positive correlation. High values on the False Positive and False Negative means there is a negative correlation.

6.1 Quality Measures

A quality measure for subgroups summarizes the interestingness of the matrix into a single number. While the quality measures increase the interestingness of the subgroup, the coverage of the subgroup might be worse.

6.1.1 Receiver Operating Characteristics (ROC)

Each subgroup forms a point in the ROC space, in terms of its false positive rate and true positive rate.

Definition 6.3 (True/False Postive Rate)**.**

$$TPR = \frac{TP}{TP+FP}$$
$$FPR = \frac{FP}{FP+TN}$$

Ideally, you have both these ratios very high or low, if we graph them in a space then the diagonal is as good as random and the left-top and bottom-right corner are perfect subgroups. the left-bottom and top-right corner are either an empty subgroup or the entire database.

Refining the subgroup inside the ROC-space means that you can only go down or to the left as it gets smaller.

6.1.2 WRAcc

A simple quality measure is the weighted relative accuracy.

Definition 6.4 (Weighted Relative Accuracy (WRAcc))**.**

$$\text{WRAcc}(A,B) = p(AB) - p(A)p(B)$$

Balance between coverage, from the confusion matrix above this would be $TT-(TT+TF)(TT+FT)$ and is interesting around 0, and a range between $-.25$ to .25

The WRAcc increases towards the top-left ROC space and decreases towards the bottom-right ROC space. It is additive for non-overlapping subgroups, and its isometrics are linear and parallel to the main diagonal.

Definition 6.5 (Cortana Quality)**.** An extension to the WRAcc is the Cortana Quality, which ensures that the WRAcc scores are between -1 and 1. $\varphi_{cq}(S) = \alpha \times \varphi_w(S)$ and α is constant per dataset.

6.2 Numeric Subgroup Discovery

Instead of a binary target, we might have a numeric target. The idea is that we find subgroups with a significantly higher or lower average value. The trade-off is the size of the subgroup and the average target value.

Some quality measures are:

Definition 6.6 (Average)**.**

$$\varphi_{avg}(s) = \frac{\sum_{i=1}^{n} t_i}{n}$$

Note that the average doesn't take the size into account of the subgroup, the following measures do:

Definition 6.7 (Mean Test)**.**

$$\varphi_{mt}(s) = \sqrt{n}(\mu - \mu_0)$$

Definition 6.8 (z-score)**.**

$$\varphi_z(s) = \frac{\mu - \mu_0}{\sigma_0/\sqrt{n}} = \frac{\sqrt{n}(\mu - \mu_0)}{\sigma_0}$$

Definition 6.9 (t-Statistic)**.**

$$\varphi_t(s) = \frac{\mu - \mu_0}{\sigma/\sqrt{n}}$$

6.3 Exceptional Model Mining

SubDisc (formerly Cortana) does classical subgroup discovery as in nominal or numeric targets. But it also does Exceptional Model Mining, where there are multiple targets such as regression, correlation and multi-label classification.

For example, we have a distribution that clearly contains a linear model but also a lot of background noise. We want to extract the data points that regress well and do another model on the other data.

6.3.1 Quality Measures

Correlation Model

- **Correlation coefficient**:

$$\varphi_\rho = \rho(S)$$

- **Absolute difference in correlation**:

$$\varphi_{abs} = |\rho(S) - \rho(S_0)|$$

- **Entropy weighted absolute difference**:

$$\varphi_{ent} = H(p) \cdot |\rho(S) - \rho(S_0)|$$

- **Statistical significance of correlation difference**:
  - Compute z-score from ρ through Fisher transformation.
  - Compute p-value from z-score.

Regression Model

- Compare slope b of

$$y_i = a + b \cdot x_i \quad \text{and} \quad y_i = a_0 + b_0 \cdot x_i$$

- Compute significance of slope difference:

$$\varphi_{ssd} = \frac{b - b_0}{\text{Standard Error of } b}$$

7 Random Forests and Ensembles

Random Forests are supervised learning models that can be used for both classification and regression tasks. They are based on an ensemble of decision trees, which individually split the dataset into subsets by evaluating feature values. These splits occur at multiple levels, enabling the model to progressively refine its predictions for the target variable.

7.1 Ensembles

An ensemble in supervised learning is a collection of models that works by aggregating the individual predictions. Generally, it is more accurate than the base model. Regression generally averages the indivdual predictions, and classification uses a majority vote.

It helps if there is more diversity between models, which can be achieved by using randomization or multiple types of classifier models.

7.1.1 Bagging

Bootstrap Aggregating, in short Bagging, is an early implementation of this idea. Here each tree is bootstrapped with random samples with replacement from the original dataset.

Definition 7.1 (Bagging)**.**

- Take random samples with replacement.
- Given a training set D of size n, generate m new training sets $D_1, \ldots, D_m$ each of size n.
- Replacement means that some observations will be repeated in each sample.
- For a large n, each D_i will contain approximately $(1 - \frac{1}{e}) \approx 63.2\%$ unique samples and 36.8% duplicates.

Overfitting is avoided in this case as the learners have little correlation, given that they learn from different datasets. The optimal number of learners can be determined by cross-validation or Out-of-Bag (OOB) estimation.

7.1.2 Random Subspace Method

A follow-up of bagging is the random subspace method. Instead of sampling, we build each tree in the ensemble from a random subset of the attributes. This method is particularly effective for high-dimensional problems as individual trees are prevented from over-focusing on attributes that appear most predictive in the training set.

7.1.3 Random Forests

Random forests combine the ideas of bagging and the random subspace method. At each split in a tree, a random subset of the attributes is selected, and the best split is chosen from this subset. The number of attributes selected is typically $\sqrt{p}$ for classification and $\frac{p}{3}$ for regression, where p is the total number of attributes. Random forests also employ out-of-bag (OOB) estimation for model evaluation and tuning.

Definition 7.2 (Random Forests)**.** Random forests are an ensemble learning method that constructs multiple decision trees during training. Predictions are made by aggregating the outputs of individual trees (e.g., by majority voting for classification or averaging for regression). The randomness introduced in both the sampling of data and the selection of attributes ensures reduced overfitting and increased generalization.

Definition 7.3 (Out-of-Bag Estimation)**.** Out-of-bag (OOB) estimation is a technique for evaluating the performance of ensemble models like random forests without the need for a separate validation set. During training, each tree is constructed using a bootstrap sample of the data, leaving approximately 36.8% of the samples out of the bootstrap sample. These out-of-bag samples are used to estimate the model's performance by testing them on the trees that did not use them during training.

The problem with Random Forests opposed to simple decision trees is that they are not very transparent due to the splitting of the model into multiple trees. To figure out the important attributes you can record the average OOB-error over all trees: e_0, then over each independent attribute j:

- Shuffle the values of the attribute j, such that it only gives noise
- Refit the Random Forest
- Record the average OOB error e_j
- Importance of j is the difference $e_j - e_0$

8 Boosting

Given a collection of learners in the decision tree, consider a class of weak learners h_m which is only slightly better than random guessing. These are generally leafs of the decision tree and are also called "decision stumps".

Build a sequence of classifiers $h_1, h_2, ..., h_k$:

- h_1 is trained on the original data
- h_2 assigns more weight to misclassified cases by h_1
- h_i assigns more weight to misclassified cases by $h_1, ..., h_{i-1}$

The final classifier is then an ensemble of $h1, ..., h_k$ with weights α_m per classifier h_m: $G(x) = \text{sign}(\sum a_m h_m(x))$.

Definition 8.1 (Learner Weights)**.** The weight α_m for a learner is determined by its performance, quantified by the error rate ϵ_m. Higher-performing learners receive greater positive weights, while poor-performing learners may receive negative weights. The weight is computed as:

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right),$$

where ϵ_m is the error rate of learner m.

Definition 8.2 (Data Weights)**.** On each iteration G_m to G_{m+1}, we increase the weights for G_{m+1} based on the G_m's misclassified cases. We increase the weight by a factor $\exp(\alpha_m)$.

8.1 AdaBoost

AdaBoost is an implementation of this idea, and works as follows:

Definition 8.3 (AdaBoost)**.**

- Initialize the observation weights $w_i = \frac{1}{N}$
- Loop over the classifiers M as m
  - Fit a classifier $h_m(x)$ to the training data using weights w_i
  - Compute the weighted error $\epsilon_m = \frac{\sum w_i \times I(y_i \neq h_m(x_i))}{\sum w_i}$
  - Compute $\alpha_m = \frac{1}{2} \ln(\frac{1-\epsilon_m}{\epsilon_m})$
  - For $i = 1$ to N
    - If x_i is misclassified, set data weight w_i to $w_i \times \exp(\alpha_m)$
- Output $G(x) = \text{sign}(\sum \alpha_m h_m(x))$

8.2 Gradient Boosting

An alternative to AdaBoost is Gradient Boosting that has a bit more theory behind the way how each iteration works. It uses a cost function $L(u, F(x))$ which is the MSE in regression and log loss in the classification. This loss function is differentiable and we can use gradient descent here.

Definition 8.4 (Gradient Boosting)**.**

- Initialize the model with a constant value F_0, e.g. the mean of target values.
- For $m = 1$ to M
  - Train weak learner $h_m(x)$ to minimize residuals from the current prediction using the loss function (c.q. train the model to the negative gradients).
  - New prediction is $F_{m+1}(x) = F_m(x) + v \times h_m(x)$, where v is the learning rate (e.g. 0.1)
- Output $F_M(x)$ as the final model

8.2.1 Loss Functions

By default, we look at the MSE, but there are other loss functions to use for the gradient descent as well, with the square loss repeated:

Definition 8.5 (Square Loss)**.**

$$L(y, F) = \frac{1}{2}(y - F)^2$$

Definition 8.6 (Absolute Loss)**.**

$$L(y, F) = |y - F|$$

Definition 8.7 (Huber Loss)**.**

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & \text{if } |y - F| \leq \delta \\ \delta\left(|y - F| - \frac{1}{2}\delta\right) & \text{if } |y - F| > \delta \end{cases}$$

Where δ refers to a parameter that decreases how extreme the output should be with respect to higher outliers.

Avoiding overfitting can be done by lowering the learning rate (in other words, shrinkage). Or, stochastic gradient boosting, where each base learner is trained from a random sample of the data with fraction f.

8.2.2 XGBoost

A modern and highly succesful version of Gradient Boosting, that did some improvements such as handling of missing data, L1 and L2 regularization that adds a penalty to loss functions and it uses second-order derivatives for more accurate optimization.