CS5228 – Tutorial 1

Data, Data Cleaning, Data Preprocessing

Figure 1 shows the first 20 data samples from the Students Performance in Exams dataset freely available on Kaggle. However, for the purpose of this tutorial, we tweaked it a little bit. After all, we don't really perform any analysis on that data here. Although only 20 samples are shown, just by looking at them, we can already gain some very basic insights that can and/or should guide any subsequent processing steps. As additional information, there are a total of 1,000 samples in the dataset.

id	gender	ethnicity	state	parent_education	email	sleeping_hours	prep_course	math_score	read_score	write_score
1	female	group b	MO	Bachelor's Degree	xxxxx	7 to 8	none	72.0	72.0	0.74
2	female	group c	VA	Some College	xxxxx	< 6	completed	69.0	90.0	0.88
3	female	group b	PR	master's degree	xxxxx	7 to 8	none	90.0	95.0	0.93
4	male	group a	CT	associate's degree	xxxxx	7 to 8	none	47.0	57.0	0.44
5	male	group-c	UM	some college	xxxxx	8 to 9	none	76.0	78.0	0.75
6	female	group b	OK	Associate's Degree	xxxxx	6 to 7	none	71.0	83.0	0.78
7	female	group-b	PA	Some College	xxxxx	6 to 7	completed	88.0	95.0	0.92
8	male	group-b	GA	Some College	xxxxx	> 9	none	40.0	43.0	0.39
9	male	group d	LA	high school	xxxxx	6 to 7	completed	64.0	64.0	0.67
10	female	group b	KY	High School	xxxxx	7 to 8	none	38.0	60.0	0.50
11	male	group c	AZ	Associate's Degree	xxxxx	< 6	none	58.0	54.0	0.52
12	male	group d	OK	associate's degree	xxxxx	8 to 9	none	40.0	52.0	0.43
13	female	group-b	OR	High School	xxxxx	7 to 8	none	65.0	81.0	0.73
14	male	group-a	SD	some college	xxxxx	6 to 7	completed	78.0	72.0	0.70
15	female	group a	KY	master's degree	xxxxx	8 to 9	none	50.0	53.0	0.58
16	female	group-c	CO	Some High School	xxxxx	< 6	none	69.0	75.0	0.78
17	male	group-c	UM	high school	xxxxx	7 to 8	none	88.0	89.0	0.86
18	female	group b	KY	Some High School	xxxxx	> 9	none	18.0	32.0	0.28
19	male	group-c	WI	Master's Degree	xxxxx	6 to 7	completed	46.0	42.0	0.46
20	female	group-c	NE	associate's degree	xxxxx	6 to 7	none	54.0	58.0	0.61

Figure 1: 20 Samples of Students Performance in Exams (modified).

Data preparation. The following questions are not "exam questions" as there can be quite some room for discussion with no single correct answer. This is particularly true since (a) we only see a small snippet of the whole dataset, (b) we have no means to perform an EDA beyond eye-balling the data, and (c) we did not specify the exact data mining task we want to solve.

1. **Types of Attributes.** For each attribute, decide whether it is *nominal*, *ordinal*, *interval*, or *ratio*. For which attributes might this decision not be so clear?

Solution:

- id: nominal
- gender: nominal
- ethnicity: nominal
- state: nominal
- parent_education: ordinal
- email: nominal
- sleeping_hours: ordinal/ratio
- prep_course: nominal
- *_score: ratio

As strings, the values for sleeping_hours can be considered ordinal, but strictly speaking, hours is a numerical measure. So sleeping_hours can be processed to result in numerical values, making this a ratio attribute. Also, whether an ordinal attribute such as parent_education might better be considered only a nominal attribute may depend on the exact task or other constraints.

2. **Data Cleaning.** Just by looking at these 20 samples, which data cleaning steps seem recommended? Note that this refers only to preprocessing steps to remove potential noise from the dataset, not any steps that might further benefit a subsequent analysis (arguably, there is no clear distinction, but we cover these steps in the next questions).

Solution:

- Normalize ethnicity (e.g. "group c" vs "group-c")
- Normalize parent_education (e.g., convert to all lowercase)
- Adjust scale of write_score

3. **Attribute/Feature Importance.** For each attribute, assess its importance (or relevance, usefulness, etc.) for a subsequent analysis. Let's assume we want to predict students' math, reading, and writing scores based on the other attributes.

Solution:

- remove id (just an "artificial" attribute)
- remove email (it's redacted anyway, and so of no use)
- probably remove state (we only have 1,000 sample and there a 50+ states, so the information w.r.t. students' state is very sparse and far from representative)

4. **Additional Data Preprocessing.** Many to most off-the-shelf data mining algorithms for clustering or classification/regression require numerical data as input. How does this affect the analysis of this dataset, and what can we do to address this using data preprocessing?

   (For this question, you are encouraged to look up alternative encoding strategies for converting categorical attributes into numerical ones. In the lecture, we only covered one-hot encoding for nominal attributes. However, there are other strategies for nominal and ordinal data, which can be useful for the project. Also, check and appreciate the pros and cons of different strategies.)

Solution:

- gender and prep_course seem to be both binary attributes, so 0/1 encoding should do just fine
- parent_education can be converted into a simple ranking of numerical values (e.g., 1, 2, ...). Many tools support **ordinal encoding** but users have to ensure that the labels reflect to semantic order of the original values (otherwise Master's Degree might be 0, and High School might be 1)
- state is definitely nominal, so we need to encode it. **One-hot encoding** is always applicable but that would result in 50+ new attributes that would also be very sparse. For classification/regression task, we could go with **target encoding** where we replace each state value with the mean of the output values over all sample with that state value (e.g., we replace "MO" with 70.3, assuming that's the mean of all math course of tuples with the state "MO"). But again, the data for state is very sparse, so it's really better to just ignore this attribute. Alternatively, maybe a generalization approach might be interesting (e.g., mapping all states to Democratic and Republican, yielding a convenient binary attribute). However, target encoding is not uncontroversial as it allows for *"data leakage"*. Simply speaking, we create

a feature based on the target, which is a kind of cheating. We look into the issue of data leakage a bit more when we cover the topics classification and regression.

- converting the sleeping_hours strings into a numerical estimate would probably be useful.

Optional "homework". In the lecture, we focused on the common challenges when working with real-world data, e.g.: noise, outliers, inconsistencies, missing values, misleading default values, imbalanced class labels, etc. However, there is also the potential case where the data was intentionally manipulated or tampered with. The purpose of such tampering is typically (a) to "inject" a pattern into the data that was not there or (b) to remove or obfuscate a pattern in the original data. Of course, both cases are highly unethical. Have a look at the website of DataColada. This blog is run by a group of researchers to identify and find evidence for data fraud, and they uncovered some very high-profile cases.

CS5228 – Tutorial 2

Clustering: K-Means & DBSCAN

K-Means and DBSCAN are two very popular clustering methods. Since clustering is typically used to find patterns in unlabeled data, it is generally very difficult to reliably assess if a resulting clustering is "good". This makes it even more important to properly understand the underlying principles, as well as the pros and cons of the different methods to better interpret the results.

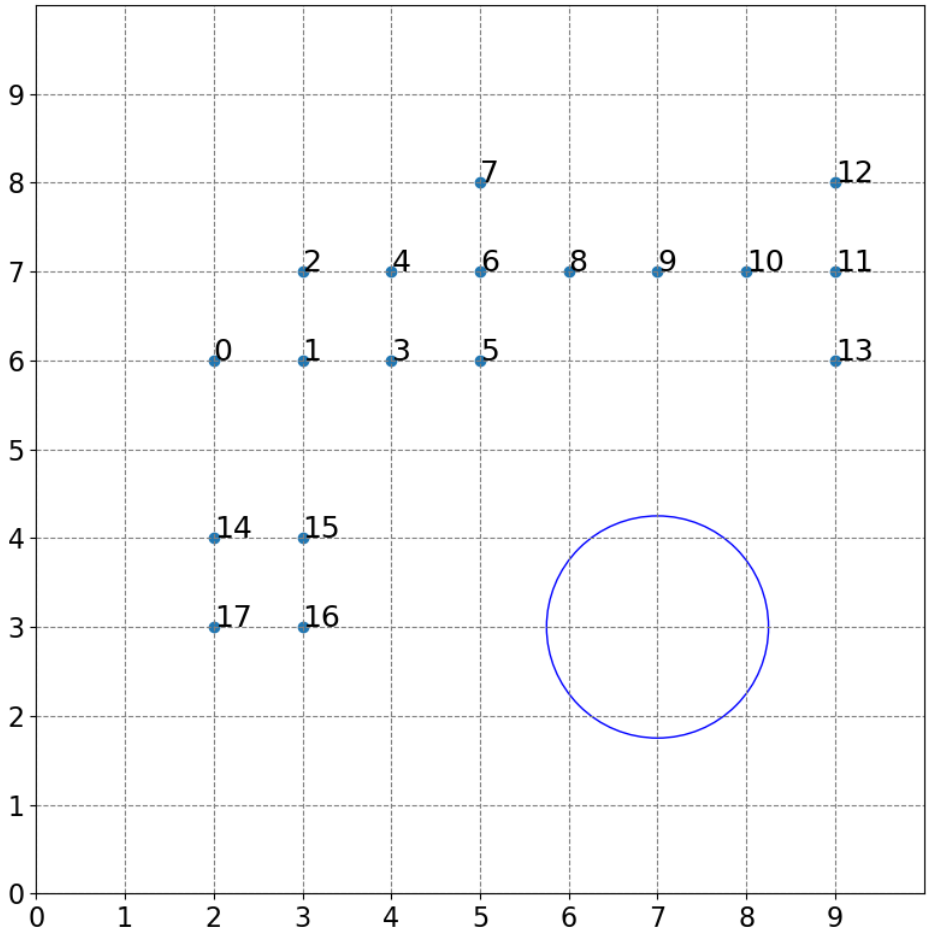


Figure 1: Toy dataset for "manually" performing DBSCAN.

1. **Performing DBSCAN "by hand".** Figure 1 shows a toy dataset with 18 data points. Let's assume we run DBSCAN over this data with $\epsilon = 1.25$ and $MinPts = 4$.

   (a) What will be the result of DBSCAN? Describe the ouput by listing all

   - core points
   - border points
   - noise points

   Solution:

   - core points = [1, 3, 4, 6, 11]
   - border points = [0, 2, 5, 7, 8, 10, 12, 13]
   - noise points = [9, 14, 15, 16, 17]

   (b) How many clusters are there, and what are their data points?

   Solution:

   - Cluster 1 = [0, 1, 2, 3, 4, 5, 6, 7, 8]
   - Cluster 2 = [10, 11, 12, 13]

   (c) Can you add 2 data points such that the resulting clustering contains only 1 cluster and no noise? If so, give the coordinates for both points!

   Solution:

   - Example solution: (2.5, 5), (7, 6.8)
   - Note that this is not a unique solution

   The simplicity of the toy dataset should allow you to answer all three tasks by just looking at the plot in Figure 1. There should be no need to actually calculate any distances. The blue circle reflects the chosen radius of $\epsilon = 1.25$ to make it easier for you.

2. **K-Means.** For the following questions, assume that we now want to run K-Means over the toy dataset shown in Figure 1.

   (a) For $K = 3$, can you find locations for the initial centroids so that the resulting clustering will contain 0, 1, 2, or 3 non-empty clusters? You can answer this question qualitatively; there is no need to list any exact coordinates for the initial centroids.

   Solution:

- There will always be at least 1 non-empty cluster; so 0 non-empty clusters can never happen.
- 1 or 2 clusters means 2 or 1 empty clusters. For example, the get 2 empty clusters, we only need to place 1 centroids "within" the dataset, and the 2 other centroids just far away.
- It is easy to see that placing the 3 centroids at the locations 4, 11, and 16 (not the only initialization) will result in 3 non-empty clusters.

(b) Now we assume that K-Means++ initialization is used. For $K = 3$, what is the minimum and maximum number of clusters? (Comment: For this question you may need to consider arbitrary datasets and not just the toy dataset!)

Solution:

- The number of clusters C is guaranteed to be in $1 \leq C \leq K$.
- After an update setup – typically already after the first one – the centroids are generally no longer at the location of data points. This means that in the next assignment there might now be a centroid which is not the closest to any other data point, thus yielding an empty cluster.

3. **K-Means vs. DBSCAN**

(a) Apart from their implementation, what is the fundamental difference between K-Means and DBSCAN?

Solution:

- K-Means is defined as an optimization problem; hence there is a notion of local and global optimal solutions. The commonly used Lloyd's algorithm is a heuristic that does no guarantee to always find the global optimum
- K-Means considers relative similarities/distances
- K-Mean favors blob-like clusters
- DBSCAN is not defined as an optimization problem, so there's no notion of a local/global optimum. The DBSCAN algorithm is not a heuristic.
- DBSCAN considers absolute similarities/distances
- DBSCAN can handle non-blob-like clusters

(b) What are meaningful criteria to decide whether K-Means or DBSCAN is the preferable clustering method for a certain task?

Solution:

- DBSCAN has the notion of noise, which can be used for outlier detection
- DBSCAN better when clusters are decidedly not blobs
- DBSCAN can work well if the parameters for ϵ and $MinPts$ can be intuitively set
- K-Means when the value for k is predefined by application context
- since K-Means considers relative similarities/distances, it's arguably easier to use for an EDA to get a meso-view of the data

(c) Come up with 5 example tasks and discuss why K-Means or DBSCAN would be your method of choice!

Solution:

1. Traffic congestion along roads based on location of cars (DBSCAN)
2. Identifying locations of low coverage, e.g., distribution of Starbucks outlets across a city (DBSCAN)
3. Credit card fraud or intrusion detection to find outliers (DBSCAN)
4. Organizing conference papers into a given set of research areas (K-Means)
5. Clustering people based on biological data (e.g., height, weight). Such data is typically always normally distributed so clusters are more likely to be blobs (K-Means).

(d) Is there any example where fundamentally only K-Means is applicable but not DBSCAN, or vice versa?

Solution:

- No, both methods only require a well-defined similarity/distance measure to be applicable.

(e) Is it possible that both K-Means and DBSCAN return the same clustering for a dataset or will the clusterings always be different?

Solution:

- It is easy to create or image a dataset where both algorithms – assuming the appropriate parameter values – will return the same clustering.
- Straightforward example: a dataset with very well-separated (i.e., very obvious) clusters. With the right parameters, both K-Means and DBSCAN will return the same set of those well-separated clusters.

CS5228 – Tutorial 3

Clustering: AGNES & Cluster Evaluation

Compared to K-Means and DBSCAN, Agglomerative Nesting (AGNES) is a hierarchical clustering method. As such, each data point may belong to different clusters depending on the hierarchy level (as AGNES yields complete clusterings, each point belongs to at least one cluster). Regarding the underlying algorithm, AGNES not only relies on the notion of distance between data points (or centroids), but also on the distance between clusters (i.e., sets of data points). This led us to the concept of *Linkage Methods* – that is, different approaches to calculate the distance between two clusters.

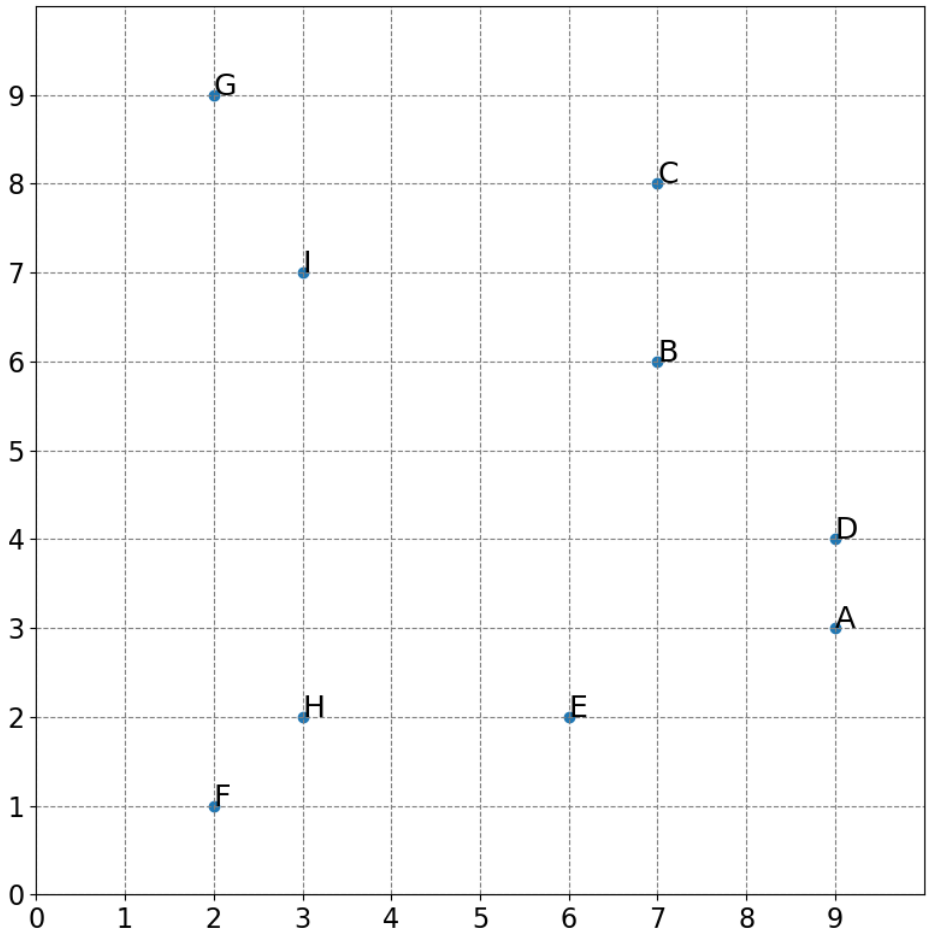


Figure 1: Toy dataset for "manually" performing AGNES.

1. **Performing AGNES "by hand".** Figure 1 shows a toy dataset with 9 data points. In the following 2 tasks, perform Hierarchical Clustering (AGNES) step by step on the dataset above. After each step write down the current set of clusters and the value of the shortest distance!

   - Denote a cluster as a sequence of points forming that cluster. For example, XYZ denotes the cluster containing the data points labeled X, Y, and Z – the order is not important
   - Write down the shortest distance between the two clusters merged in each step. Note that the points are conveniently placed to make the calculation of distances pretty straightforward. If needed, round the distances to 2 decimal places (e.g., sqrt(2) = 1.41)
   - The table below shows an example of clustering 3 data points X, Y, and Z.

Start	X, Y, Z	At the start, each data point forms a cluster
1.22	Y, XZ	Cluster X and Z where closest with a distance of 1.22
2.00	XYZ	Cluster XZ and Y where closest with a distance of 2.0
End	XYZ	At the end, all data points are within a single cluster

   (a) **Perform AGNES step by step using Single Linkage!** Write down the process of forming the clusters as indicated in the example table above; you can omit the third column with the comments.

   Solution:

Start	A, B, C, D, E, F, G, H, I
1.00	AD, B, C, E, F ,G, H, I
1.41	AD, FH, B, C, E, G, I
2.00	AD, FH, BC, E, G, I
2.24	AD, FH, BC, GI, E
2.83	ABCD, FH, GI, E
3.00	ABCD, EFH, GI
3.16	ABCDEFH, GI
4.12	ABCDEFGHI
End	ABCDEFGHI

   (b) **Perform AGNES step by step using Complete Linkage!** Write down the process of forming the clusters as indicated in the example table above; you can omit the third column with the comments.

   Solution:

Start	A, B, C, D, E, F, G, H, I
1.00	AD, B, C, E, F ,G, H, I
1.41	AD, FH, B, C, E, G, I
2.00	AD, FH, BC, E, G, I
2.24	AD, FH, BC, GI, E
3.61	ADE, FH, BC, GI
5.83	ADE, BCGI, FH
7.62	ADEFH, BCGI
9.22	ABCDEFGHI
End	ABCDEFGHI

(c) **Compare and discuss the results!** Even for this small toy dataset the resulting clustering will differ for Single Linkage and Complete Linkage. Briefly describe qualitatively, how does the choice of the linkage method affect the result! What does that mean for the potential shape of clusters on real-world data? (Hint: Check in what step the results from (a) and (b) start to differ).

Solution:

- Until the 4th step resulting in (AD, FH, BC, GI, E), both Single Linkage and Complete Linkage perform the same merges; this seems intuitive as these 4 pairs of data points kind of form well-separated clusters (also: in the beginning, when most clusters still just contain a single data point, the linkage methods matters much less or not at all)
- In step 5, Single Linkage merges Clusters BC and AD since it only looks at the two closest points B and D, respectively; this would also be true if, say, Cluster would already contain more points "above" B and C as they would matter in case of Single Linkage
- In contrast, w.r.t. Complete Language, Clusters BC and AD are relatively far apart since here the two far-away points C and A matter.
- Generally speaking, Complete Linkage is more likely to yield blob-like clusters, while Single Linkage potentially may yield clusters of arbitrary shape.

(d) **Analyze the performance!** In the lecture, we briefly mentioned that Single Linkage can be implemented more efficiently. Briefly describe qualitatively why this is the case. Having performed AGNES "by hand" in (a) and (b) should help with that.

Solution:

- Once we have the initial distance matrix with all pairwise distances between data points, we have already all the information to perform Single Linkage

- For Single Linkage the distance between two clusters will always be the distance between some data points (which we already calculated); this is not for the other Single Linkages.
- To convince yourself, you can create the initial 9x9 distance matrix M containing the pairwise distances between data points. You will see that all distances in the result table of (a) will be somewhere in M; the same will not hold for the result table of (b).

2. **Evaluation of Clusterings** We have seen in the lecture that – apart from any ground truth in the form of labeled data – there is no perfect method to reliably evaluate the quality of a clustering. While different methods exist, they all have their limitations and it's important to be aware of those.

   (a) What are limitations of using SSE as a general metric to measure the quality of a clustering?

   Solution:
   - SSE favors blob-like clusters
   - SSE is always decreasing
   - SSE does not punish large number of clusters
   - The elbow method not so straightforward to apply

   (b) What does the Silhouette Score (SC) do better than SSE but which limitation still remains?

   Solution:
   - SC is not monotonically decreasing and punishes large number of clusters
   - SC still favors blob-like clusters

   (c) If clusterings are so difficult and unreliable to evaluate, why can we still say that clustering is such a very useful data mining method?

   Solution:
   - General-purpose data mining method ("only" distance/similarity measure required, works on unlabeled data).
   - Clustering methods are arguably intuitive, making the results (relatively) easy to interpret.

- Calculating, inspecting, evaluating clusters can be part of the EDA and data preprocessing (cf., binning & smoothing).
- Evaluation of clusterings in practice is often very pragmatic (e.g., parameter values given by the task; only individual clusters might be important, and not the complete clustering).
- Clustering provides a useful (and straightforward) meso-view on the data.

CS5228 – Tutorial 4

Association Rule Mining

Association rule mining is a method with the goal to identify frequently occurring patterns, correlations, or associations from transactional or similar datasets. The most popular application context is in Market Basket Analysis where Association Rule Mining is used to identify supermarket products that are frequently bought together (but we also saw another example in the lecture)

1. **Basic definitions.** Association Rule Mining aims to find "interesting" rules, interesting co-occurrences of items. To quantify "interesting", a series of different metrics exist. In the lecture, we mainly covered *support*, *confidence*, and *lift*. For the following tasks, let's assume the following basic dataset comprising 8 transactions.

tid	transactions
1	B, C, D, E, F
2	E, C
3	D, B, D, A
4	G, E, H, C
5	H, A, G, D, B
6	B, E, G
7	B, A, D
8	A, D, B, C

   (a) **Calculate the following values:**

   - $support(\{A\})$, $support(\{B\})$ $support(\{A, B\})$
   - $support(\{A\} \rightarrow \{B\})$, $support(\{B\} \rightarrow \{A\})$
   - $confidence(\{A\} \rightarrow \{B\})$, $confidence(\{B\} \rightarrow \{A\})$
   - $lift(\{A\} \rightarrow \{B\})$, $lift(\{B\} \rightarrow \{A\})$

   Solution:

   - $support(\{A\}) = 1/2$, $support(\{B\}) = 3/4$ $support(\{A, B\}) = 1/2$
   - $support(\{A\} \rightarrow \{B\}) = 1/2$, $support(\{B\} \rightarrow \{A\}) = 1/2$
   - $confidence(\{A\} \rightarrow \{B\}) = 1.0$, $confidence(\{B\} \rightarrow \{A\}) = 2/3$
   - $lift(\{A\} \rightarrow \{B\}) = 4/3$, $lift(\{B\} \rightarrow \{A\}) = 4/3$

(b) **Connections between metrics.** In a) we calculated the *support*, *confidence*, and *lift* for different itemsets and association rules. However, we do not need to calculate all values individually. Based on the definitions of the metrics, which calculations can we skip?

> **Solution:**
>
> - $support(X \to Y) = support(X \cup Y)$
> - $support(X \to Y) = support(Y \to X)$
> - $lift(X \to Y) = lift(Y \to X)$ based on its definition

(c) **"Usefulness" of different metrics.** What makes *support* and *confidence* more useful compared to other metrics such as *lift*, *conviction*, *collective strength*, *leverage*?

> **Solution:** Both *support* and *confidence* have the anti-monotone property which can be exploited for the Apriori Algorithm for finding association rules

(d) **"Importance" of different metrics.** What makes an association rule interesting? A high *support*, high *confidence*, high *lift*, high *conviction*, etc.?

> **Solution:** No metric is intrinsically the most indicative one for describing how interesting an association rule is. All metrics look at different aspects that make a rule interesting and which aspect might be most important typically depends on the exact application context.

2. **Finding the relevant rules.** In the lecture, we were mainly interested in association rules with a high support and a high confidence. This was also convenient since large(r) values for *minsup* and *minconf* typically speed up the execution of the Apriori algorithm.

   In the following, let's assume a transaction dataset for medical data analysis similar to what we saw on the lecture slides. In more details, each transaction contains as items a set of symptoms, together with one item indicating a positive or negative COVID-19 test result. For example, our dataset might look like this

tid	transactions
1	cough, fatigue, COVID-19-negative
2	anosmia, cough, fatigue, COVID-19-positive
3	anosmia, fatigue, headache, heart palpitations, COVID-19-positive
4	cough, fatigue, headache, COVID-19-negative
5	headache, stomach pain, COVID-19-negative
6	cough, heart palpitations, COVID-19-negative
7	anosmia, headache, stomach pain, COVID-19-positive
...	...

Our task is now to find rules that indicate which set of symptoms are most likely associated with a positive COVID-19 test. For example, we want to find rules like {anosmia, fatigue}→{COVID-19-positive}.

(a) **Choice of *minsup* and *minconf*.** How might the setup and the task above affect which values for *minsup* and *minconf* are meaningful? Hint: Assume that a large majority of the COVID-19 test results in our dataset are negative.

> **Solution:** If we assume that most COVID-19 test results have been negative, then any rules where {COVID-19-positive} is on the right-hand side won't be very frequent (compared to rules where {COVID-19-negative} is on the right-hand side). As such, we cannot set *minsup* too high, or we won't find any relevant rule.

(b) **Tweaking the dataset.** Can we simplify this task by only considering those transactions that contain COVID-19-positive, and remove all transactions that contain COVID-19-negative?

> **Solution:** No, we should not do this as that might yield misleading results. For example, we might get a rule with a high support, say, {cough}→{COVID-19-positive}. However, if a cough is so common even for negative results, we would overestimate its importance as an indicator for a positive test result. Also note that any rules with {COVID-19-positive} on the right-hand side would have a confidence of 1.0.

3. **Complexity Analysis.** The most naive approach for mining association rules would be to generate all possible rules and check if their support and confidence exceeds the specified thresholds *minsup* and $minconf$. In the lecture, you have learned that, given d unique items in a dataset of transactions, there are $3^d - 2^{d+1} + 1$ possible rules.

   Proof that d unique items result in $3^d - 2^{d+1} + 1$ possible rules! (Hint: Write out all possible rules for $d = 2, 3, 4, ...$ items; you should quickly spot the pattern that will allow you to validate the formula).

> **Solution:**
>
> - Each item has 3 possibilities to appear in a rule: on the left side of the rule, on the right side of the rule, or not at all. That reflects the 3^d possibilities.
> - However, these 3^d rules include invalid ones where the left and/or right side of the rule is empty. Of the 3^d rules, there are 2^d where the left side is empty, and 2^d where the right side is empty. We have to subtract these invalid combinations, and $2^d + 2^d = 2^{d+1}$.
> - Note that we now have subtracted the rule $\{\} \rightarrow \{\}$ twice. So we need '+1' to correct for this.

CS5228 – Tutorial 5

Classification & Regression I (Evaluation)

1. **Basic classification metrics.** Assume that you have trained a binary classifier that aims to predict if a bank customer will default on his/her credit. Your test data contained 4,000 samples and your final model yields the following confusion matrix:

prediction \ actual label	**1** (default)	**0** (no default)
1 (default)	**260**	**610**
0 (no default)	**10**	**3120**

(a) Calculate the *Accuracy*, *Specificity*, *Sensitivity*, *Recall*, *Precision*, and *F1 score*!

Solution:

- $Accuracy = \frac{260+3120}{260+3120+10+610} = 0.84$
- $Specificity = \frac{3120}{3120+610} = 0.84$
- $Sensitivity/Recall = \frac{260}{260+10} = 0.96$
- $Precision = \frac{260}{260+610} = 0.3$
- $F1 = 2 \cdot \frac{0.3 \cdot 0.96}{0.3+0.96} = 0.46$

(b) For each metric, provide a verbal interpretation of the resulting value in the context of predicting a customer's likelihood to default on his or her credit! Discuss if we can be happy with the result, or what result might cause problems in practice!

Solution:

- **Accuracy:** With respect to all predictions for all customers, the classifier will be correct about 84% of the time.

- **Specificity:** If a customer will *not* default, the classifier will very likely predict it correctly (with 84% probability).
- **Sensitivity/Recall:** If a customer will default, the classifier will very likely predict it correctly (with 96% probability).
- **Precision:** If the classifier predicts that a customer will default, it will be only correct 30% of the time
- **F1:** A balanced consideration of both Recall and Precision.

The rather high values for Accuracy and Specificity can be a bit misleading since the dataset is a bit imbalanced given that most customers do not default on the credit. However, the main problem is the low Precision value. It essentially means that the classifier will predict default "too often", i.e., in many cases where the customer would not default. If the bank decides to approve or deny credit based on this result, many customers will unjustifiably not qualify for a credit.

2. **Imbalanced datasets.** One reason why we have different metrics to evaluate the quality of a classifier is because of imbalanced datasets where a majority class contains most of the samples whereas a minority class contains only a fraction of samples (assuming a binary classification task).

   (a) List 5 example applications where you would expect a very imbalanced dataset.

Solution: In general, every application where we need to predict a rare class qualifies here

- Credit card fraud
- Intrusion detection
- Earthquake / tsunami / etc. warning system
- Automated missile defense system
- Suspect / criminal profiling

   (b) While not covered in the lecture, what do you think can be done to address the issue of imbalanced datasets (beyond picking the right metric)?

Solution:

- Collect more data, if possible and/or practical
- Generation of synthetic data, if possible and/or practical
- Undersampling of majority class

- Oversampling of minority class
- Data augmentation / "smart" oversampling

3. **Assessing classification errors.** In case of a binary classification, we can make 2 types of errors:

   - False Positives (FP), also called Type I Error
   - False Negatives (FN), also called Type II Error

   In the lecture, we mentioned that in many cases these two types of errors are not equally problematic.

   (a) List 2 example applications where False Positives are more problematic than False Negatives, and vice versa. Provide a brief explanation!

   Solution: False Negatives are more problematic than False Positives ⇒ aiming for a high Recall

   - Fraud detection: When trying to detect fraudulent transactions in the banking etc., missing a true instance of fraud (false negatives) can result in financial losses for both the customers and the institution. While incorrectly flagging legitimate transactions as fraudulent (false positives) could inconvenience customers and lead to frustration, it's often more critical to have a high recall to ensure that as many actual cases of fraud are caught as possible.
   - Medical Diagnosis: Missing to classify a high-risk patient as such can be lethal to this patient. In contrast, incorrectly classifying a healthy patient as a high-risk one is arguably less problematic (although it is likely to cause these healthy patients to worry). Of course, false positives still are problematic as they can involve additional expensive tests and can worry the patients.

   False Positives are more problematic than False Negatives ⇒ aiming for a high Precision

   - Spam detection: It's usually OK to occasionally let a spam email through the filter. However, filtering a non-spam email that might have been very important can have severe consequences. In other words, if we label an email as spam, we want to be very certain about it.
   - Recommender systems: We can train a classifier to predict whether a user will like a, say, movie. Since there are countless numbers to choose from, there's generally no harm to miss out on movies that we should recommend. However, recommending too many movies the users won't like will negatively affect users' perception of the quality of the recommender system.

(b) Regarding any difference between errors of Type I and Type II, how would you assess their relative importance for the following application use cases:

- Earthquake warning systems
- Automatic missile defense system

Solution: Intuitively, you don't want to miss any potential earthquake or any missile attack. So ideally the number False Negatives should be 0. However, since these are arguably rare events, aiming for $FN = 0$ is difficult to justify. The problem is that False Positives can also have severe consequences. In case of earthquakes, a false warning may lead to panic and over-reactions. So while False Negatives are arguably worse, the rarity of earthquakes and missile attacks (+ the consequences of False Positives) have to be taken into consideration.

4. **How good is "good enough"?**

(a) Say you trained a binary sentiment classifier that classifies social media posts (e.g., tweets) into "negative" and "positive". Your datasets for training and testing were balanced and sufficiently large. Let's assume that the F1 score of your classifier is 0.85. How would you assess if this is a good result?

Solution:

- A "random guesser" would get an F1 score of around 0.5, so the classifier is certainly much better than that
- Comparing a classifier typically involves a comprehensive comparison with other solutions so see how it performs against state-of-the-art models.
- While an F1 score 1.0 is the theoretical upper bound, in practice the goal is often below that. This is particularly true for such subjective use cases like sentiment analysis where different people might give different sentiments to the same tweet. For example, if people agreed on tweets' sentiments in 90% of the cases, then the classifier is rather close to this number.

(b) Say you trained a classifier that identifies whether an image contains a Car, Boat, or Plane, and the F1-score is very high, say, 0.99. Your datasets for training and testing were balanced and sufficiently large. What might be a reason why the classifier would suddenly perform poorly in practice?

Solution:

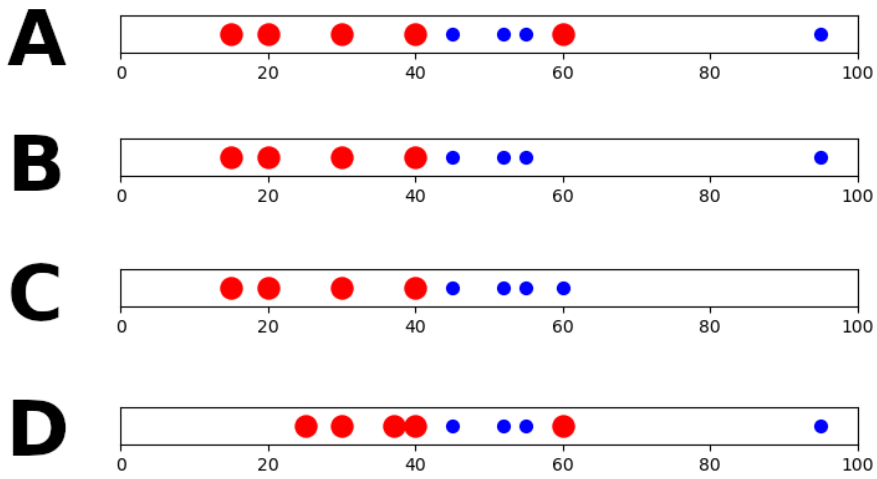
- Apart from the size of the dataset and the balance of class labels, another important requirement is that the dataset is *representative*.

- A "bad" dataset would contain only images with planes in the sky, boats on the open sea, and cars on roads. In this case, the classifier might identify planes because of the blue/grey background (sky).
- In short, it's not obvious that the classifiers learned the "correct" pattern.

CS5228 – Tutorial 6

Classification & Regression II (Tree-Based Models)

1. **Effects of data distribution on Decision Trees.** The figure below shows 4 distributions of the values for a single feature. The color and shape of the dots reflect the class label. Since we only have two colors/sizes, the example application is a binary classification task.



(a) Assume a Decision Tree classifier that only performs binary splits. For each data distribution A-D, where (approximately) would the classifier split the values into 2 child nodes?

Solution:

- For all for distributions, the threshold for the best split would be around 42.5

(b) Let's assume our data points only have this one feature. Just by looking at the data distributions A-D for this single feature, what can we say about the "look" of the final decision tree (without any pre or post-pruning)?

Solution:

- Distributions B and C require only this single split to yield child nodes without any impurity. So for this branch of the Decision Tree, the learning algorithm stops.

- Distributions A and D require both 2 additional splits to ensure nodes without any impurity.

Note that this result might vary when considering more than this single feature.

(c) Given the results from (a) and (b), summarize how the distribution of feature values affects the training of a Decision Tree.

Solution:

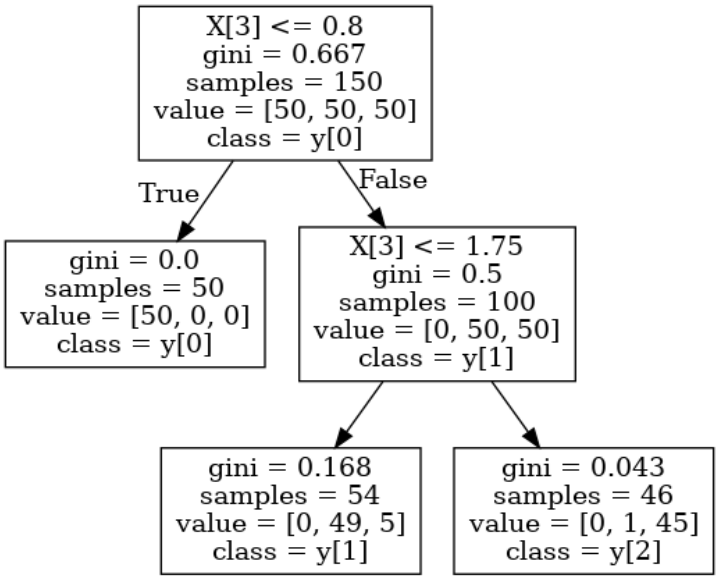
- As long as an outlier does not affect the "order" of data points with respect to their class labels, the outlier won't affect the split regarding the resulting child nodes. So in this regard, Decision Trees are somewhat robust against outliers.
- Even if the order of data points w.r.t. their class labels is preserved, the exact threshold where to perform the split still vary, of course
- Distributions A and require both 2 additional splits to ensure nodes without any impurity.
- If an outlier does affect the order data points w.r.t. their class labels, then this will generally result in a different split. And since single Decision Trees (not Tree Ensembles) are quite sensitive to changes in the dataset, such outliers do have an impact on the resulting Decision Tree.

(d) You can now add a single data point of Class **Blue** into Distribution A? Where would you place this new data point to maximize the negative effect on the resulting Decision Tree in terms of the required splits?

Solution:

- Placing the new data point of Class **Blue** between two data Points of Class **Red** would require 2 additional splits. Any other placement only in 1 split at worst (e.g., at value 5).
- An interesting special case is to place the new data point of Class **Blue** directly on top of an existing data point of Class **Red**. This would not result in more than 2 additional splits, but it would mean that the Decision Tree can no longer perfectly fit the data any longer.

2. **Interpreting Decision Trees.** The Decision Tree shown below has been trained over the IRIS Dataset with a maximum depth of 2. Recall that each data sample has 4 numerical features (all measures in centimeters), and is labeled with 1 out of 3 classes.



(a) What insights can you get from this Decision Tree?

Solution:

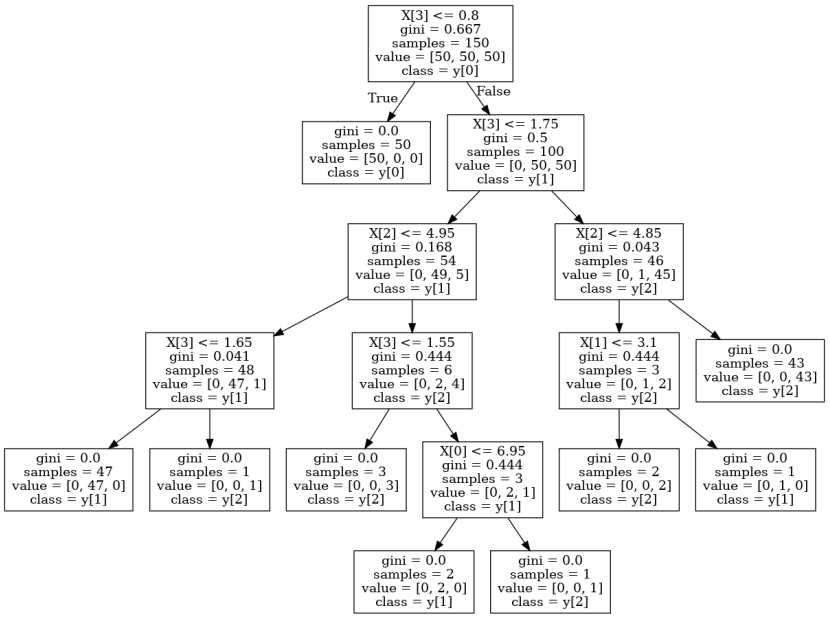
- Feature 3 is chosen for the first split at the root, it is thus the strongest predictor
- The samples of Class 0 are easiest to identify since the root split is sufficient to separate them from the samples of the other 2 classes.
- Side note: This example shows that a child may be split using the same feature as the parent node.

(b) Assume you train the Decision Tree without any restrictions on its maximum depth. Given the Decision Tree above, which statement can you make about the full Decision Tree.

Solution:

- The first 2 splits will naturally be identical since the specification of a maximum depth does not affect the algorithm for finding the best split.
- The full Decision tree will be deeper/higher since the child nodes of the "restricted" tree above are still impure.
- It's not really possible to tell how many more splits are needed as it heavily depends on the data distribution which is not sufficiently expressed in the Decision Tree.

- At the very minimum, 2 more splits are required to separate the data samples of 2 impure child nodes in the restricted tree.
- In the worst case, for each impure child node, O(N) more splits are needed where N is the number of data samples in that child node
- The Decision Tree below shows to full Decision Tree trained over the IRIS dataset



(c) How would your answer for (a) and (b) change if all the input features would have been standardized before training the Decision Tree.

Solution:

- Apart from the specific thresholds, nothing would change and the Decision Trees would look the same.
- Decision Trees do not take the interaction between features into account, so any form of normalization/standardization to make features "equally important" are not needed.
- Standardization does not affect the "order" of data samples with respect to their values, which in turn does not affect the algorithm for finding the best split (cf. Task 1)

(d) Assume someone gives you the optimal Decision Tree, i.e., optimal in the sense that it results in the highest accuracy (or any other suitable metric). What can we say about the root node of this optimal Decision Tree?

Solution:

- As the training algorithm for a Decision Tree is a heuristic, there is a non-zero probability that the optimal Decision Tree performs the first split not w.r.t. Feature 1.
- However, since Feature 3 is such a strong predictor, the probability that Feature 3 is used for the first split in the optimal Decision Tree is arguably very high.