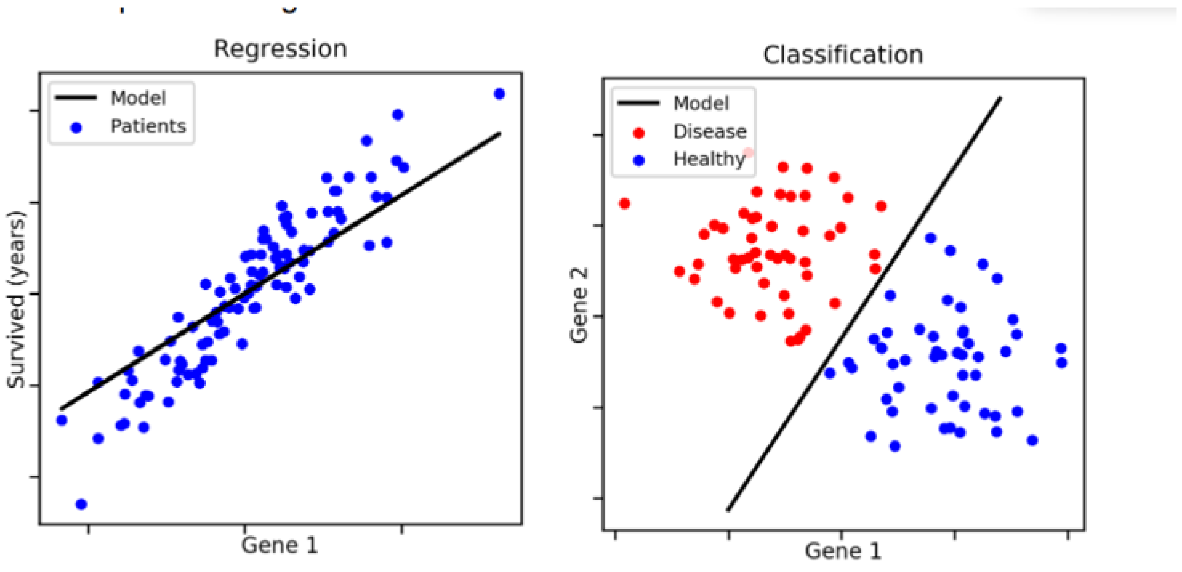
DATA MINING AND MACHINE LEARNING

ASSIGNMENT:1

1.[3 pts] Both classification and regression look for a function for prediction. Explain the difference in meaning of these functions using the example in slide p. 18 (Intro-DM-ML chapter)



From slide p.18 the key difference between regression and classification functions is in the type of output they predict, as reflected in the image.

1. **Regression:** The goal is to predict a continuous quantitative value. In the regression plot on the left, the model is predicting a numeric value — in this case, the number of years a patient has survived based on Gene 1 expression. The black line represents the regression model, which fits a continuous function to predict survival time as a linear function of the gene expression. The function outputs a real number (e.g., years survived), which can take any value in a continuous range.

2. **Classification:** The goal is to predict a qualitative, categorical outcome. The classification plot on the right shows a model classifying patients as either diseased (red points) or healthy (blue points) based on two gene expressions (Gene 1 and Gene 2). The black line in the plot is the decision boundary that separates the two classes, meaning the function splits the data into two distinct categories. The output is not a continuous number but rather a category label (diseased or healthy).

In summary:

- Regression outputs a continuous prediction (e.g., survival time).
- Classification outputs a categorical prediction (e.g., diseased or healthy).

2.[3pts/ea] Suppose we have a dataset with many features, and it contains every possible combination of feature values.

(i) explain why there is no difference in performance (supervised or unsupervised) between different algorithms

When a dataset contains every possible combination of feature values, the performance of different algorithms (both supervised and unsupervised) will be essentially the same for the following reasons:

Supervised Learning:

In supervised learning, the goal is to map input features to a known output label (target). When the dataset contains every possible combination of feature values, the following occurs:

- **No generalization needed**: Since all possible input-output combinations are already present in the dataset, any supervised learning algorithm will simply memorize or perfectly "fit" the data. The dataset will already contain the correct answer for every possible feature combination, and there's no need to generalize to unseen data.
- **No differentiation between algorithms**: Any algorithm, whether it's linear regression, decision trees, or neural networks, will fit the data perfectly. There's no difference in how they generalize or handle the data because there is no unseen data to test their ability to generalize. All algorithms will yield the same performance (likely a perfect accuracy) as they are working with a fully representative dataset.
- **Overfitting becomes irrelevant**: Overfitting typically happens when the model memorizes the training data without being able to generalize to unseen data. In this case, overfitting is no longer an issue because there is no "unseen" data — all possible combinations are already known.

Unsupervised Learning:

Unsupervised learning algorithms attempt to find patterns, clusters, or structures within the data without knowing the labels in advance. If every possible combination of feature values is present:

- **No hidden structure**: Since the dataset contains all possible feature combinations, there's no hidden pattern or clustering structure to discover. All potential relationships between features are fully represented and explicit within the dataset.
- **Same outcome for all algorithms**: Clustering algorithms (like k-means, hierarchical clustering, etc.) and dimensionality reduction algorithms (like PCA) will not discover new or meaningful patterns because the data already contains all combinations. As a result, the outcome for all unsupervised algorithms will be the same since there is nothing "new" to learn or reveal.

General Reasoning for Both:

- **Data completeness**: In both supervised and unsupervised scenarios, the fact that the dataset contains every possible combination of feature values means the dataset is complete. This completeness eliminates the need for an algorithm to learn from incomplete or sparse data and negates the need for generalization.
- **No algorithm-specific advantage**: Normally, different algorithms perform better or worse depending on factors like noise, complexity, or distribution of data. However, with a dataset that covers all possible scenarios, there is no advantage or disadvantage for one algorithm over another. All algorithms will have the same amount of information and will perform equally well (or equally poorly, depending on the goal).

Thus, in this context, there is no difference in performance between algorithms, whether supervised or unsupervised, as the task becomes simply handling or memorizing a complete dataset, rather than learning from incomplete or noisy data.

(ii) With enough number of features, explain why this type of dataset is impossible to happen.

A dataset containing every possible combination of feature values becomes impossible to construct when the number of features is large due to several fundamental challenges, including exponential growth in combinations, storage and memory limitations, and practical constraints in data collection.

Exponential Growth in Combinations (Combinatorial Explosion):

As the number of features increases, the number of possible combinations of feature values grows exponentially.

Suppose you have ‘k’ features, and each feature can take ‘n’ distinct values. The total number of combinations n^k

For example:

- If each feature is binary (0 or 1), the number of combinations is 2^k
- For 3 binary features, there are $2^3=8$ combinations.

This combinatorial explosion makes it impossible to store or handle all possible combinations of feature values as the number of features grows. Even for modest feature sizes, the number of possible combinations becomes enormous.

Memory and Storage Constraints:

Storing such a dataset would require an immense amount of storage, far beyond the capacity of current technology.

Even if you managed to store the dataset, loading and processing it in memory would require astronomical computational resources that exceed modern computer architectures' capabilities.

Data Collection Impossibility:

In real-world scenarios, collecting data for every possible combination of feature values is not feasible.

Collecting data for even a fraction of the possible feature combinations would take an unrealistic amount of time. For a large number of features, collecting data for each possible combination would be virtually impossible due to constraints in time, resources, and access.

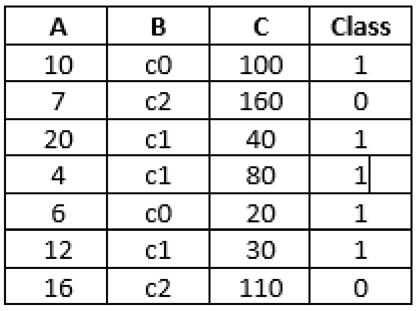
Overfitting:

A dataset with every possible combination of feature values is highly likely to overfit the training data. This means that the model would learn the training data too well, leading to poor generalization performance on new, unseen data.

Conclusion:

When the number of features becomes large, the combinatorial explosion leads to an exponential growth in possible combinations, making it impossible to store, process, or even collect all combinations of feature values. This is compounded by real-world constraints, such as physical or logical impossibilities and the curse of dimensionality, making a dataset containing every possible combination of feature values impractical and unrealistic.

3. [3pts/ea] With the following dataset,



A	B	C	Class
10	c0	100	1
7	c2	160	0
20	c1	40	1
4	c1	80	1
6	c0	20	1
12	c1	30	1
16	c2	110	0

(i). what kind of preprocessing tasks are necessary in this dataset.

The dataset shown in the image has several preprocessing tasks that would typically be required before it can be used in machine learning models. The necessary preprocessing steps include:

Handling Categorical Data (Column B):

Column B contains categorical data (c0, c1, c2), which cannot be directly used in most machine learning algorithms.

Preprocessing Task: Convert categorical values into numerical format using methods such as:

- **One-Hot Encoding:** Create new binary columns for each category (e.g., B_c0, B_c1, B_c2), where each column has a value of 1 if the corresponding row belongs to that category, and 0 otherwise.
- **Label Encoding:** Assign each category a unique integer value (c0 → 0, c1 → 1, c2 → 2), though this introduces ordinal relationships that may not exist.

Feature Scaling (Columns A and C):

Columns A and C contain numerical data that appear to be on different scales (e.g., column A ranges from 4 to 20, while column C ranges from 20 to 160).

Preprocessing Task: Apply feature scaling techniques to normalize or standardize the values so that the scales are consistent.

- **Standardization:** Subtract the mean and divide by the standard deviation (z-score normalization).
- **Normalization:** Scale the features to a range (e.g., between 0 and 1) using min-max scaling.

Handling the Target Variable (Class):

The Class column seems to be binary (0 and 1). This doesn't require any preprocessing if performing binary classification.

Preprocessing Task: Ensure the target variable is in the correct format (e.g., integers for classification tasks) and that there are no inconsistencies in its representation.

Handling Outliers:

Preprocessing Task: Detect and potentially remove or adjust outliers if they exist, as they can skew models, especially those sensitive to data distribution (e.g., linear regression).

The general steps involved in the Preprocessing Tasks are to:

1. Encode categorical data (Column B).
2. Scale numerical features (Columns A and C).
3. Check for missing values and handle if needed.
4. Ensure target variable (Class) is in the correct format.
5. Check for outliers in the data (optional).

These preprocessing steps will ensure the dataset is suitable for training machine learning models.

(ii) change A=10, 7, 20 to min-max scaling, respectively.

```
import numpy as np

# Original values of A

A = np.array([10, 7, 20])
```

```
# Min-Max Scaling formula: (x - min) / (max - min)

A_min = np.min(A)

A_max = np.max(A)

# Apply min-max scaling

A_scaled = (A - A_min) / (A_max - A_min)

A_scaled
```

After applying min-max scaling to the values of column A (10, 7, 20), the scaled values are:

- **A = 10 → 0.2308**
- **A = 7 → 0**
- **A = 20 → 1**

These scaled values are normalized between 0 and 1

(iii) show the result of one-hot-encoding for attribute B

```
import pandas as pd

# Original values of B

B = ['c0', 'c2', 'c1', 'c1', 'c0', 'c1', 'c2']

# Create a DataFrame with B values

df = pd.DataFrame({'B': B})

# Apply one-hot encoding

B_one_hot_encoded = pd.get_dummies(df, columns=['B'])

B_one_hot_encoded
```

The result of one-hot encoding for attribute **B** is as follows:

	B_c0	**B_c1**	**B_c2**
0	1	0	0
1	0	0	1

2	0	1	0
3	0	1	0
4	1	0	0
5	0	1	0
6	0	0	1

Each row is encoded into three columns, where a value of 1 represents the category present for that specific row, and 0 indicates absence.

(iv) Compute target encoder value with smoothing for B='c2'

To compute the target encoder value with smoothing for categorical variable B = 'c2', we can follow the process of target encoding with a smoothing parameter. Target encoding replaces a categorical value with the mean of the target variable (Class) for that category, but smoothing helps prevent overfitting, especially for categories with few examples.

The formula for target encoding with smoothing is:

$$\textbf{Target Encoding with Smoothing} = \frac{n_{category} \cdot mean_{category} + m \cdot mean_{global}}{n_{category} + m}$$

Where:

- n is the number of instances of category B=′c2′,
- mean is the mean of the target variable (Class) for category B=′c2′,
- m is a smoothing parameter (often set to a small constant, e.g., 1),
- global mean is the mean of the target variable (Class) across the entire dataset.

Step 1: Calculate the Required Values

1. Count of instances for B=′c2′:

   From the dataset:

   - Row 2: B=′c2′, Class = 0
   - Row 7: B=′c2′, Class = 0

   n=2

2. Mean Class for B=′c2′:

   Mean $\frac{0+0}{2}$ = 0

3. Global Mean:

   Total Class values: 1+0+1+1+1+1+0 = 5

   Number of rows: 7 global mean=5/7 ≈ 0.7143

4. Smoothing Parameter:

   Let's use m=1

Step 2: Compute the Target Encoding Value

Now we can substitute the values into the target encoding formula:

Target Encoding= $\frac{(2\cdot 0+1\cdot 0.7{,}43)}{2+1} = \frac{0+0.7143}{3} \approx 0.2381$

Result: The target encoding value for B=′c2′ with smoothing is approximately 0.2381.

(v) In slide p. 38 (Data preprocessing), explain why sigmoid function is used in target encoding smoothing? Can you suggest any other function for smoothing?

The sigmoid function is often used in target encoding smoothing because it maps any real-valued number into the range (0, 1). This property is particularly useful for probabilities, as it ensures that the smoothed target encoding remains a valid probability value.

Reasons for Using the Sigmoid Function:

1. **Bounded Output:** The sigmoid function compresses the output to a range between 0 and 1, making it suitable for scenarios where the target variable represents probabilities.
2. **Differentiability:** The sigmoid function is smooth and differentiable, which is beneficial for optimization algorithms that rely on gradients.
3. **Interpretability:** The outputs can be interpreted as probabilities, which align well with many machine learning models that predict binary outcomes.

Alternative Functions for Smoothing:

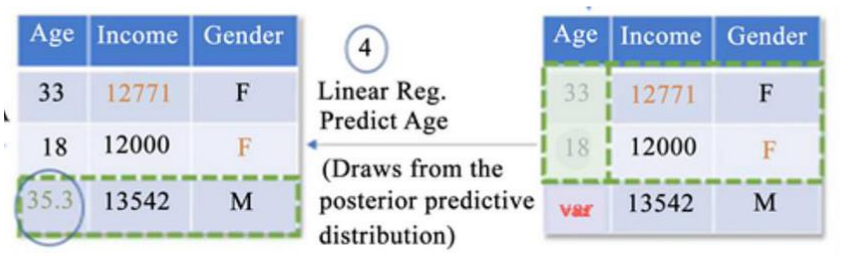
1. **Tanh Function:** Similar to the sigmoid, but it maps values to the range (-1, 1). It can be useful when the target values can take on both negative and positive values.

2. **Softmax Function:** Often used in multi-class classification problems. It transforms a vector of values into probabilities that sum to 1.

3. **Exponential Moving Average:** This method smooths values by taking a weighted average of the current value and previous values, which can be useful for time-series data.

4. **Gaussian Kernel Smoothing:** This method applies a Gaussian function to weight nearby observations more heavily, providing a smooth estimate of the target variable.

4.[3 pts] Each of these functions has its own characteristics and may be more suitable depending on the specific context and requirements of the modeling task.



Age	Income	Gender
33	12771	F
18	12000	F
35.3	13542	M

Age	Income	Gender
33	12771	F
18	12000	F
var	13542	M

The image illustrates a part of the Model-Based Iterative Imputation process, specifically focusing on predicting the "Age" variable using linear regression.

Current Data Table:

The left side shows a dataset with three columns: Age, Income, and Gender.

The rows contain values for two individuals (ages 33 and 18) with known income and gender, while the age for the third individual (with an income of 13542) is missing and noted as "var" (variable).

Linear Regression to Predict Age (Step 4):

The process involves using a linear regression model to predict the missing "Age" value based on the available data. The model uses the relationship between "Income" and "Gender" to estimate the age.

The regression model is built using the existing entries (ages 33 and 18) along with their corresponding income and gender.

Posterior Predictive Distribution:

The predicted age for the individual with the missing age is drawn from the posterior predictive distribution. This means that the prediction is not just a single point estimate but rather a distribution of possible values, reflecting uncertainty in the prediction.

This approach allows for more robust estimates by considering variability and uncertainty in the data.

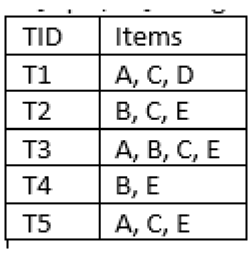
Updated Data Table:

The right side of the image shows the updated data table after applying the linear regression model. The predicted age for the individual with the missing value is now represented, although it is still marked as "var," indicating that it is a variable estimate derived from the regression model.

Summary

This process highlights how linear regression can be employed to fill in missing values in a dataset iteratively. By using existing relationships in the data, the model generates more informed estimates rather than relying on simpler methods like mean imputation. The use of posterior predictive distributions adds a layer of statistical rigor, acknowledging the uncertainty associated with predictions. This is a crucial step in preparing data for further analysis or modeling.

5.[3 pts/ea] Using the following transaction db,



TID	Items
T1	A, C, D
T2	B, C, E
T3	A, B, C, E
T4	B, E
T5	A, C, E

(min support=3)

(i) Show frequent 1-itemsets along with support values.

To find the frequent 1-itemsets from the provided transaction database with a minimum support of 3, we have to first count the occurrences of each item across the transactions.

Transaction Data:

- **T1**: A, C, D
- **T2**: B, C, E
- **T3**: A, B, C, E
- **T4**: B, E
- **T5**: A, C, E

Step 1: Count the occurrences of each item.

Item	Transactions	Count
A	T1,T3,T5	3
B	T2,T3,T4	3
C	T1,T2,T3,T5	4
D	T5	1
E	T2,T3,T4,T5	4

Step 2: Identify frequent 1-itemsets.

An itemset is considered frequent if its count is greater than or equal to the minimum support (3).

Frequent 1-itemsets:

- **A**: Support = 3
- **B**: Support = 3
- **C**: Support = 4
- **E**: Support = 4

Summary of Frequent 1-itemsets:

Item	Support
A	3
B	3
C	4
E	4

These are the frequent 1-itemsets along with their support values based on the given transaction database. Item D is not frequent as its support is less than the minimum threshold of 3.

(ii) Show frequent 2-itemsets along with support values.

To find the frequent 2-itemsets from the transaction database with a minimum support of 3, we have to first identify all possible pairs of items from the transactions and then count their occurrences.

Transaction Data:

- T1: A, C, D
- T2: B, C, E
- T3: A, B, C, E
- T4: B, E
- T5: A, C, E

Step 1: List all possible 2-itemsets and count their occurrences.

Possible 2-itemsets:

1. (A, B)
2. (A, C)
3. (A, D)
4. (A, E)
5. (B, C)
6. (B, D)

7. (B, E)
8. (C, D)
9. (C, E)
10. (D, E)

Counting Occurrences:

- (A, B): T3 (1)
- (A, C): T1, T3, T5 (3)
- (A, D): T1 (1)
- (A, E): T3, T5 (2)
- (B, C): T2, T3 (2)
- (B, D): None (0)
- (B, E): T2, T3, T4 (3)
- (C, D): T1 (1)
- (C, E): T2, T3, T5 (3)
- (D, E): None (0)

Step 2: Identify frequent 2-itemsets.

An itemset is considered frequent if its count is greater than or equal to the minimum support (3).

Frequent 2-itemsets:

- (A, C): Support = 3
- (B, E): Support = 3
- (C, E): Support = 3

Summary of Frequent 2-itemsets:

Itemset	Support
(A, C)	3
(B, E)	3
(C, E)	3

These are the frequent 2-itemsets along with their support values based on the given transaction database.

(iii) Show frequent 3-itemsets along with support values.

To find the frequent 3-itemsets from the transaction database with a minimum support of 3, we will identify all possible combinations of three items from the transactions and count their occurrences.

Transaction Data:

- T1: A, C, D
- T2: B, C, E
- T3: A, B, C, E
- T4: B, E
- T5: A, C, E

Step 1: List all possible 3-itemsets and count their occurrences.

Possible 3-itemsets:

1. (A, B, C)
2. (A, B, D)
3. (A, B, E)
4. (A, C, D)
5. (A, C, E)
6. (A, D, E)
7. (B, C, D)
8. (B, C, E)
9. (B, D, E)
10. (C, D, E)

Counting Occurrences:

- (A, B, C): T3 (1)
- (A, B, D): None (0)

- (A, B, E): None (0)
- (A, C, D): T1 (1)
- (A, C, E): T5, T3 (2)
- (A, D, E): None (0)
- (B, C, D): None (0)
- (B, C, E): T2, T3 (2)
- (B, D, E): None (0)
- (C, D, E): None (0)

Step 2: Identify frequent 3-itemsets.

An itemset is considered frequent if its count is greater than or equal to the minimum support (3).

Frequent 3-itemsets:

None of the 3-itemsets meet the minimum support of 3.

Summary of Frequent 3-itemsets:

Itemset	Support
None	0

There are no frequent 3-itemsets based on the given transaction database with the specified minimum support of 3.

(iv) For an association rule {B,C} -> {E}, compute support, confidence, and lift values, respectively.

To compute the support, confidence, and lift for the association rule {B,C}→{E}we will use the transaction data provided earlier.

Transaction Data:

- T1: A, C, D
- T2: B, C, E
- T3: A, B, C, E

- T4: B, E
- T5: A, C, E

Step 1: Calculate Support

Support of an itemset is defined as the proportion of transactions in which the itemset appears.

- Support of {B,C,E}: Count of transactions containing B and C and E.

Transactions containing {B,C,E}:

- T2: B, C, E
- T3: A, B, C, E

Count = 2

Total transactions = 5

Support({B,C}→{E})= $\frac{\text{Count of } \{B,C,E\}}{\text{Total Transactions}} = \frac{2}{5}$ = 0.4

Step 2: Calculate Confidence

Confidence of an association rule A→B is defined as the proportion of transactions that contain A which also contain B.

- Confidence of {B,C}→{E}:

Count of transactions containing {B,C}:

- T2: B, C, E
- T3: A, B, C, E

Count = 2

Confidence({B,C}→{E})= $\frac{Support(\{B,C,E\})}{Support(\{B,C\})} = \frac{2/5}{2/5}$ = 1.0

Step 3: Calculate Lift

Lift is defined as the ratio of the observed support to that expected if B and C were independent.

$$\text{Lift} = \frac{\text{Support}(\{B,C,E\})}{\text{Support}(\{B,C\}) * \text{Support}(\{E\})} = \frac{1.0}{0.8} = 1.25$$

Therefore the

- Support of B,C→E: 0.4
- Confidence of B,C→E: 1.0
- Lift of B,C→E: 1.25

(v) With N items anf M transactions. What is the time complexity generating candidate itemsets (along with support values) using brute force method (without Apriori principle)

The brute force method for generating candidate itemsets involves examining all possible combinations of items in the transaction database. So we can analyze the time complexity by using

Definitions:

- N: Number of unique items.
- M: Number of transactions.

Steps in the Brute Force Method:

1. Generate all possible item sets: The total number of possible item sets (including the empty set) is 2^N. This includes all combinations of the N items.
2. Count support for each itemset: For each candidate itemset, you need to check all M transactions to see if the itemset is present. This requires examining each transaction for each candidate itemset.

Time Complexity Analysis:

- Generating item sets: The generation of all possible item sets takes $O(2^N)$ time.
- Counting support: For each candidate itemset, counting its support involves checking all M transactions. This takes O(M) time per itemset.

Overall Time Complexity:

Therefore, the overall time complexity for generating candidate item sets and counting their support values using the brute force method is: $O(2^N \times M)$

Conclusion:

The time complexity of generating candidate item sets (along with support values) using the brute force method is O($2^N \times M$). This approach is computationally expensive, especially as N increases, making it impractical for large datasets.

(vi) [6 pts] Change transaction db to FP-Tree

To convert the given transaction database into an FP-Tree, we need to follow these steps:

1. Create a frequency table to determine the support count of each item.
2. Sort the items in each transaction based on their frequency in descending order.
3. Build the FP-Tree by inserting each transaction into the tree.

Given Transaction Database:

TID	Items
T1	A, C, D
T2	B, C, E
T3	A, B, C, E
T4	B, E
T5	A, C, E

Step 1: Create a Frequency Table

Count the occurrences of each item:

- A: 3
- B: 3
- C: 4
- D: 1
- E: 3

Step 2: Sort Items in Transactions

We will only keep items that meet the minimum support threshold (which is 3 in this case) and sort them based on their frequency:

- T1: A, C (D is removed, as its support is 1)
- T2: B, C, E (all items kept)
- T3: A, B, C, E (all items kept)
- T4: B, E (both items kept)
- T5: A, C, E (all items kept)

After sorting based on frequency (A, B, C, E are the only items kept):

- T1: A, C
- T2: B, C, E

- T3: A, B, C, E
- T4: B, E
- T5: A, C, E

Step 3: Build the FP-Tree

Now, we will insert each sorted transaction into the FP-Tree:

Insert T1 (A, C):

```
 (null)
   |
  A:1
   |
  C:1
```

Insert T2 (B, C, E):

```
(null)
  |
 A:1
  |
 C:1
  |
 B:1
  |
 E:1
```

Insert T3 (A, B, C, E):

```
(null)
  |
 A:2
  |
 C:2
  |
 B:1
  |
 E:1
```

Insert T4 (B, E):

```
(null)
  |
 A:2
  |
 C:2
  |
 B:2
  |
 E:2
```

Insert T5 (A, C, E):

```
(null)
  |
 A:3
  |
 C:3
  |
 B:2
  |
 E:2
```

Final FP-Tree Structure

The final FP-Tree structure will be as follows:

```
(null)
  |
 A:3
  |
 C:3
  |
 B:2
  |
 E:2
```

Therefore the FP-Tree constructed from the given transaction database effectively represents the transactions while maintaining the frequency counts of the items. Each path in the tree corresponds to a transaction, and the counts reflect the support of the items in the transactions.

6. [3 pts/ea] (Refer to p. 28-29 in Association) We are going to apply association rules in classification learning.

(i) To use association rules in classification learning, what form should the association rules take?

To use association rules in classification learning, the association rules should typically be as follows:

Form of Association Rules for Classification:

1. Antecedent (Condition): This is the "if" part of the rule, which consists of one or more items or features that are present in the data. It represents the conditions under which a certain outcome is expected.
2. Consequent (Conclusion): This is the "then" part of the rule, which indicates the class label or outcome that is predicted if the antecedent conditions are met.

General Structure:

If (Antecedent), then (Consequent)If (Antecedent), then (Consequent)

Example based on the rules from the image:

- Rule: (budget-resolution = n, MX-missile = n, el-salvador-aid = y) → republican

  Antecedent: budget-resolution = n, MX-missile = n, el-salvador-aid = y

  Consequent: republican

Key Characteristics:

- Support: The frequency with which the antecedent appears in the dataset.
- Confidence: The likelihood that the consequent is true given that the antecedent is true.
- Lift: A measure of how much more likely the consequent is given the antecedent compared to the overall probability of the consequent.

Application in Classification:

- The rules can be used to classify new instances by checking which rules apply (i.e., which antecedents match the features of the new instance) and then predicting the class based on the corresponding consequents.
- The rules should be evaluated for their support and confidence to ensure they are strong enough for reliable classification.

In classification learning using association rules, the rules should clearly define conditions (antecedents) that lead to specific outcomes (consequents), allowing for effective classification of new instances based on learned patterns.

(ii) We want to assign different weights to each rule, and use weighted voting for classification. Provide your opinion on methods for determining the importance of rules.

Assigning different weights to association rules for weighted voting in classification can enhance the predictive performance of the model. Several methods for determining the importance of rules are as follows:

1. Support and Confidence

- Support: This measures how frequently the rule appears in the dataset. Higher support can indicate a more reliable rule.
- Confidence: This measures the likelihood that the consequent is true when the antecedent is true. Higher confidence suggests stronger predictive power.
- Weight Calculation: One approach is to assign weights based on a combination of support and confidence, such as: Weight=Support × Confidence

2. Lift

- Lift: This measures how much more likely the consequent is given the antecedent compared to the overall probability of the consequent. A lift greater than 1 indicates a positive correlation.
- Weight Calculation: Assign weights based on lift values, as higher lift values suggest stronger relationships: Weight=Lift

3. Statistical Significance

- Use statistical tests (e.g., Chi-square test) to determine if the observed frequency of rule occurrences is significantly different from what would be expected by chance.
- Rules that are statistically significant can be assigned higher weights.

4. F1 Score

- Calculate the F1 score for each rule, which considers both precision and recall. This can be particularly useful in imbalanced datasets.
- Weight Calculation: Assign weights based on the F1 score to balance false positives and false negatives.

5. Ensemble Methods

- Use ensemble techniques like Random Forest or Gradient Boosting to evaluate the importance of each feature (or rule) in making predictions.
- Rules can be weighted according to their contribution to the overall model performance.

6. Domain Knowledge

- Incorporate expert knowledge or insights from the domain to assign weights based on the relevance or importance of specific rules in the context of the problem.

7. Rule Redundancy/Correlation

- Analyze the correlation between rules. If rules are redundant (i.e., they provide similar information), consider giving higher weights to less correlated rules to promote diversity in predictions.

Therefore combining multiple methods can provide a more robust approach to determining the importance of rules. The goal is to ensure that the most relevant and predictive rules have a greater influence on the classification outcome, improving the overall performance of the model.

7. [3 pts/ea] In p. 9 in Linear Regression, we use sum of squared error.

(i) Explain why we use squared value.

In linear regression, we use the sum of squared errors (SSE) for several important reasons:

Emphasis on Larger Errors

- Squaring the errors gives more weight to larger errors. This means that if a prediction is significantly off, it will have a disproportionately large impact on the total error. This helps in emphasizing the importance of minimizing larger discrepancies between predicted and actual values.

Mathematical Properties

- The squared error function is a smooth, continuous function, which makes it easier to optimize using calculus. The squared function is differentiable, allowing for the use of gradient descent and other optimization techniques to find the minimum error.

Non-Negativity

- Squaring ensures that all error values are non-negative, which means that the total error (RSS) cannot be negative. This property simplifies the interpretation of the error metric, as it always reflects a "cost" that needs to be minimized.

Normal Distribution Assumption

- Linear regression assumes that the errors (residuals) are normally distributed. The squared error loss function aligns with this assumption, leading to optimal parameter estimates under the assumption of normally distributed errors.

Geometric Interpretation

- In a geometric sense, minimizing the sum of squared errors corresponds to finding the point (line) that is closest to all data points in a Euclidean distance sense. This aligns with the intuitive idea of "best fit."

Therefore using squared values in the error function of linear regression provides a robust, mathematically tractable, and interpretable way to quantify and minimize prediction errors, enhancing the model's performance and reliability.

(ii) If we use sum of absolute value, do we have any problem ?

Using the sum of absolute errors (SAE) instead of the sum of squared errors (SSE) in regression has its own set of advantages and disadvantages. Some potential problems and considerations when using absolute errors are as follows:

Non-Differentiability

- The absolute value function is not differentiable at zero. This can complicate optimization algorithms that rely on gradient-based methods, as the gradient may not exist at certain points, making it harder to find a minimum.

Less Sensitivity to Outliers

- While this can be an advantage in some cases, using absolute errors means that larger errors do not have as much influence on the total error as they do with squared errors.

This could lead to a model that is less sensitive to outliers, potentially resulting in poorer fit in datasets where outliers are important.

Optimization Complexity

- Minimizing the sum of absolute errors typically requires linear programming techniques or iterative methods, which can be more computationally intensive compared to the closed-form solutions available for SSE.

Lack of Unique Solutions

- The sum of absolute errors can lead to multiple equally optimal solutions, especially in cases where data points are collinear. This can make interpretation of the model more complex, as there may not be a unique "best fit" line.

Geometric Interpretation

- The geometric interpretation of minimizing absolute errors corresponds to finding the median of the residuals rather than the mean. This can lead to different results than those obtained from minimizing squared errors, especially in skewed distributions.

Therefore while using the sum of absolute errors can provide robustness against outliers and simpler interpretations in some contexts, it also introduces challenges in optimization and can lead to less sensitivity to larger errors.

8. Given the error function of linear regression,

$$\boldsymbol{J(w) = \sum_i (y_i - \hat{y}_i)^2}$$

(i)[3 pts] If $\boldsymbol{\hat{y}_i = w_0 + w_1X_1 + w_2X_2 + \cdots + w_pX_p}$, is it guaranteed to find global optimum? Explain the reason.

In linear regression, the error function is given by:

$$J(w) = \sum_i (y_i - \hat{y}_i)^2$$

where $\hat{y}_i = w_0 + w_1X_1 + w_2X_2 + \cdots + w_pX_p$ This function guarantees a global optimum depends on several factors, particularly the nature of the error function and the model.

Reasons Why a Global Optimum is Guaranteed:

Convexity of the Error Function:

The error function J(w) is a quadratic function of the parameters w (the coefficients). Quadratic functions are convex, meaning that they have a single minimum point. In the context of linear regression, this means that there is a unique global minimum for the cost function.

Nature of the Model:

The linear model y^ᵢ is linear in terms of the parameters w. Since the model is linear, the relationship between the parameters and the output is straightforward, leading to a well-defined optimization landscape.

Closed-Form Solution:

Linear regression has a closed-form solution derived from the normal equations. This solution provides the exact values of the parameters that minimize the error function, confirming the existence of a global optimum.

Well-Behaved Data:

Assuming that the data is well-behaved (e.g., no perfect multicollinearity, sufficient variation in the predictors), the optimization process will converge to the global optimum.

Therefore, because the error function J(w) in linear regression is convex and has a unique global minimum due to its quadratic nature, it is guaranteed to find a global optimum when optimizing the model parameters.

(ii) [3 pts] Suppose $\hat{y}_i = w_0 + w_1X_1$. Show the derivative of $\frac{\partial J(w)}{\partial w_0}$ and $\frac{\partial J(w)}{\partial w_1}$, respectively

(8) (ii) To find the derivatives of the error function $J(\omega)$ w.r.t ω_0 and ω_1 in the linear regression model $\hat{y}_i = \omega_0 + \omega_1 X_1$ can be defined as:

$J(\omega) = \mathcal{E}_i (y_i - \hat{y}_i)^2 = \mathcal{E}_i (y_i - (\omega_0 + \omega_1 x_1))^2$

→ Derivative w.r.t ω_0

To find $\frac{\partial J(\omega)}{\partial \omega_0}$, we apply the chain rule

$$\frac{\partial J(\omega)}{\partial \omega_0} = \frac{\partial}{\partial \omega_0} \mathcal{E}_i (y_i - (\omega_0 + \omega_1 X_1))^2$$

Using the chain rule:

$$= \mathcal{E}_i \, 2(y_i - (\omega_0 + \omega_1 x_1)) \cdot \frac{\partial}{\partial \omega_0} (y_i - (\omega_0 + \omega_1 x_1))$$

the derivative of $y_i - (\omega_0 + \omega_1 x_1)$ w.r.t ω_0 is -1

$$= \mathcal{E}_i \, 2(y_i - (\omega_0 + \omega_1 x_1)) \cdot (-1)$$

Simplifying this:

$$= -2 \mathcal{E}_i (y_i - (\omega_0 + \omega_1 X_1))$$

Therefore we have $\frac{\partial J(\omega)}{\partial \omega_0} = -2 \mathcal{E}_i (y_i - (\omega_0 + \omega_1 x_1$

→ Derivative w.r.t ω_1

$$\Sigma_i 2(y_i-(\omega_0+\omega_1 x_1)) \cdot \frac{\partial}{\partial \omega_1}(y_i-(\omega_0+\omega_1 x_1))$$

the derivative of $y_i-(\omega_0+\omega_1 x_1)$ w.r.t ω_1 is $-x_1$:

$$= \Sigma_i 2(y_i-(\omega_0+\omega_1 x_1)) \cdot (-x_1)$$

Simplifying this :

$$= -2 \Sigma_i (y_i-(\omega_0+\omega_1 x_1)) x_1$$

$$\therefore \quad \frac{\partial J(\omega)}{\partial \omega_1} = -2 \Sigma_i (y_i-(\omega_0+\omega_1 x_1)) x_1$$

9. [5 pts] For data i, suppose y_i= target(label) and $\hat{y}_i$= prediction by linear regression.

Suppose $\hat{y} = 1 + 3 * x_1$. Given a dataset D={(1,6), (3,8)} Compute residual error (e_i) for each data and total sum of squared errors.

To compute the residual error (ei) for each data point and the total sum of squared errors, we need the equation of the linear regression line.

1. Find the equation of the linear regression line:

The equation of a linear regression line is given by:

y = mx + b

where:

- m is the slope of the line
- b is the y-intercept

To find m and b, we can use the following formulas:

m = (Σxy - (Σx * Σy) / n) / (Σx^2 - (Σx)^2 / n)

b = (Σy - m * Σx) / n

where:

- Σxy is the sum of the product of x and y values
- Σx is the sum of x values
- Σy is the sum of y values
- n is the number of data points

For the given dataset D={(1,6), (3,8)}, we have:

Σx = 1 + 3 = 4

Σy = 6 + 8 = 14

Σxy = (1 * 6) + (3 * 8) = 30

$\Sigma x^2 = 1^2 + 3^2 = 10$

$n = 2$

Using the formulas, we can calculate:

$m = (30 - (4 * 14) / 2) / (10 - (4)^2 / 2) = 1$

$b = (14 - 1 * 4) / 2 = 5$

Therefore, the equation of the linear regression line is:

$y = x + 5$

2. Compute the residual error (ei) for each data point:

The residual error is the difference between the actual y value (target) and the predicted y value (prediction) for each data point.

For the first data point (1,6):

ei = target - prediction = 6 - (1 + 5) = 0

For the second data point (3,8):

ei = target - prediction = 8 - (3 + 5) = 0

3. Compute the total sum of squared errors:

The total sum of squared errors is the sum of the squared residual errors.

Total sum of squared errors = $0^2 + 0^2 = 0$

Therefore, the residual error (ei) for each data point is 0, and the total sum of squared errors is 0. This indicates that the linear regression line perfectly fits the given data points.