Mathematical Logic for Sentiment Analysis

Logical Formulas

Let P be the proposition "The post contains positive sentiment."
Let N be the proposition "The post contains negative sentiment."
Let Q be the proposition "The post contains neutral sentiment."

The sentiment S can be represented as $S = P \vee N \vee Q$ (The post is positive or negative or neutral).

Machine Learning Model

Input: Social media post text
Output: Probability distribution over sentiments (positive, negative, neutral)

The machine learning model learns the mapping from post text to sentiments using a neural network.

$$\text{Output distribution} = \text{NeuralNetwork}(\text{PostText}) \tag{1}$$

Post Text: "I love the new movie! The plot was fantastic."

Sentiment Analysis Example

Social Media Post:

> "I love the new movie! The plot was fantastic."

Logical Formulas

Let P: The post contains positive sentiment.
Let N: The post contains negative sentiment.
Let Q: The post contains neutral sentiment.

The sentiment S can be represented as $S = P \vee N \vee Q$ (The post is positive or negative or neutral).

Rule-Based Analysis

1. If the post contains words like "love," "fantastic," etc., assign positive sentiment.
2. If the post contains words like "hate," "disappointing," etc., assign negative sentiment.
3. Otherwise, assign neutral sentiment.

Machine Learning Model (Rule-Based)

$$S = \begin{cases} P, & \text{if post contains positive words} \\ N, & \text{if post contains negative words} \\ Q, & \text{otherwise} \end{cases} \tag{2}$$

Sentiment Analysis Rules

Positive Sentiment Rules

- Presence of positive words: "love," "fantastic," "amazing," "great," etc.
- Positive expressions: "I enjoyed," "I'm happy," "I recommend," etc.
- Emoticons indicating positivity: etc.

Negative Sentiment Rules

- Presence of negative words: "hate," "disappointing," "awful," "terrible," etc.
- Negative expressions: "I didn't like," "I'm disappointed," "I wouldn't recommend," etc.
- Emoticons indicating negativity: etc.

Neutral Sentiment Rules

- General statements without strong positive or negative words.
- Descriptive statements without expressing a clear opinion.
- Informational content without emotional tone.

Product Recommendation Example

Product Features:

Good battery life (F), High customer rating (C), Affordable (P)

Logical Formulas

Let F: The product has good battery life.
Let C: The product has a high customer rating.
Let P: The product is affordable.
Let R: The product is recommended.

The recommendation R can be represented as $R = (F \wedge C) \vee (P \wedge C)$ (The product has good battery life and high customer rating, or the product is affordable and has a high customer rating).

Rule-Based Analysis

1. If the product has both good battery life (F) and a high customer rating (C), recommend the product.
2. If the product is affordable (P) and has a high customer rating (C), recommend the product.
3. Otherwise, do not recommend the product.

Machine Learning Model (Rule-Based)

$$R = \begin{cases} \text{Recommend}, & \text{if } (F \wedge C) \vee (P \wedge C) \\ \text{Do Not Recommend}, & \text{otherwise} \end{cases} \tag{3}$$

Numerical Example: Product Recommendation

Logical Formula for Product Recommendation

Let R: The product is recommended.
Let F: The product has good battery life.
Let C: The product has a high customer rating.
Let P: The product is affordable.

The recommendation R can be represented as:

$$R = (F \wedge C) \vee (P \wedge C)$$

Explanation

- $(F \wedge C)$: This part is true if the product has both good battery life (F) and a high customer rating (C).
- $(P \wedge C)$: This part is true if the product is affordable (P) and has a high customer rating (C).
- $(F \wedge C) \vee (P \wedge C)$: The entire expression is true if at least one of the conditions $(F \wedge C)$ or $(P \wedge C)$ is true. In other words, the product will be recommended if it has either good battery life and a high customer rating or if it is affordable and has a high customer rating.

Product Features:

Good battery life (F): Yes
High customer rating (C): 4.5 out of 5
Affordable (P): Yes

Logical Formulas

Let F: The product has good battery life.
Let C: The product has a high customer rating.
Let P: The product is affordable.
Let R: The product is recommended.

The recommendation R can be represented as $R = (F \wedge C) \vee (P \wedge C)$ (The product has good battery life and high customer rating, or the product is affordable and has a high customer rating).

Rule-Based Analysis

1. If the product has both good battery life (F) and a high customer rating (C), recommend the product.
2. If the product is affordable (P) and has a high customer rating (C), recommend the product.
3. Otherwise, do not recommend the product.

Calculation

$$R = (F \wedge C) \vee (P \wedge C) = (\text{Yes} \wedge 4.5) \vee (\text{Yes} \wedge 4.5)$$

Check each part of the formula:

- $F \wedge C$: Yes $\wedge$ 4.5 (True)
- $P \wedge C$: Yes $\wedge$ 4.5 (True)

Since at least one part of the formula is true, $R = \text{True}$.

Machine Learning Model (Rule-Based)

$$R = \begin{cases} \text{Recommend}, & \text{if } (F \wedge C) \vee (P \wedge C) \\ \text{Do Not Recommend}, & \text{otherwise} \end{cases}$$

Since R is true in this case, the machine learning model would recommend the product.

Numerical Example: Image and Speech Recognition

Recognition Features:

Image contains a cat ($I = \text{True}$)
Spoken word is "hello" ($S = \text{True}$)

Logical Formulas

Let I: The image contains a cat.
Let S: The spoken word is "hello."
Let R: The recognition is correct.

The recognition R can be represented as:

$$R = I \land S$$

Rule-Based Analysis

1. If the image contains a cat (I) and the spoken word is "hello" (S), the recognition is correct.
2. Otherwise, the recognition is incorrect.

Calculation

$$R = I \land S = \text{True} \land \text{True} = \text{True}$$

Machine Learning Model (Rule-Based)

$$R = \begin{cases} \text{Correct}, & \text{if } I \land S \\ \text{Incorrect}, & \text{otherwise} \end{cases}$$

Since R is true in this case, the machine learning model would classify the recognition as "Correct."

Image and Speech Recognition

Image Recognition

Problem Statement

Given an image, determine what objects or patterns are present in the image.

Solution Approach

1. **Feature Extraction:**
   - Represent the image using features such as edges, corners, or complex patterns.
   - Common techniques include using Convolutional Neural Networks (CNNs) for automatic feature learning.
2. **Model Training:**
   - Train a machine learning model, often a neural network, on a labeled dataset of images.

- The model learns to associate extracted features with specific classes (e.g., recognizing a cat).

3. **Classification:**
    - Use the trained model to classify new images into predefined categories.
    - The model outputs a probability distribution over possible classes.

Common Algorithms

- Convolutional Neural Networks (CNNs)
- Transfer learning (fine-tuning pre-trained models)

Speech Recognition

Problem Statement

Given an audio signal, transcribe the spoken words.

Solution Approach

1. **Feature Extraction:**
    - Convert the audio signal into a spectrogram or features representing frequency content over time.
    - Common techniques include Mel-Frequency Cepstral Coefficients (MFCCs).
2. **Model Training:**
    - Train a machine learning or deep learning model on a labeled dataset of audio samples.
    - The model learns the mapping between audio features and corresponding transcriptions.
3. **Sequence Modeling:**
    - Use recurrent neural networks (RNNs) or sequence-to-sequence models to handle sequential nature of speech.

Common Algorithms

- Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks
- Connectionist Temporal Classification (CTC) loss

Logic Behind the Problems

1. **Data Representation:**
   - Both tasks require effective representation of input data.
   - Images are represented as pixel values, and audio signals are transformed into spectrograms or features.
2. **Feature Learning:**
   - Deep learning models, especially CNNs and RNNs, excel at learning hierarchical features from raw data.
3. **Training on Labeled Data:**
   - Supervised learning is crucial for both tasks.
   - Models need large labeled datasets to generalize well to unseen data.
4. **Complex Model Architectures:**
   - State-of-the-art models involve complex architectures to capture intricate patterns and dependencies.

Popular Frameworks

- TensorFlow and PyTorch are widely used for implementing deep learning models in image and speech recognition.

Numerical Example: Online Fraud Detection

Transaction Features:

Transaction amount is high ($A = \text{High}$)
Merchant has a low reputation ($M = \text{Low}$)

Logical Formulas

Let A: Transaction amount is high.
Let M: Merchant has a low reputation.
Let F: The transaction is fraudulent.
The fraud detection logic can be represented as:

$$F = A \wedge M$$

Rule-Based Analysis

1. If the transaction amount is high (A) and the merchant has a low reputation (M), the transaction is fraudulent.
2. Otherwise, the transaction is not fraudulent.

Calculation

$$F = A \wedge M = \text{High} \wedge \text{Low} = \text{Fraudulent}$$

Machine Learning Model (Rule-Based)

$$F = \begin{cases} \text{Fraudulent}, & \text{if } A \wedge M \\ \text{Not Fraudulent}, & \text{otherwise} \end{cases}$$

Numerical Example: Online Fraud Detection

Transaction Features:

Transaction amount is 500 ($A = 500$)
Merchant has a low reputation ($M = 3$)

Logical Formulas

Let A: Transaction amount is high.
Let M: Merchant has a low reputation.
Let F: The transaction is fraudulent.
The fraud detection logic can be represented as:

$$F = A \wedge M$$

Rule-Based Analysis

1. If the transaction amount is high ($A = 500$) and the merchant has a low reputation ($M = 3$), the transaction is fraudulent.
2. Otherwise, the transaction is not fraudulent.

Calculation

$$F = A \wedge M = 500 \wedge 3 = 1500$$

Machine Learning Model (Rule-Based)

$$F = \begin{cases} \text{Fraudulent}, & \text{if } A \wedge M \\ \text{Not Fraudulent}, & \text{otherwise} \end{cases}$$

Decision Tree Algorithm: Step-by-Step Explanation

Dataset

Consider a dataset with features X_1 and X_2 and a binary target variable Y:

X_1	X_2	Y
3	5	0
2	4	1
4	6	0
5	2	1

Step 1: Splitting Criteria

Choose a feature and a threshold to split the dataset. For example, let's split based on $X_1 > 3.5$.

Step 2: Build the Tree

if $X_1 > 3.5$:
  if $X_2 > 4.5$:
    Leaf Node 1: $Y = 0$
  else:
    Leaf Node 2: $Y = 1$
else:
  Leaf Node 3: $Y = 0$

This structure represents a simple decision tree. Each internal node represents a decision based on a feature and threshold, and each leaf node represents the predicted class.

Step 3: Leaf Nodes

Assign class labels to leaf nodes based on the majority class in the corresponding subsets:

Leaf Node 1: $Y = 0$ (Subset where $X_1 > 3.5$ and $X_2 > 4.5$)
Leaf Node 2: $Y = 1$ (Subset where $X_1 > 3.5$ and $X_2 \leq 4.5$)
Leaf Node 3: $Y = 0$ (Subset where $X_1 \leq 3.5$)

This decision tree can be used to predict the target variable Y for new instances by following the decision path based on the features.

Decision Tree Algorithm: Numerical Example

Dataset:

X_1	X_2	Y
3	7	0
5	8	1
4	6	0
8	5	1

Step-by-Step Explanation

1. Initial Impurity:

Calculate the impurity of the entire dataset.

$$\text{Impurity} = 1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2$$

2. Feature Selection:

Calculate the impurity reduction for each feature.

Feature	Impurity Reduction
X_1	0.25
X_2	0.125

Select X_1 as the splitting feature.

3. Node Splitting:

Split the dataset based on X_1 with a threshold (e.g., $X_1 < 4.5$).

4. Recursion:

Repeat steps 1-3 for each child node.

5. Leaf Nodes:

Assign the majority class to each leaf node.

Dataset

Consider the following dataset:

CreditScore	Income	LoanApproval
700	50000	1
600	30000	0
750	80000	1
650	40000	0

Step 1: Splitting Criteria

Choose a feature and a threshold to split the dataset. For example, let's split based on CreditScore > 650.

Step 2: Build the Tree

```
if CreditScore > 650 :
  if Income > 35000 :
    Leaf Node 1: LoanApproval = 1
  else:
    Leaf Node 2: LoanApproval = 0
else:
  Leaf Node 3: LoanApproval = 0
```

Step 3: Leaf Nodes

Leaf Node 1: LoanApproval $= 1$ (Subset where CreditScore > 650 and Income > 35000)
Leaf Node 2: LoanApproval $= 0$ (Subset where CreditScore > 650 and Income ≤ 35000)
Leaf Node 3: LoanApproval $= 0$ (Subset where CreditScore ≤ 650)

Decision Tree Example: Placement in Computer Science Engineering

Dataset

Consider the following dataset:

ProgrammingScore	DataStructuresScore	Placement
80	75	1
60	65	0
90	80	1
70	50	0

Step 1: Splitting Criteria

Choose a feature and a threshold to split the dataset. For example, let's split based on ProgrammingScore > 75.

Step 2: Build the Tree

```
if ProgrammingScore > 75 :
  if DataStructuresScore > 70 :
    Leaf Node 1: Placement = 1
  else:
    Leaf Node 2: Placement = 0
else:
  Leaf Node 3: Placement = 0
```

Step 3: Leaf Nodes

Leaf Node 1: Placement = 1 (Subset where ProgrammingScore > 75 and DataStructuresScore > 70)
Leaf Node 2: Placement = 0 (Subset where ProgrammingScore > 75 and DataStructuresScore ≤ 70)
Leaf Node 3: Placement = 0 (Subset where ProgrammingScore ≤ 75)

Issues with Decision Trees

1. **Overfitting:**
   - **Issue:** Decision trees are prone to overfitting, especially when the tree is deep and captures noise in the training data.
   - **Explanation:** A deep tree may fit the training data perfectly but may not generalize well to unseen data. It captures noise and outliers in the training set, leading to poor performance on new data.
2. **Sensitivity to Data Changes:**
   - **Issue:** Small changes in the training data can result in a completely different tree structure.
   - **Explanation:** Decision trees are sensitive to variations in the training data, which can lead to instability. A small change in input data might lead to a significant change in the tree structure.
3. **High Variance:**
   - **Issue:** Decision trees can have high variance, leading to different predictions for similar instances.
   - **Explanation:** Due to their sensitivity to data changes and tendency to overfit, decision trees can have high variance. This can be mitigated by using techniques like pruning and ensemble methods.
4. **Bias Towards Dominant Classes:**
   - **Issue:** Decision trees tend to be biased toward dominant classes in imbalanced datasets.

- **Explanation:** In datasets where one class significantly outnumbers the others, the decision tree might favor the dominant class, leading to suboptimal predictions for minority classes.

5. **Limited Expressiveness:**
   - **Issue:** Decision trees may struggle to express complex relationships in the data.
   - **Explanation:** Decision trees are composed of simple decision nodes, and creating complex decision boundaries might require a large number of nodes. Other algorithms, like ensemble methods or neural networks, may be better suited for capturing intricate patterns.
6. **Instability:**
   - **Issue:** Small changes in the training data or input features can result in a different tree structure.
   - **Explanation:** The decision tree-building process is based on greedy algorithms, which make decisions at each node independently. As a result, small changes in input data can lead to different splits and, ultimately, different tree structures.
7. **Difficulty Handling Numeric Data:**
   - **Issue:** Decision trees can be inefficient in handling numeric features.
   - **Explanation:** Decision trees work well with categorical features but might struggle with continuous numeric features. Techniques like binning or using tree variants like Random Forests can mitigate this issue.
8. **Biased Towards Features with More Levels:**
   - **Issue:** Features with more levels are often favored during the tree-building process.
   - **Explanation:** Features with a large number of levels may be preferred over features with fewer levels, impacting the fairness of the model.

Techniques and Solutions for Issues with Decision Trees

1. **Overfitting:**
   - **Technique:** Pruning the tree by limiting its depth or setting a minimum number of samples required to split a node.
   - **Solution:** Use hyperparameters like ‘max_depth‘ and ‘min_samples_split‘ to control the size of the tree.

2. **Sensitivity to Data Changes:**
    - **Technique:** Ensemble methods such as Random Forests, which build multiple trees and average their predictions.
    - **Solution:** Utilize Random Forests to reduce sensitivity to individual data variations.
3. **High Variance:**
    - **Technique:** Ensemble methods like Bagging or Boosting.
    - **Solution:** Apply techniques such as Random Forests (Bagging) or Gradient Boosting (Boosting) to reduce variance and improve generalization.
4. **Bias Towards Dominant Classes:**
    - **Technique:** Adjust class weights or use balanced datasets during training.
    - **Solution:** Specify class weights or provide balanced datasets to mitigate bias towards dominant classes.
5. **Limited Expressiveness:**
    - **Technique:** Use ensemble methods or more complex models like Random Forests, Gradient Boosting, or Neural Networks.
    - **Solution:** Employ models with higher expressiveness to capture complex relationships in the data.
6. **Instability:**
    - **Technique:** Use techniques like feature bagging or adding randomization during tree building.
    - **Solution:** Apply techniques to introduce randomness, reducing sensitivity to small changes in the training data.
7. **Difficulty Handling Numeric Data:**
    - **Technique:** Binning or discretization of numeric features.
    - **Solution:** Convert numeric features into categorical features through binning to enhance decision tree performance.
8. **Biased Towards Features with More Levels:**
    - **Technique:** Feature engineering, normalization, or dimensionality reduction.
    - **Solution:** Preprocess data by engineering features, normalizing, or reducing dimensionality to avoid bias towards features with more levels.

Step-by-Step Explanation for K-Nearest Neighbor Rule

1. K-Nearest Neighbor Rule:

K-nearest neighbor is a classification algorithm that assigns a class label to an input based on the majority class of its k-nearest neighbors in the feature space.

2. Measuring Similarity:

To determine the "closeness" of data points, a common measure is Euclidean distance. For two points (x_1, y_1) and (x_2, y_2), the Euclidean distance d is calculated as:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

3. Dealing with Irrelevant Attributes:

Issue: Including irrelevant attributes may negatively impact the distance calculation.
Solution: Feature selection or dimensionality reduction techniques can be employed to exclude irrelevant attributes. For instance, consider a dataset with attributes $\{x, y, z\}$, and if z is irrelevant, you can exclude it.

4. Scaling Problems:

Issue: Features with different scales can dominate the distance calculation.
Solution: Standardize or normalize features to bring them to a similar scale. For example, normalize x and y between 0 and 1:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

$$y' = \frac{y - \min(y)}{\max(y) - \min(y)}$$

Real-Time Example for K-Nearest Neighbor Rule

Consider a dataset with the following points:

Point A: (ProgrammingScore = 80, MathsScore = 90, EnglishScore = 75)
Point B: (ProgrammingScore = 65, MathsScore = 75, EnglishScore = 80)
Point C: (ProgrammingScore = 95, MathsScore = 85, EnglishScore = 70)
New Point X: (ProgrammingScore = 70, MathsScore = 80, EnglishScore = 85)

We want to classify New Point X using KNN with $k = 2$. Let's calculate the Euclidean distance, deal with irrelevant attributes, and address scaling problems.

1. Calculate Euclidean Distance:

For points A, B, and C, calculate the Euclidean distance to New Point X using the formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

2. Deal with Irrelevant Attributes:

Assuming an irrelevant attribute Z, exclude Z from the distance calculation.

3. Address Scaling Problems:

Normalize the scores between 0 and 1:

$$\text{Normalized Score} = \frac{\text{Score} - \min(\text{Score})}{\max(\text{Score}) - \min(\text{Score})}$$

Apply normalization to ProgrammingScore, MathsScore, and EnglishScore.

K-Nearest Neighbors (KNN) Classification Example

Given dataset:

Point A: (ProgrammingScore = 80, MathsScore = 90, EnglishScore = 75)
Point B: (ProgrammingScore = 65, MathsScore = 75, EnglishScore = 80)
Point C: (ProgrammingScore = 95, MathsScore = 85, EnglishScore = 70)
New Point X: (ProgrammingScore = 70, MathsScore = 80, EnglishScore = 85)

Step 1: Calculate Euclidean Distance

For New Point X:

$$d_{AX} = \sqrt{(80 - 70)^2 + (90 - 80)^2 + (75 - 85)^2}$$
$$d_{BX} = \sqrt{(65 - 70)^2 + (75 - 80)^2 + (80 - 85)^2}$$
$$d_{CX} = \sqrt{(95 - 70)^2 + (85 - 80)^2 + (70 - 85)^2}$$

Step 2: Choose K and Find Nearest Neighbors

With $k = 2$, the two nearest neighbors are Point B and Point A.

Step 3: Make Prediction

Since the majority class among the two nearest neighbors is Class 1, we predict that New Point X belongs to Class 1.

K-Nearest Neighbors (KNN) Classification Example with $k = 3$

Given dataset:

Point A: (ProgrammingScore = 80, MathsScore = 90, EnglishScore = 75)
Point B: (ProgrammingScore = 65, MathsScore = 75, EnglishScore = 80)
Point C: (ProgrammingScore = 95, MathsScore = 85, EnglishScore = 70)
New Point X: (ProgrammingScore = 70, MathsScore = 80, EnglishScore = 85)

Step 1: Calculate Euclidean Distance

For New Point X:

$$d_{AX} = \sqrt{(80-70)^2 + (90-80)^2 + (75-85)^2}$$
$$d_{BX} = \sqrt{(65-70)^2 + (75-80)^2 + (80-85)^2}$$
$$d_{CX} = \sqrt{(95-70)^2 + (85-80)^2 + (70-85)^2}$$

Step 2: Choose K and Find Nearest Neighbors

With $k = 3$, the three nearest neighbors are Point B, Point A, and Point C.

Step 3: Make Prediction

Considering the majority class among the three nearest neighbors:

Point B: Class 1
Point A: Class 1
Point C: Class 0

Since the majority class is Class 1, we predict that New Point X belongs to Class 1.

Choosing k in K-Nearest Neighbors (KNN)

The choice of the value for k in K-Nearest Neighbors (KNN) is an important consideration in the modeling process. Here are some factors to keep in mind:

1. **Odd vs. Even k:**
   - It's often recommended to use an odd k value to avoid ties when determining the majority class.
   - If there is a tie in the number of neighbors, an odd k ensures a clear majority.
2. **Dataset Size:**
   - For small datasets, consider using a smaller k value to capture local patterns.
   - Larger datasets may benefit from larger k values.
3. **Impact of Outliers:**
   - Smaller k values are more sensitive to outliers and noisy data.
   - Larger k values tend to be more robust to outliers.
4. **Computational Complexity:**
   - As k increases, the computational complexity of predicting a new instance also increases.
   - Larger k values might result in slower predictions.
5. **Balancing Bias and Variance:**
   - Smaller k values tend to have lower bias but higher variance.
   - Larger k values tend to have higher bias but lower variance.
6. **Cross-Validation:**
   - Use cross-validation techniques to find an optimal k value by evaluating the performance on different folds.
7. **Rule of Thumb:**
   - A common rule of thumb is to set k to the square root of the number of data points (n).

There is no strict maximum limit for k, but the choice involves trade-offs between bias and variance. Experiment with different k values based on the characteristics of your dataset and specific goals.

Dataset

Consider the following dataset:

CreditScore	Income	LoanApproval
700	50000	1
600	30000	0
750	80000	1
650	40000	0