National Institute of Technology
Srinagar

PsyExpert: Rule-Based Emotion and Personality Analysis System

By

Khushboo Dar (2022BITE002)

Hitesh (2022BITE003)

Submit to: Aqib Zahoor

July 2025

Abstract

PsyExpert is a rule-based psychological analysis system combining text-based emotion detection with an interactive MBTI questionnaire. Featuring an enhanced Tkinter GUI, it provides personality insights without machine learning dependencies.

Aim

Develop a rule-based system that:

1. Detects emotions via keyword patterns
2. Assesses personality using MBTI
3. Delivers tailored advice
4. Features conversational GUI

Methodology

Emotion Detection

1. 7 emotion categories
2. Negation/intensity detection
3. Weighted scoring

Personality Analysis

1. Dual assessment modes
2. 4-question MBTI test
3. Persistent type storage

Conclusion

PsyExpert effectively combines text analysis with guided questioning in a transparent rule-based framework. Future enhancements include user profiles while maintaining explainability.

Appendix A:Implementation Code

Listing 1: main.py - Main Application

```
import re
import random
import tkinter as tk
from tkinter import scrolledtext, messagebox, simpledialog

class PsyExpert:
    def __init__(self):
        # Initialize knowledge bases
        self.emotion_keywords = self._init_emotion_keywords()
        self.mbti_rules = self._init_mbti_rules()
        self.advice_rules = self._init_advice_rules()
        self.mbti_questions = self._init_mbti_questions()
        self.user_mbti = None

        # Create GUI
        self.root = tk.Tk()
        self.root.title("PsyExpert - Psychological Analysis
           System")
        self._create_gui()
    def _init_emotion_keywords(self):
    #Initialize emotion detection rules with enhanced negative
       terms
      return {
        'happy': ['happy', 'joy', 'excited', 'great', 'wonderful'
           , 'awesome',
                'fantastic', 'amazing', 'delighted', 'content',
                   'pleased',
                'joyful', 'cheerful', 'optimistic', 'thrilled'],
        'sad': ['sad', 'depressed', 'unhappy', 'miserable', '
           gloomy', 'down',
            'heartbroken', 'hopeless', 'lonely', 'empty', '
               tearful',
            'grief', 'sorrow', 'despair', 'melancholy', 'blue'
               , 'hurt'],
        'angry': ['angry', 'mad', 'furious', 'annoyed', '
           irritated', 'frustrated',
                'outraged', 'bitter', 'resentful', 'hostile', '
                   aggravated',
                'hate', 'rage', 'livid', 'fuming', 'irate', '
                   seething', 'enraged'],
        'fearful': ['scared', 'afraid', 'fear', 'terrified', '
           anxious', 'nervous',
                'worried', 'panicked', 'stressed', '
                   apprehensive', 'tense',
                'dread', 'horror', 'phobia', 'uneasy', '
                   distress', 'overwhelmed'],
```

```
        'surprised': ['surprised', 'shocked', 'amazed', '
            astonished', 'astounded',
                      'stunned', 'dumbfounded', 'flabbergasted', '
                          bewildered'],
        'disgusted': ['disgust', 'disgusted', 'revolted', '
            sickened', 'repulsed',
                      'contempt', 'aversion', 'distaste', '
                          horrified', 'appalled'],
        'negative': ['stress', 'pain', 'suffer', 'ill', 'sick', '
            bad', 'awful',
                     'terrible', 'negative', 'bleak', 'dismal']
    }

    def _init_mbti_rules(self):
        """Initialize MBTI personality rules"""
        return {
            'IE': [
                ('I', ['alone', 'quiet', 'think', 'reflect', '
                    private', 'solitude',
                        'reserved', 'recharge', 'individual', '
                            concentrate']),
                ('E', ['people', 'party', 'social', 'talk', '
                    energized', 'group',
                        'outgoing', 'interact', 'external', '
                            communicate'])
            ],
            'SN': [
                ('S', ['facts', 'details', 'present', 'practical'
                    , 'real', 'actual',
                        'specific', 'concrete', 'experience', '
                            sensible']),
                ('N', ['ideas', 'future', 'imagine', 'abstract',
                    'theories', 'possibilities',
                        'patterns', 'insights', 'symbolic', '
                            conceptual'])
            ],
            'TF': [
                ('T', ['logic', 'objective', 'analysis', '
                    critique', 'fair', 'reason',
                        'principles', 'consistency', 'justice', '
                            impersonal']),
                ('F', ['harmony', 'values', 'compassion', '
                    empathy', 'feelings', 'caring',
                        'relationships', 'subjective', 'personal',
                            'considerate'])
            ],
            'JP': [
                ('J', ['plan', 'organized', 'structure', 'decide'
                    , 'control', 'scheduled',
                        'closure', 'methodical', 'prepared', '
                            systematic']),
```

```
                ('P', ['flexible', 'spontaneous', 'options', '
                    adapt', 'curious', 'open-ended',
                        'explore', 'wait', 'pressure', 'impulsive'
                            ])
            ]
        }

    def _init_advice_rules(self):
        """Initialize psychological advice rules"""
        return {
            'happy': [
                "It's wonderful that you're feeling happy!
                    Consider journaling about what's bringing you
                    joy to reinforce these positive emotions.",
                "Positive emotions are great for building
                    resilience. Think about how you can create
                    more moments like these in your daily life."
            ],
            'sad': [
                "When feeling sad, try reaching out to friends or
                     engaging in activities you normally enjoy.
                    Remember that feelings are temporary.",
                "Sadness often signals something needs attention.
                     Be gentle with yourself and consider what
                    might need care or healing."
            ],
            'angry': [
                "For anger, try deep breathing exercises or
                    physical activity to release tension. Consider
                     what's really behind the anger.",
                "Anger often points to unmet needs or violated
                    boundaries. Reflect on what you might need to
                    communicate or change."
            ],
            'fearful': [
                "With fear or anxiety, grounding techniques can
                    help. Name 5 things you can see, 4 you can
                    touch, 3 you can hear, 2 you can smell, and 1
                    you can taste.",
                "Anxiety often comes from anticipating the future
                    . Try focusing on the present moment and what
                    you can control right now."
            ],
            'surprised': [
                "Unexpected events can be disorienting. Take time
                     to process the surprise before reacting.",
                "Surprises challenge our expectations. Consider
                    whether this surprise might lead to new
                    opportunities or perspectives."
            ],
            'disgusted': [
```

```
                "Disgust often signals something violates your
                   values. Consider what boundaries might need
                   reinforcement.",
                "This feeling of disgust may be protecting you
                   from something harmful. Reflect on what values
                    are being challenged."
            ],
            'neutral': [
                "You seem to be in a neutral state. This can be a
                    good time for reflection or mindfulness
                   practice.",
                "Neutral emotions provide balance. Consider what
                   might help you maintain this equilibrium."
            ]
        }
    def _init_mbti_questions(self):# new
        """Initialize MBTI assessment questions"""
        return [
            {
                'dimension': 'IE',
                'question': "When you need to recharge, do you
                   prefer:\n\n"
                            "1) Spending time alone (Introversion
                               )\n"
                            "2) Being with friends or in social
                               settings (Extroversion)",
                'options': {'1': 'I', '2': 'E'}
            },
            {
                'dimension': 'SN',
                'question': "Which do you find more natural:\n\n"
                           "1) Focusing on facts and concrete
                              details (Sensing)\n"
                           "2) Thinking about possibilities and
                              abstract ideas (Intuition)",
                'options': {'1': 'S', '2': 'N'}
            },
            {
                'dimension': 'TF',
                'question': "When making decisions, do you tend
                   to:\n\n"
                           "1) Prioritize logic and objective
                              analysis (Thinking)\n"
                           "2) Consider people's feelings and
                              subjective values (Feeling)",
                'options': {'1': 'T', '2': 'F'}
            },
            {
                'dimension': 'JP',
                'question': "In your daily life, do you prefer:\n
                   \n"
```

```
                        "1) Structure, plans, and decided
                           things (Judging)\n"
                        "2) Flexibility, spontaneity, and
                           keeping options open (Perceiving)",
                'options': {'1': 'J', '2': 'P'}
            }
        ]

    def conduct_mbti_assessment(self):#new
        """Conduct MBTI assessment through dialog questions"""
        mbti_result = []

        for question in self.mbti_questions:
            while True:
                answer = simpledialog.askstring(
                    "MBTI Assessment",
                    question['question'],
                    parent=self.root
                )

                if answer and answer.strip() in ['1', '2']:
                    mbti_result.append(question['options'][answer
                       .strip()])
                    break
                elif answer is None:  # User clicked cancel
                    return None
                else:
                    messagebox.showerror("Invalid Input", "Please
                        enter 1 or 2")

        self.user_mbti = ''.join(mbti_result)
        self._update_conversation(
            f"PsyExpert: Based on your answers, your likely MBTI
               type is {self.user_mbti}.\n"
            f"PsyExpert: You can now share your thoughts and I'll
                analyze them considering your personality.\n"
        )
        self.personality_label.config(text=f"Personality: {self.
           user_mbti} (assessed)")
        return self.user_mbti

    def _create_gui(self):
        """Create the graphical user interface with MBTI
           assessment button"""
        # Configure main window
        self.root.geometry("800x600")
        self.root.configure(bg="#f0f0f0")

        # Main conversation area
        self.conversation = scrolledtext.ScrolledText(
```

```
            self.root, width=70, height=25, wrap=tk.WORD, bg="
                white", fg="#333333",
            font=("Arial", 10))
        self.conversation.grid(row=0, column=0, columnspan=2,
            padx=10, pady=10, sticky="nsew")
        self.conversation.insert(tk.END, "PsyExpert: Hello! I'm
            here to help you understand your emotions and
            personality traits.\n")
        self.conversation.insert(tk.END, "PsyExpert: Tell me
            about your feelings or thoughts today.\n")
        self.conversation.config(state=tk.DISABLED)

        # User input area
        self.user_input = tk.Entry(
            self.root, width=60, bg="white", fg="#333333", font=(
                "Arial", 10),
            relief=tk.SOLID, borderwidth=1)
        self.user_input.grid(row=1, column=0, padx=10, pady=10,
            sticky="ew")
        self.user_input.bind("<Return>", self.process_input)
        self.user_input.focus_set()

        # Send button
        self.send_button = tk.Button(
            self.root, text="Send", command=self.process_input,
                bg="#4a90e2", fg="white",
            font=("Arial", 10, "bold"), relief=tk.FLAT, padx=15)
        self.send_button.grid(row=1, column=1, padx=10, pady=10,
            sticky="e")

        # Analysis frame
        self.analysis_frame = tk.LabelFrame(
            self.root, text="Analysis Results", bg="#f0f0f0", fg=
                "#333333",
            font=("Arial", 10, "bold"))
        self.analysis_frame.grid(
            row=0, column=2, rowspan=2, padx=10, pady=10, sticky=
                "nsew")

        # Emotion label
        self.emotion_label = tk.Label(
            self.analysis_frame, text="Emotion: Not analyzed yet"
                ,
            bg="#f0f0f0", fg="#333333", font=("Arial", 10),
                wraplength=200, justify=tk.LEFT)
        self.emotion_label.pack(pady=5, padx=5, anchor="w")

        # Personality label
        self.personality_label = tk.Label(
            self.analysis_frame, text="Personality: Not analyzed
                yet",
```

```
            bg="#f0f0f0", fg="#333333", font=("Arial", 10),
               wraplength=200, justify=tk.LEFT)
        self.personality_label.pack(pady=5, padx=5, anchor="w")

        # Advice label
        self.advice_label = tk.Label(
            self.analysis_frame, text="Advice: Waiting for input"
               ,
            bg="#f0f0f0", fg="#333333", font=("Arial", 10),
               wraplength=200, justify=tk.LEFT)
        self.advice_label.pack(pady=5, padx=5, anchor="w")

        # Configure grid weights
        self.root.grid_columnconfigure(0, weight=3)
        self.root.grid_columnconfigure(1, weight=1)
        self.root.grid_columnconfigure(2, weight=2)
        self.root.grid_rowconfigure(0, weight=1)
        # Add MBTI assessment button
        self.assess_button = tk.Button(
            self.root, text="Assess MBTI", command=self.
               conduct_mbti_assessment,
            bg="#e67e22", fg="white", font=("Arial", 10, "bold"),
                relief=tk.FLAT)
        self.assess_button.grid(row=2, column=0, columnspan=2,
           pady=5, sticky="ew")

        # Configure grid weights
        self.root.grid_columnconfigure(0, weight=3)
        self.root.grid_columnconfigure(1, weight=1)
        self.root.grid_columnconfigure(2, weight=2)
        self.root.grid_rowconfigure(0, weight=1)
        self.root.grid_rowconfigure(1, weight=0)
        self.root.grid_rowconfigure(2, weight=0)

    def process_input(self, event=None):
        """Process user input and generate responses"""
        user_text = self.user_input.get().strip()
        if not user_text:
            return

        # Add user message to conversation
        self._update_conversation(f"You: {user_text}")

        # Analyze the input
        emotion = self.detect_emotion(user_text)

        # Use assessed MBTI if available, otherwise analyze from
           text
        if self.user_mbti:
            personality = self.user_mbti
            personality_source = "(assessed)"
```

```
        else:
            personality = self.analyze_personality(user_text)
            personality_source = "(analyzed)"

        advice = self.generate_advice(emotion, personality)

        # Update analysis labels
        self.emotion_label.config(text=f"Emotion: {emotion.
           capitalize()}")
        self.personality_label.config(text=f"Personality: {
           personality} {personality_source}")
        self.advice_label.config(text=f"Advice: {advice}")

        # Generate response
        response = f"PsyExpert: I sense you're feeling {emotion}.
            "

        if personality == "balanced":
            response += "Your personality traits seem fairly
               balanced. "
        else:
            response += f"Your responses suggest {personality}
               traits. "

        response += advice + "\n"
        self._update_conversation(response)

        # Clear input
        self.user_input.delete(0, tk.END)

#-----------------------------------------------
    def _update_conversation(self, text):
        """Update the conversation display"""
        self.conversation.config(state=tk.NORMAL)
        self.conversation.insert(tk.END, text + "\n")
        self.conversation.config(state=tk.DISABLED)
        self.conversation.see(tk.END)

    def detect_emotion(self, text):
        """Enhanced emotion detection with weighting and context
           """
        text = text.lower()
        emotion_scores = {emotion: 0 for emotion in self.
           emotion_keywords}

        # Check for negation patterns (e.g., "not happy")
        negations = re.compile(r'\b(not|never|no)\b[\w\s]+?\b(' +
            '|'.join(
            [kw for sublist in self.emotion_keywords.values() for
                kw in sublist]) + r')\b')
```

```
        # Find and remove negated emotions
        for match in negations.finditer(text):
            negated_emotion = next(
                (emotion for emotion, keywords in self.
                   emotion_keywords.items()
                 if match.group(2) in keywords), None)
            if negated_emotion:
                emotion_scores[negated_emotion] -= 2

        # Score positive matches
        for emotion, keywords in self.emotion_keywords.items():
            for keyword in keywords:
                # Higher score for exact matches
                if re.search(r'\b' + re.escape(keyword) + r'\b',
                   text):
                    emotion_scores[emotion] += 2
                # Lower score for partial matches
                elif keyword in text:
                    emotion_scores[emotion] += 1

        # Check for intensity modifiers
        intensifiers = ['very', 'really', 'extremely', '
           incredibly', 'terribly']
        for intensifier in intensifiers:
            if intensifier in text:
                following_word = re.search(r'\b' + intensifier +
                   r'\s+(\w+)', text)
                if following_word:
                    intensified_emotion = next(
                        (emotion for emotion, keywords in self.
                           emotion_keywords.items()
                         if following_word.group(1) in keywords),
                            None)
                    if intensified_emotion:
                        emotion_scores[intensified_emotion] += 1

        # Get top emotion
        top_emotion = max(emotion_scores.items(), key=lambda x: x
           [1])

        if top_emotion[1] <= 0:
            return "neutral"
        return top_emotion[0]

    def analyze_personality(self, text):
        """Enhanced personality analysis with context awareness
           """
        text = text.lower()
        personality = []
        threshold = 2  # Minimum score needed to register a trait
```

```
        for dimension, traits in self.mbti_rules.items():
            trait_scores = {trait[0]: 0 for trait in traits}

            for trait, keywords in traits:
                for keyword in keywords:
                    # Exact matches score higher
                    if re.search(r'\b' + re.escape(keyword) + r'\
                       b', text):
                        trait_scores[trait] += 2
                    # Partial matches score lower
                    elif keyword in text:
                        trait_scores[trait] += 1

            # Check for preference patterns (e.g., "I prefer
               being alone")
            preference_pattern = re.compile(
                r'\b(prefer|like|enjoy|rather|favor)\b[\w\s]+?\b(
                   ' +
                '|'.join([kw for trait in traits for kw in trait
                   [1]]) + r')\b')

            for match in preference_pattern.finditer(text):
                preferred_trait = next(
                    (trait[0] for trait in traits if match.group
                       (2) in trait[1]), None)
                if preferred_trait:
                    trait_scores[preferred_trait] += 2

            # Determine dominant trait
            max_score = max(trait_scores.values())
            if max_score < threshold:
                # Not enough evidence - make neutral choice
                personality.append('X')
            else:
                dominant_trait = max(trait_scores.items(), key=
                   lambda x: x[1])[0]
                personality.append(dominant_trait)

        # Convert to MBTI format (replace X with balanced
           indicator)
        mbti_type = []
        for i, trait in enumerate(personality):
            if trait == 'X':
                # For unclear traits, use midpoint indicator
                mbti_type.append('-')
            else:
                mbti_type.append(trait)

        return ''.join(mbti_type) if any(t != '-' for t in
           mbti_type) else "balanced"
```

```
    def generate_advice(self, emotion, personality):
        """Enhanced advice generation with personality-specific
           suggestions"""
        # Base advice selection
        emotion_advice = random.choice(self.advice_rules.get(
           emotion,
            ["Try reflecting on your current state and what might
                be influencing it."]))

        if not personality or personality == "balanced":
            return emotion_advice

        # Personality-specific additions
        advice_additions = []

        # Introvert/Extrovert
        if personality[0] == 'I':
            advice_additions.append(
                "As someone who seems introverted, you might
                   benefit from some quiet time to reflect.")
        elif personality[0] == 'E':
            advice_additions.append(
                "As someone who seems extroverted, discussing
                   this with someone you trust could be helpful."
                   )

        # Sensing/Intuition
        if personality[1] == 'S':
            advice_additions.append(
                "Your practical nature suggests focusing on
                   concrete steps might help you process this.")
        elif personality[1] == 'N':
            advice_additions.append(
                "Your intuitive side might appreciate exploring
                   the deeper
                   meaning behind this experience.")

        # Thinking/Feeling
        if personality[2] == 'T':
            advice_additions.append(
                "Your logical approach could help you analyze the
                    situation objectively.")
        elif personality[2] == 'F':
            advice_additions.append(
                "Your emotional awareness is a strength in
                   understanding these feelings.")

        # Judging/Perceiving
        if personality[3] == 'J':
            advice_additions.append(
```

```
                "Your structured nature might benefit from
                   creating a plan to address this.")
        elif personality[3] == 'P':
            advice_additions.append(
                "Your adaptability can help you remain open to
                   different ways of handling this.")

        # Combine all advice
        return f"{emotion_advice} {' '.join(advice_additions)}"

    def run(self):
        """Run the application"""
        self.root.mainloop()

# Create and run the application
if __name__ == "__main__":
    app = PsyExpert()
    app.run()
```

Appendix B: Sample Output

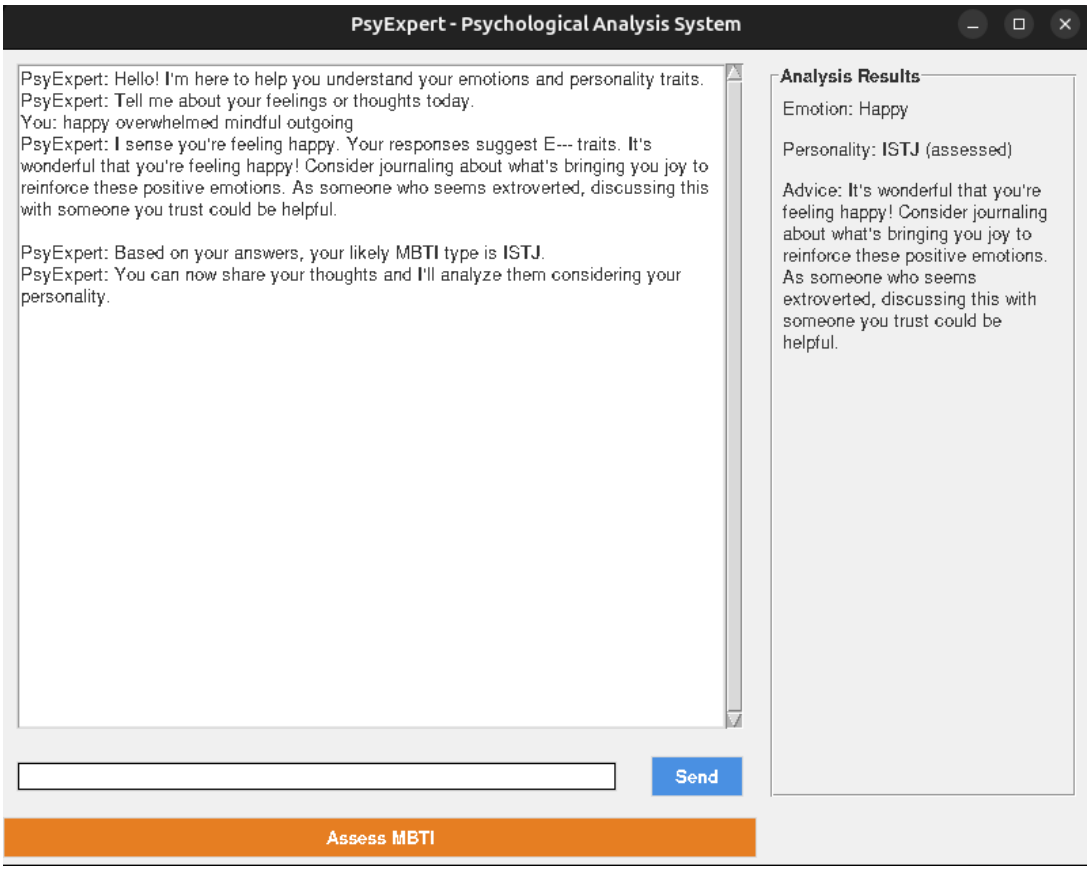

Figure 1: PsyExpert GUI showing emotion detection, personality analysis, and advice