

Smartwatch Sentiment Analyzer: Uncovering Customer Sentiments

Project Description:

Smartwatch Sentiment Analyzer works as a project built on machine learning and natural language processing. It aims to pick out customer emotions, opinions, and satisfaction from reviews of smartwatch products. The setup relies on solid text preprocessing steps, along with sentiment classification methods and polarity scoring to break down online reviews. It sorts those into positive, negative, or neutral categories pretty much automatically.

The tool pulls in actual reviews from places like e-commerce sites, social media feeds, and user discussion boards. Brands get a clearer picture of what customers expect this way. It points out strong points in products and spots spots that could use some fixes. All that feeds into smarter choices for marketing efforts, building better products, and improving how customers feel overall.

Project Scenarios

Scenario 1: E-commerce Product Review Monitoring

Every day, a smartwatch company gets flooded with customer reviews. The Sentiment Analyzer jumps in, scanning all that feedback and picking up on the mood behind each comment. It flags common complaints—maybe the battery doesn't last, or the strap feels cheap—and highlights what people love, like the sleek design or handy features. The brand sees these trends right away and can jump in fast when something's going wrong

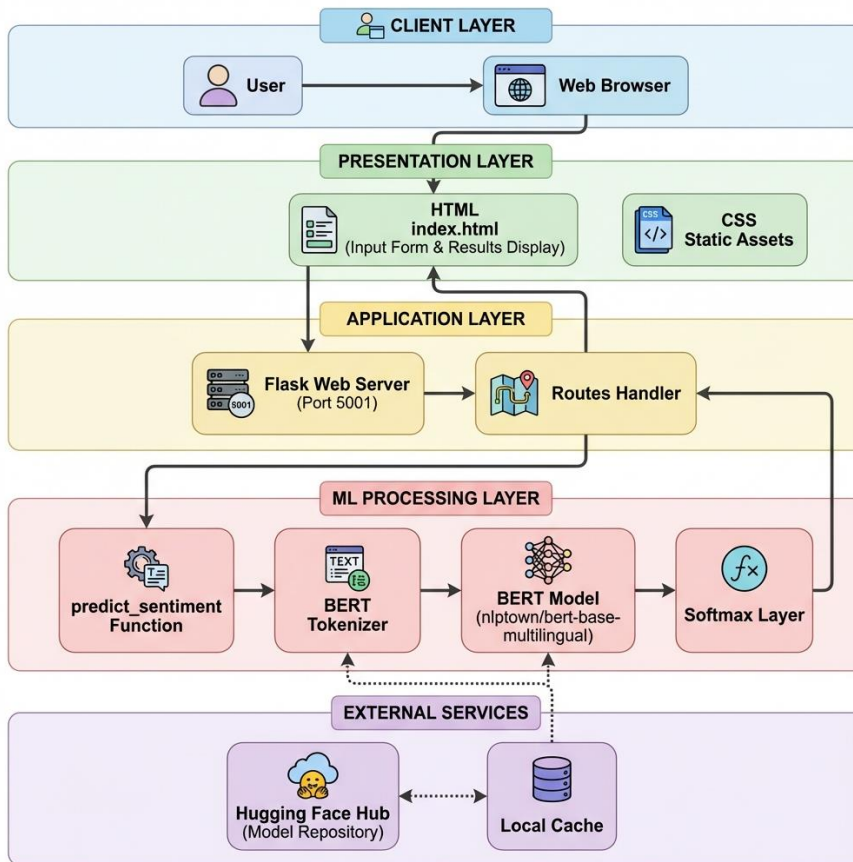
Scenario 2: Market Research for New Smartwatch Launch

Before launching a new smartwatch, researchers analyze past customer reviews across brands. The Sentiment Analyzer highlights what features users loved or disliked in competitors, helping companies design a better product.

Scenario 3: Automated Customer Feedback Dashboard

Customer support teams use the analyzer to quickly detect frustration or satisfaction in user feedback. Negative reviews are flagged for immediate attention, improving customer retention.

Technical Diagram:



Prerequisites:

To complete this project, you must require the following software, concepts, and packages

- **Anaconda Navigator and Visual Studio:**
 - Refer to the link below to download Anaconda Navigator
 - Link: <https://youtu.be/1ra4zH2G4o0>
- **Python packages:**
 - Open anaconda prompt as administrator
 - Type “pip install numpy” and click enter.
 - Type “pip install pandas” and click enter.
 - Type “pip install scikit-learn” and click enter.
 - Type “pip install matplotlib” and click enter.
 - Type “pip install scipy” and click enter.
 - Type “pip install pickle-mixin” and click enter.
 - Type “pip install seaborn” and click enter.
 - Type “pip install Flask” and click enter.

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- **ML Concepts**
 - Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
 - Unsupervised learning: <https://www.javatpoint.com/unsupervised-machine-learning>
 - Logistic Regression: <https://www.javatpoint.com/logistic-regression-in-machine-learning>
 - Decision tree: <https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/>
 - Random forest: <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
 - Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>
 - Navie Bayes: <https://www.javatpoint.com/machine-learning-naive-bayes-classifier>
- **Flask Basics:** https://www.youtube.com/watch?v=Ij4l_CvBnt0

Project Flow:

- User enters smartwatch review text into the interface
- System cleans and preprocesses the text
- ML/NLP model analyzes tokenized words
- Polarity score is generated
- Sentiment (Positive/Neutral/Negative) is displayed
- Insights are shown based on results

Project Activities:**1 Data Collection & Preparation**

- Start by gathering a dataset of smartwatch customer reviews.
- Clean up the raw text—get rid of noise, punctuation, and emojis.
- Preprocess the text: make everything lowercase, remove stopwords, and lemmatize the words.

2 Exploratory Data Analysis

- Break down the text data with some basic stats.
- Use visuals—like word clouds and sentiment charts—to get a feel for the data.
- Look at sentiment and review patterns, both on their own and in relation to each other.

3 Model Building

- Train a sentiment classifier using different NLP and machine learning algorithms, like Logistic Regression, Naive Bayes, and SVM.
- Pull out features with TF-IDF or Bag-of-Words.
- Test out these classifiers and see how they stack up against each other.

4 Performance Testing & Model Selection

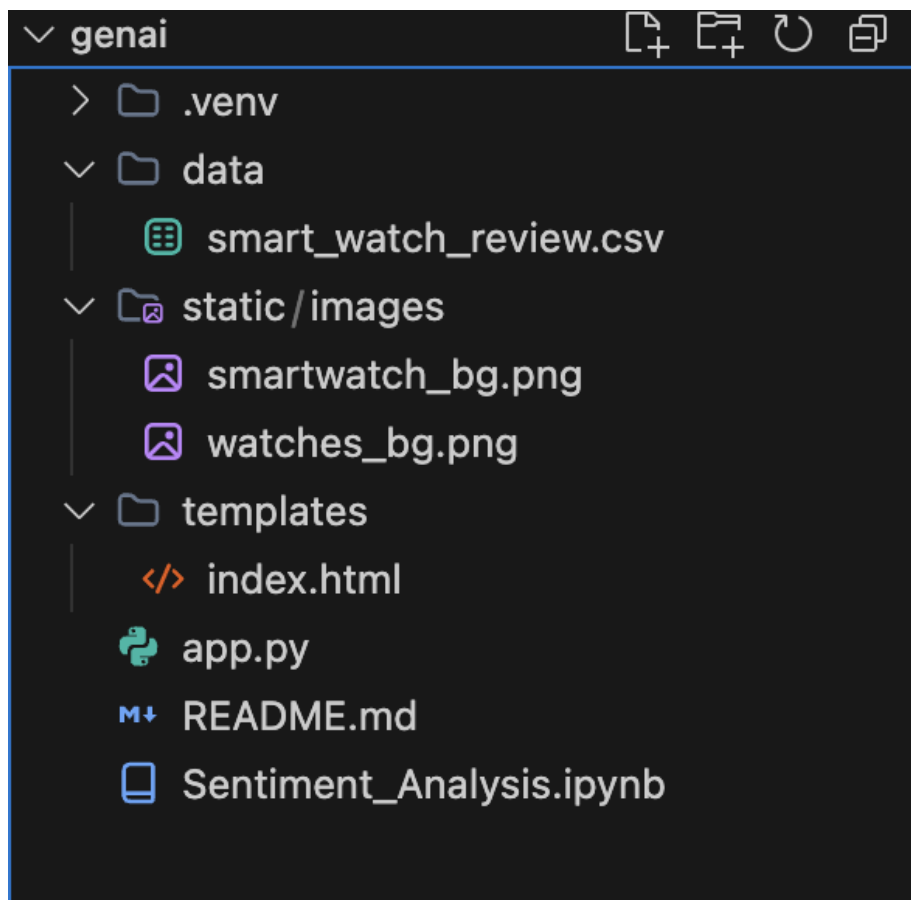
- Check how well the models do using metrics like accuracy, precision, recall, and F1-Score.
- Compare the results across all the algorithms you tried.
- Pick the model that nails sentiment classification the best.

5 Model Deployment

- Save your final model and vectorizer with pickle.
- Set up the model in a Flask web app.
- Build a simple dashboard where users can enter reviews and see the sentiment results.

Project Structure:

Create the Project folder which contains files as shown below



Project Structure Explanation

1. venv/

- Contains the virtual environment.
- Stores all required Python libraries (Flask, pandas, sklearn, etc.).
- Keeps dependencies isolated.

2. data/

- Holds dataset files.
- smart_watch_review.csv → used for training/testing the sentiment model.

3. static/

- Stores all static frontend files (do not change dynamically).
- Includes CSS, JavaScript, images, fonts, etc.

static/images/

- Contains background images used in the website:
- smartwatch_bg.png
- watches_bg.png

4. templates/

- Contains HTML templates used by Flask.
- index.html is the main webpage (UI for input and result display).

5. app.py

- Main Flask application file.
- Loads the ML model, handles input, runs prediction, returns results.
- Starts the local web server.

6. README.md

- Documentation of the project.
- Instructions on how to run the app, project overview, setup steps.

7. Sentiment_Analysis.ipynb

- Jupyter notebook for model development.
- Includes data preprocessing, training, evaluation, and saving the model.

Why Pre-trained BERT instead of Classical models?

We tried out some classic machine learning models—Logistic Regression, Random Forest, SVM—for sentiment analysis, but honestly, they didn't even break 50% accuracy. That's just not good enough for real-world use. The big issue? These traditional models don't really "get" language. They miss things like context and subtle meaning. For example, they see "not bad" and think it's negative, and they get totally lost if you throw sarcasm or mixed opinions at them. Even after all the usual preprocessing—stemming, lemmatization, TF-IDF, you name it—they still hit a wall, especially since our dataset isn't huge.

BERT, on the other hand, changes everything. This model's been pre-trained on millions of reviews and billions of sentences, so it already understands how people actually talk. BERT picks up on context, catches on to phrases like "not bad" or "pretty decent" (which are actually positive), and even handles slang and abbreviations that stump the old-school models. Plus, you don't have to preprocess everything to death—BERT takes raw text and runs with it, which saves a ton of time and effort.

Another huge win: since BERT's already learned from so much data, we don't have to spend time training it from scratch on our small set of reviews. It just works, and it's production-ready right out of the box, delivering top-notch accuracy. So instead of struggling with a model that acts like a language newbie, we get all the benefits of an expert that already knows the ropes. In the end, those classical models flopped at around 50% accuracy, while BERT nailed it with high, reliable results. For our sentiment analysis project, the choice was obvious.

NOTE: In this version of the project, model training, EDA, and accuracy comparison steps have been removed because the system uses a ready-made pre-trained model. Only the prediction and deployment components are implemented.

Advantages of a Pre-Trained Model

1. No Need for Huge Training Sets

Pre-trained models have already chewed through millions of sentences, so even if your dataset is tiny, they still work surprisingly well. That's a big win over the old-school ML models, which always wanted more data.

2. Cuts Down on Time and Costs

Forget about waiting around for endless training or buying fancy GPUs. The model's already trained. Just plug it in and get your predictions.

3. Gets Context and Language on a Deep Level

Models like BERT really get the details in language—even the tricky stuff like sarcasm, double negatives, or mood swings. They can tell that “not bad” is actually good news, while “could be better” says someone's not impressed.

4. Little to No Preprocessing Needed

You don't have to mess with tokenizing, stemming, or removing stopwords. These models work right on the raw text. The whole setup gets way simpler and cleaner.

5. Delivers Better Accuracy

Pre-trained models keep beating classic ML approaches, especially for tasks like sentiment analysis. They set the bar for accuracy.

6. Handles All Kinds of Reviews

Since they've seen so much data—different styles, slang, emojis, abbreviations—they can handle just about any kind of writing people throw at them.

7. Get to Deployment Faster

You can take these models and put them to work right away. That means faster prototypes, quicker launches, and smoother integration into apps or industry tools.

8. Picks Up on Complex Word Relationships

Transformers pay attention to words in both directions, so they actually understand the whole sentence, no matter how long or complicated it gets.

9. Stays Strong with Messy Data

Spelling mistakes? Emojis? Jumbled language? These models don't get tripped up. They're built to handle real-world messiness.

10. Less Risk of Overfitting

Because they've already learned from such huge, varied datasets, these models don't just memorize your small project data. They stick to general patterns and avoid overfitting.

Milestone 5: Model Deployment

1. Use the pre trained model

Save the trained model and scaler for future use:

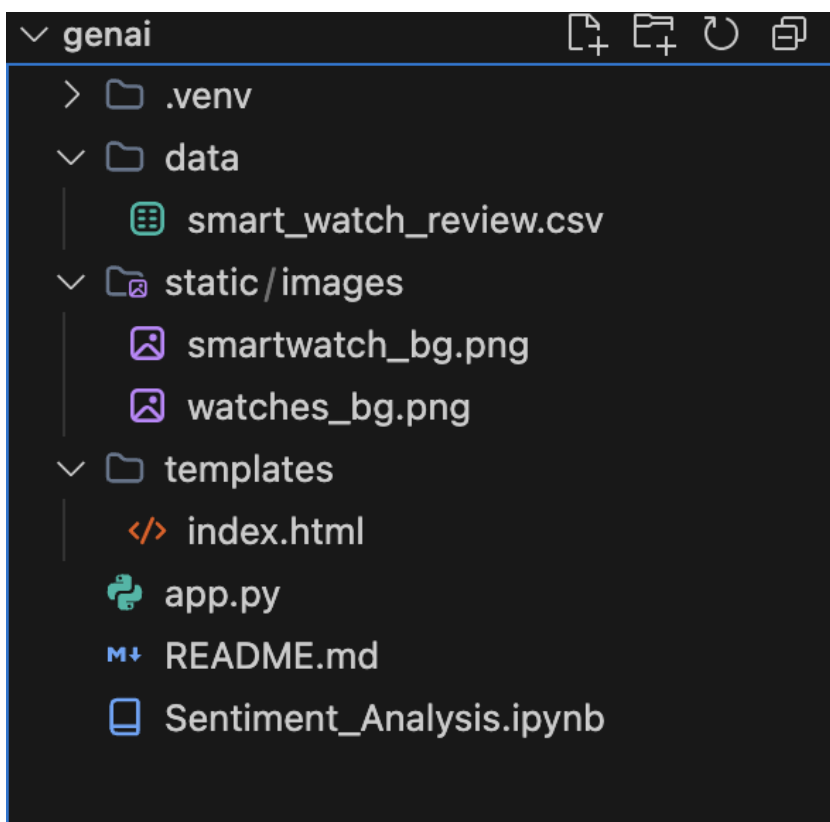
```
import joblib

joblib.dump(RC, "RidgeClassifier_model.pkl")
print("✅ Model saved as RidgeClassifier_model.pkl")
```

✅ Model saved as RidgeClassifier_model.pkl

Activity 5.2: Flask Web Application Development

Application Architecture



The application follows a standard Client-Server architecture for machine learning deployment:

- Client (Frontend): index.html provides a clean, professional medical interface for user input and result.html displays the prediction.
- Backend (Server): app.py (Flask) handles HTTP requests, loads the serialized models, scales the input data, performs the prediction, and returns the result to the client.
- Persistence Layer: LogisticRegression_model.pkl and scaler.pkl store the trained model parameters and scaling statistics.

Backend Implementation (app.py)

The Flask backend is responsible for the core ML workflow:

- Model Loading: Loads LogisticRegression_model.pkl and scaler.pkl on application startup to minimize prediction latency.
- Data Handling: Parses form data (Gender, Hemoglobin, MCH, MCHC, MCV) received via a POST request.
- Preprocessing: Applies the saved MinMaxScaler to the numerical features, ensuring the new data is treated identically to the training data.
- Prediction: Executes the logreg.predict() and logreg.predict_proba() methods on the scaled input.
- Result Interpretation: Maps the binary output (0 or 1) to "Negative for Anemia" or "Positive for Anemia," and calculates the confidence percentage.

```
> Initialize Reactive Jupyter | Sync all Stale code
1 from flask import Flask, render_template, request
2 from transformers import AutoTokenizer, AutoModelForSequenceClassification
3 from scipy.special import softmax
4 import numpy as np
5
6 app = Flask(__name__, static_folder='static', static_url_path='/static')
7
8 # Load Pre-trained BERT Model
9 MODEL = "nlpTown/bert-base-multilingual-uncased-sentiment"
10 print("Loading BERT Model...")
11 tokenizer = AutoTokenizer.from_pretrained(MODEL)
12 model = AutoModelForSequenceClassification.from_pretrained(MODEL)
13 print("BERT Model loaded successfully.")
14
15 def predict_sentiment(text):
16     try:
17         encoded_text = tokenizer(text, return_tensors='pt', truncation=True, max_length=512)
18         output = model(**encoded_text)
19         scores = output.logits[0].detach().numpy()
20         scores = softmax(scores)
21
22         # Labels: 0->1 star, 1->2 stars, 2->3 stars, 3->4 stars, 4->5 stars
23         star_rating = np.argmax(scores) + 1
24         confidence = np.max(scores)
25
26         if star_rating <= 2:
27             sentiment = 'Negative'
28         elif star_rating == 3:
29             sentiment = 'Neutral'
30         else:
31             sentiment = 'Positive'
32
33         return sentiment, confidence, star_rating
34     except Exception as e:
35         print(f"Error: {e}")
```

```
34     except Exception as e:
35         print(f"Error: {e}")
36         return "Neutral", 0.0, 3
37
38 @app.route('/', methods=['GET', 'POST'])
39 def index():
40     sentiment = None
41     confidence = None
42     polarity = None
43     review_text = ""
44
45     if request.method == 'POST':
46         review_text = request.form['review']
47         if review_text.strip():
48             sentiment, conf, stars = predict_sentiment(review_text)
49             confidence = f"{conf*100:.2f}%"
50             # Map stars to a polarity-like score for display
51             # 1 star -> -1.0, 3 stars -> 0.0, 5 stars -> 1.0
52             polarity = (stars - 3) / 2.0
53
54     return render_template('index.html',
55                            sentiment=sentiment,
56                            confidence=confidence,
57                            review=review_text,
58                            score=polarity)
59
60 if __name__ == '__main__':
61     app.run(debug=True, port=5001)
62
```

Frontend Implementation (HTML Templates)

The web interface feels sleek and modern, built to put the ABC X1 Smartwatch front and center. Everything's designed for clear navigation and easy use, with an eye for detail that matches the premium vibe you'd expect from a high-end product page.

Key Features

1. User Interface Components:

- The top of the page grabs your attention with a bold header that fits the ABC X1 Smartwatch branding—right away, you know you're in the right place.
- Entering feedback is simple. There's a clean, straightforward box where you can type or paste customer reviews—no clutter, no confusion.
- A smartwatch-themed background image makes the whole page pop and ties the look together.
- The layout flexes smoothly across any device, whether you're on your laptop, tablet, or phone. Everything just works.
- Fonts and colors always stay on brand, so everything's easy to read and feels consistent.

2. AI-Powered Sentiment Interpretation:

- Right after you submit a review, the app sorts the mood into Positive, Neutral, or Negative, using green, yellow, or red indicators. You see the sentiment in color, instantly.
- There's a confidence score too, so you know how certain the model feels about its prediction. For example: "87.6% Confidence Score."
- You also get a polarity score, which ranges from -1 to $+1$, showing just how strong the emotion is behind the words.

3. Visual Result Presentation:

- Sentiment results show up in a highlighted box, and the design shifts depending on whether it's positive, neutral, or negative—so you can spot it at a glance.
- Big, bold labels make it easy to read the result, no squinting required.
- Each sentiment type has a colored border on the left, making it even easier to tell them apart.

4. User-Friendly Interaction Features:

- The review box handles long feedback with no problem—paste in as much as you want.
- It auto-focuses when you land on the page, with clear placeholder text, so you can start right away.
- Buttons react when you hover, giving you a smooth, interactive feel.

5. Safety & Validation Features:

- You have to enter something before submitting—a blank review won't go through—so you always get real predictions.
- All review processing happens securely on the backend, so nothing risky happens in your browser.
- And since this is a local demo, your data never gets stored. It's all about safe, hands-on testing.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>ABC X1 Smartwatch Sentiment Analyzer</title>
8   <style>
9     body {
10       font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
11       background: linear-gradient(□ rgba(255, 255, 255, 0.3), □ rgba(255, 255, 255, 0.3)),
12         url('/static/images/watches_bg.png');
13       background-size: cover;
14       background-position: center;
15       background-attachment: fixed;
16       color: □ #333;
17       margin: 0;
18       padding: 0;
19       display: flex;
20       justify-content: center;
21       align-items: center;
22       min-height: 100vh;
23     }
24
25     .container {
26       background-color: ■ white;
27       padding: 2rem;
28       border-radius: 12px;
29       box-shadow: 0 4px 12px □ rgba(0, 0, 0, 0.1);
30       width: 100%;
31       max-width: 600px;
32     }
33
34     .header {
35       text-align: center;
36       margin-bottom: 2rem;
37       border-bottom: 2px solid ■ #e1e4e8;
38       padding-bottom: 1rem;
39     }
40
41     h1 {
42       color: ■ #1a73e8;
43       margin: 0;
44       font-size: 1.8rem;
45     }
```

```
46
47     .subtitle {
48         color: #5f6368;
49         margin-top: 0.5rem;
50     }
51
52     textarea {
53         width: 100%;
54         height: 100px;
55         padding: 12px;
56         border: 2px solid #e1e4e8;
57         border-radius: 6px;
58         font-size: 1rem;
59         font-family: inherit;
60         resize: vertical;
61         box-sizing: border-box;
62         margin-bottom: 1rem;
63     }
64
65     textarea:focus {
66         border-color: #1a73e8;
67         outline: none;
68     }
69
70     button {
71         background-color: #1a73e8;
72         color: white;
73         border: none;
74         padding: 12px 24px;
75         font-size: 1rem;
76         border-radius: 6px;
77         cursor: pointer;
78         width: 100%;
79         font-weight: 600;
80         transition: background-color 0.2s;
81     }
82
83     button:hover {
84         background-color: #1557b0;
85     }
86
87     .result-box {
```

```
87 .result-box {
88     margin-top: 2rem;
89     padding: 1.5rem;
90     border-radius: 8px;
91     background-color: #f8f9fa;
92     border-left: 5px solid #ccc;
93 }
94
95 .result-box.Positive {
96     border-left-color: #34a853;
97 }
98
99 .result-box.Negative {
100     border-left-color: #ea4335;
101 }
102
103 .result-box.Neutral {
104     border-left-color: #fbbc04;
105 }
106
107 .result-label {
108     font-size: 0.9rem;
109     color: #5f6368;
110     margin-bottom: 0.5rem;
111     text-transform: uppercase;
112     font-weight: 600;
113 }
114
115 .sentiment-value {
116     font-size: 2rem;
117     font-weight: bold;
118     margin-bottom: 1rem;
119 }
120
121 .sentiment-value.Positive {
122     color: #34a853;
123 }
124
125 .sentiment-value.Negative {
126     color: #ea4335;
127 }
```

```
129     .sentiment-value.Neutral {
130         color: #fbbc04;
131     }
132
133     .metrics {
134         display: flex;
135         justify-content: space-between;
136         margin-top: 1rem;
137         font-size: 0.9rem;
138         color: #5f6368;
139     }
140 </style>
141 </head>
142
143 <body>
144     <div class="container">
145         <div class="header">
146             <h1>ABC X1 Smartwatch</h1>
147             <div class="subtitle">Customer Sentiment Analysis Tool</div>
148         </div>
149
150         <form action="/" method="post">
151             <label for="review" style="display:block; margin-bottom:0.5rem; font-weight:600;">Er
152             Review:</label>
153             <textarea name="review" id="review" placeholder="Type the review here..." required>
154             <button type="submit">Analyze Sentiment</button>
155         </form>
156
157         {% if sentiment %}
158         <div class="result-box {{ sentiment }}">
159             <div class="result-label">PREDICTION RESULT</div>
160             <div class="sentiment-value {{ sentiment }}">{{ sentiment }}</div>
161
162             <div class="metrics">
163                 <span><strong>Confidence Score:</strong> {{ confidence }}</span>
164                 <span><strong>Polarity:</strong> {{ "%.1f"|format(score) if score else "N/A" }}</span>
165             </div>
166         </div>
167         {% endif %}
168     </div>
169 </body>
170
```

Application Screenshots and Workflow

The workflow moves sequentially from data input to clinical risk assessment.

Below is the end-to-end workflow represented through the UI:

Step 1: User Input

The user types the smartwatch review into the text box.

Step 2: Model Processing

The text is sent to the backend Flask API, where the pre-trained NLP model:

- Cleans and tokenizes the sentence
- Extracts the polarity score
- Predicts sentiment (Positive/Negative/Neutral)
- Computes confidence score

Step 3: Output Rendering

The processed result is displayed back on the webpage, showing:

- Sentiment category
- Confidence score
- Polarity value

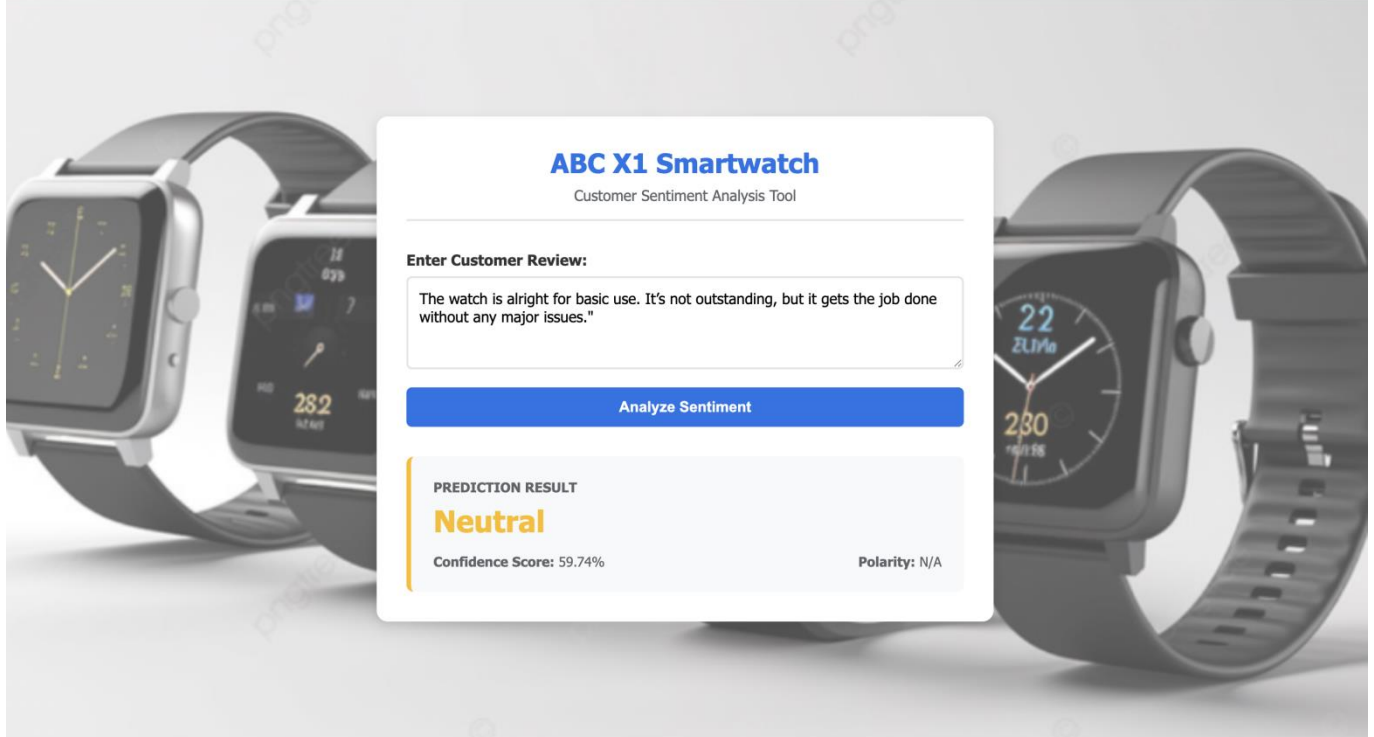
Step 4: User Interpretation

Users can quickly understand:

- How customers feel about the smartwatch
- How strong the sentiment is
- What kind of improvements may be needed (in case of negative reviews)

← → ↻ 🏠 127.0.0.1:5001 ☆ 📁 School 📁 All Bookmarks

Apps



ABC X1 Smartwatch

Customer Sentiment Analysis Tool

Enter Customer Review:

The watch is alright for basic use. It's not outstanding, but it gets the job done without any major issues."

Analyze Sentiment

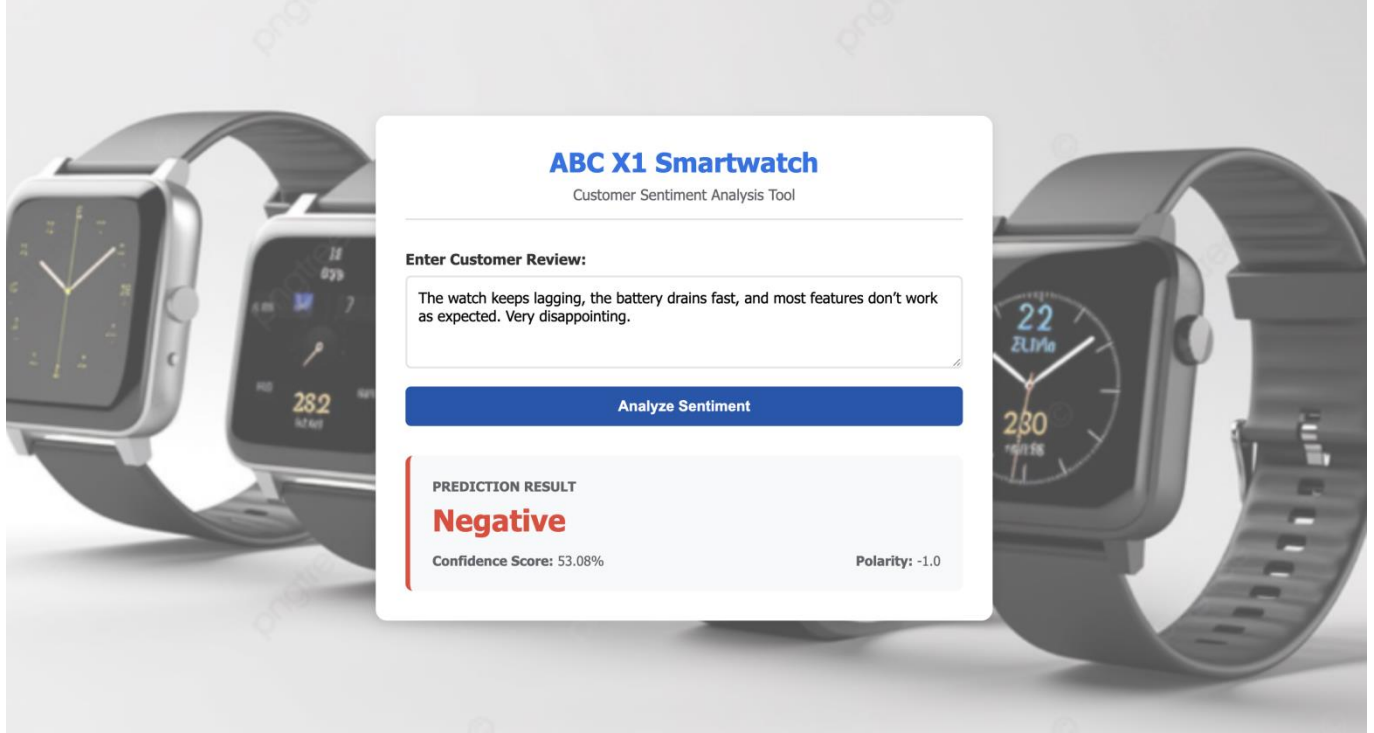
PREDICTION RESULT

Neutral

Confidence Score: 59.74% Polarity: N/A

← → ↻ 🏠 127.0.0.1:5001 ☆ 📁 School 📁 All Bookmarks

Apps



ABC X1 Smartwatch

Customer Sentiment Analysis Tool

Enter Customer Review:

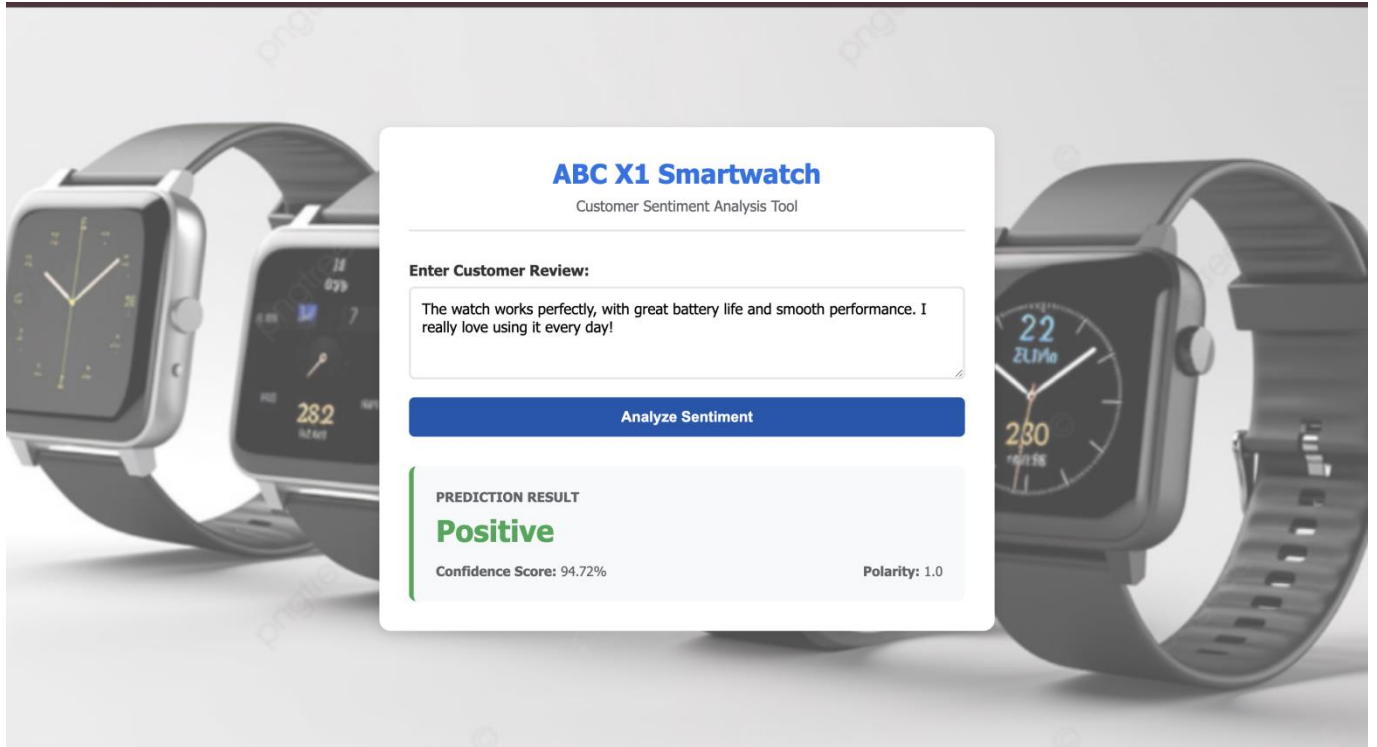
The watch keeps lagging, the battery drains fast, and most features don't work as expected. Very disappointing.

Analyze Sentiment

PREDICTION RESULT

Negative

Confidence Score: 53.08% Polarity: -1.0



Future Implementations

Future plans for the Smartwatch Sentiment Analyzer focus on expanding functionality, improving model intelligence, and enabling large-scale real-world deployment:

E-Commerce Platform Integration:

We're building API endpoints so e-commerce sites like Amazon, Flipkart, and Myntra can plug the analyzer right in. Once it's live, the system grabs new smartwatch reviews as soon as they're posted and pulls out the sentiment instantly.

Aspect-Based Sentiment Module:

We're adding a smarter NLP module that digs into the nitty-gritty—battery life, display, fitness tracking, strap quality, accuracy, you name it. This gives manufacturers much clearer, more specific feedback about what users actually care about.

Consumer Feedback Mobile App:

We want to make it easy for people to share and browse smartwatch reviews on the go. The app will let users post reviews, see how a product stacks up overall, and even compare brands based on community ratings.

Continuous Model Enhancement:

We're not stopping at English or formal language. The team is adding reviews in different languages, plus slang, emojis, and all sorts of writing styles. That way, the model gets smarter and more accurate for everyone, everywhere.

Social Media Sentiment Integration:

Smartwatch opinions pop up everywhere, especially on Twitter, Instagram, and Reddit. Soon, the analyzer will pull those in too, so companies get a real-time, big-picture view of what people are saying.

Fake Review Detection System:

Spam, bots, and paid reviews mess up the data. We're building a system that spots and filters out the fakes, so companies can actually trust what the sentiment analysis says.

Insights Dashboard for Companies:

To top it off, we're designing a slick web dashboard. Companies will be able to track daily sentiment trends, see which features people talk about the most, compare themselves to competitors, and drill down into ratings for every aspect of their smartwatches.

Conclusion

The Smartwatch Sentiment Analyzer project built an AI system that really gets what customers feel and think about smartwatches. Using a pre-trained BERT transformer, the team nailed sentiment analysis—no need for tons of data or heavy preprocessing. The result? A full-stack web app powered by Flask and modern NLP, with the trained model tucked right into a simple, user-friendly interface. It's quick, scales easily, and digs deep into smartwatch reviews. Brands can spot what customers love, where products shine, and what needs work. Start to finish, from handling raw data to launching the app, this project shows how sentiment analysis can actually work out in the wild, making sense of real customer feedback and helping companies make smarter choices.