To align the FCR score with the NPS score using the five-level sentiment scale (Very Negative to Very Positive), we can create a matrix that combines both metrics to categorize customer sentiment. Here's how it would work:

**Five-Level Sentiment Scale:**

1. **Very Negative**
2. **Negative**
3. **Neutral**
4. **Positive**
5. **Very Positive**

**Alignment Rules:**

**1. Detractors (NPS 0-6):**

* **Low FCR (0-49%):** **Very Negative Sentiment**
  + These customers are already dissatisfied (Detractors), and a low FCR score exacerbates their negative experience, leading to a very negative sentiment.
* **Moderate FCR (50-79%):** **Negative Sentiment**
  + While their issues might get resolved eventually, the fact that it required multiple contacts still leaves them with a negative impression.
* **High FCR (80-100%):** **Neutral Sentiment**
  + Even though they are Detractors, resolving their issue on the first contact might improve their perception slightly, leading to a neutral sentiment.

**2. Passives (NPS 7-8):**

* **Low FCR (0-49%):** **Negative Sentiment**
  + Passives are generally satisfied but not overly enthusiastic. If their issues require multiple contacts to resolve, it could push them toward a negative sentiment.
* **Moderate FCR (50-79%):** **Neutral Sentiment**
  + If their issues are resolved with some effort, their overall experience remains satisfactory but not exceptional, leading to a neutral sentiment.
* **High FCR (80-100%):** **Positive Sentiment**
  + A high FCR score for Passives can enhance their satisfaction level, moving their sentiment toward positive.

**3. Promoters (NPS 9-10):**

* **Low FCR (0-49%):** **Neutral Sentiment**
  + Promoters are highly satisfied overall, but a low FCR might slightly dampen their enthusiasm, bringing their sentiment to a neutral level.
* **Moderate FCR (50-79%):** **Positive Sentiment**
  + Even if the issue required some effort to resolve, Promoters are likely to remain positive due to their overall satisfaction with the service.
* **High FCR (80-100%):** **Very Positive Sentiment**
  + Promoters with a high FCR score will likely have a very positive sentiment, as their already high satisfaction is reinforced by an efficient and effective support experience.

**Summary of Sentiment Alignment:**

| **NPS Category** | **FCR Score** | **Sentiment** |
| --- | --- | --- |
| **Detractors** | **Low FCR** | Very Negative |
| **Detractors** | **Moderate FCR** | Negative |
| **Detractors** | **High FCR** | Neutral |
| **Passives** | **Low FCR** | Negative |
| **Passives** | **Moderate FCR** | Neutral |
| **Passives** | **High FCR** | Positive |
| **Promoters** | **Low FCR** | Neutral |
| **Promoters** | **Moderate FCR** | Positive |
| **Promoters** | **High FCR** | Very Positive |

**How to Use This Alignment:**

* **Customer Feedback Analysis:** After collecting NPS and FCR data, you can use this matrix to classify customers into the five sentiment categories.
* **Targeted Improvements:** Focus on improving FCR for Detractors and Passives to move them toward a more positive sentiment, potentially converting them into Promoters.
* **Customer Experience Strategy:** Use the combined insights from NPS and FCR to refine your customer service strategies, ensuring high FCR for all customer interactions to maximize positive sentiment.

This alignment allows for a nuanced understanding of customer sentiment, blending both overall satisfaction (NPS) and service efficiency (FCR) to provide actionable insights.

The implications of Net Promoter Score (NPS) and First Contact Resolution (FCR) scores in a business context are significant, as they directly impact customer satisfaction, loyalty, operational efficiency, and, ultimately, the company’s financial performance. Here’s how each score can affect the business:

**1. Net Promoter Score (NPS):**

**Implications:**

* **Customer Loyalty and Retention:**
  + **High NPS:** Indicates a strong base of loyal customers who are likely to continue using your products or services and recommend them to others. This can lead to increased customer retention and reduced churn rates.
  + **Low NPS:** Suggests that customers are dissatisfied and may be considering switching to competitors. This can result in higher churn rates and loss of market share.
* **Brand Reputation:**
  + **High NPS:** Promoters act as brand ambassadors, spreading positive word-of-mouth, which can enhance your brand's reputation and attract new customers.
  + **Low NPS:** Detractors may share their negative experiences with others, damaging your brand's reputation and deterring potential customers.
* **Revenue Growth:**
  + **High NPS:** A high proportion of Promoters can lead to organic growth through referrals, increasing revenue without significant additional marketing spend.
  + **Low NPS:** Poor customer satisfaction can lead to declining sales, increased acquisition costs (as you need to replace lost customers), and ultimately lower profitability.
* **Customer Insights:**
  + NPS provides valuable feedback on customer sentiment, helping you identify areas where your products, services, or customer experience might need improvement.

**2. First Contact Resolution (FCR):**

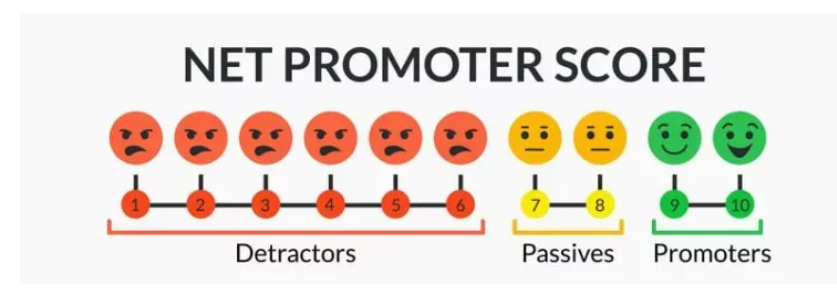
**Implications:**

* **Operational Efficiency:**
  + **High FCR:** Indicates that your customer support is efficient, resolving issues quickly without requiring multiple interactions. This can reduce the workload on support teams, lower operational costs, and improve service levels.
  + **Low FCR:** Requires more resources to handle repeated contacts, increasing operational costs and potentially overwhelming your support teams, leading to longer response times and lower overall service quality.
* **Customer Satisfaction and Experience:**
  + **High FCR:** Directly correlates with higher customer satisfaction, as customers value quick and effective problem resolution. This contributes to positive sentiment and can improve NPS.
  + **Low FCR:** Leads to frustration and dissatisfaction, as customers have to spend more time and effort to resolve their issues. This can negatively affect customer loyalty and satisfaction, potentially lowering NPS.
* **Cost Management:**
  + **High FCR:** By resolving issues on the first contact, businesses can reduce the cost per resolution, as fewer resources are needed to handle follow-up interactions. This leads to lower overall customer service costs.
  + **Low FCR:** Multiple contacts increase the cost of resolution and can lead to inefficiencies in the support process. This can also increase customer service overhead and reduce profitability.
* **Employee Productivity and Morale:**
  + **High FCR:** Support teams are likely to be more productive and satisfied when they can resolve issues effectively on the first contact, leading to better morale and reduced burnout.
  + **Low FCR:** Repeated interactions with dissatisfied customers can demoralize support staff, leading to lower productivity, higher turnover, and potentially impacting the quality of customer service.

**Combined Implications of NPS and FCR:**

* **Strategic Insights:** Together, NPS and FCR provide a comprehensive view of customer loyalty and service efficiency. High NPS combined with high FCR indicates a well-functioning business with strong customer relationships and operational efficiency. Conversely, low scores in both metrics may signal underlying issues that need to be addressed urgently.
* **Targeted Improvements:** Businesses can use insights from NPS and FCR to prioritize improvements in areas that directly impact customer satisfaction and loyalty. For example, if NPS is low but FCR is high, the issue might be with the product or service itself, not the customer support. Conversely, if FCR is low, enhancing support efficiency could lead to a significant boost in NPS.
* **Financial Performance:** Both metrics are closely tied to the financial health of a business. High NPS can lead to increased customer lifetime value, reduced churn, and higher revenues, while high FCR can reduce operational costs and improve profitability. Conversely, low scores can negatively impact revenue, increase costs, and harm long-term business sustainability.

In summary, NPS and FCR are critical indicators of a company’s customer-centric performance. Monitoring and improving these scores can lead to enhanced customer loyalty, better brand reputation, higher efficiency, and improved financial outcomes.

# Creating a new column with the NPS profile of each client, to facilitate calculating NPS  
def define\_profile(score):  
 if score <= 6:  
 return 'Detractor'  
 elif score <= 8:  
 return 'Passive'  
 else:  
 return 'Promoter'  
  
df['PROFILE'] = df['OVERALL\_RATING'].apply(define\_profile)

<https://www.sqmgroup.com/resources/library/blog/impact-fcr-nps>

To answer whether First Call Resolution (FCR) positively impacts transactional NPS, we surveyed customers who used a call center within one business day with leading North American organizations. [SQM Group's research](https://www.sqmgroup.com/research) shows that **every 1% improvement in First Call Resolution increases transactional NPS by 1.4 points** for the average call center. FCR and transactional NPS have a statistical correlation of .64, which would be considered high and shows that FCR positively impacts transactional NPS.

**Demystifying Neural Networks: Sentiment Analysis**

Understanding the Emotions



Image created with Bard

*This article is part of the series*[*Demystifying Neural Networks*](https://medium.com/@weidagang/demystifying-neural-networks-5edbd0a361c1)*.*

**Introduction**

In the era of data, understanding opinions, emotions, and sentiments expressed in text data has become crucial for businesses, policymakers, and researchers alike. Sentiment analysis, a subfield of natural language processing (NLP), offers a powerful lens to gauge public sentiment, analyze customer reviews, and monitor brand health in real-time. This blog post dives into the intricacies of sentiment analysis, explaining its workings and showcasing a hands-on example using the IMDB movie reviews dataset with Keras / TensorFlow.

**What is Sentiment Analysis?**

Sentiment analysis, sometimes referred to as opinion mining, is the computational process of identifying and categorizing opinions expressed in a piece of text, especially to determine whether the writer’s attitude towards a particular topic, product, or service is positive, negative, or neutral. At its core, sentiment analysis involves the application of machine learning techniques to text data to understand the underlying emotions.

**How Does Sentiment Analysis Work?**

In this article, we’ll build a deep learning model for sentiment analysis. Here’s a closer look at this model:

* Embedding Layer: This initial layer transforms words into dense vectors of fixed size, capturing the semantic meaning of each word. Unlike one-hot encoding, embeddings provide a more efficient and nuanced representation of words, allowing the model to understand similarities between words based on their context.
* GlobalAveragePooling1D Layer: Following the embedding layer, this layer reduces the dimensionality of the data by averaging over the sequence dimension. This step simplifies the model by condensing the information from each review into a fixed-size vector, facilitating the processing of variable-length text.
* Dense Layers: The model includes dense layers, where neurons are fully connected. The first dense layer uses ReLU (Rectified Linear Unit) activation for its ability to introduce non-linearity, allowing the model to learn complex relationships in the data. The final layer employs a sigmoid activation function, outputting a probability score between 0 and 1 that indicates the sentiment of the review.

This structure is adept at sentiment analysis because it combines the depth of understanding provided by the Embedding layer with the classification power of Dense layers, making it highly effective for interpreting the nuances of human language in text form.

**Example**

To put theory into practice, let’s analyze sentiments of movie reviews using the IMDB dataset with TensorFlow, a popular open-source library for machine learning and deep learning applications.

**The IMDB Movie Reviews Dataset**

The IMDB movie reviews dataset is a labeled dataset. It consists of 50,000 movie reviews from the Internet Movie Database (IMDB) split into two sets: 25,000 reviews for training and 25,000 reviews for testing. Each set contains an equal number of positive and negative reviews, making it a balanced dataset. The positive reviews are those with a sentiment score of 7 or higher (out of 10), and the negative reviews have a score of 4 or lower, with neutral reviews typically excluded. This labeling makes the IMDB dataset particularly useful for binary sentiment classification tasks, where the goal is to train a model to predict whether a given review expresses a positive or negative sentiment.

To give you a taste of what the IMDB dataset looks like, here are a few anonymized examples of movie reviews:

1. Positive Review: “An absolute masterpiece! The performances were top-notch, and the storyline was both engaging and thought-provoking. Definitely a must-watch.”
2. Negative Review: “Unfortunately, this movie failed to deliver on its promising premise. The plot was predictable, and the acting was lackluster. A disappointing experience.”

These examples illustrate the variance in sentiment and the subjective nature of movie reviews, highlighting the challenges and opportunities in analyzing such data.

**Building the Sentiment Analysis Model with Keras**

We use Keras to preprocess the text data, build a neural network model, and train it for sentiment classification. The code is available in this [colab notebook](https://colab.research.google.com/drive/1a4aq5F--x_tEDyddecXtJJRhQwjjt_DE?usp=sharing" \t "_blank):

from keras.preprocessing.sequence import pad\_sequences  
from keras.models import Sequential  
from keras.layers import Embedding, GlobalAveragePooling1D, Dense  
from keras.datasets import imdb  
import numpy as np  
  
# Constants for data preprocessing  
max\_length = 256 # Maximum length of the sequences  
padding\_type = 'post' # Padding type for sequences shorter than the maximum length  
vocab\_size = 10000 # Size of the vocabulary used in the Embedding layer  
  
# Load the IMDB dataset  
(train\_data, train\_labels), (test\_data, test\_labels) = imdb.load\_data(num\_words=vocab\_size)  
  
# Helper function to preprocess data  
def preprocess\_data(data):  
 return pad\_sequences(data, maxlen=max\_length, padding=padding\_type)  
  
# Preprocess the data  
train\_data = preprocess\_data(train\_data)  
test\_data = preprocess\_data(test\_data)  
  
# Define the model architecture  
def build\_model(vocab\_size, embedding\_dim=16, hidden\_units=16):  
 model = Sequential([  
 Embedding(vocab\_size, embedding\_dim, input\_length=max\_length),  
 GlobalAveragePooling1D(),  
 Dense(hidden\_units, activation='relu'),  
 Dense(1, activation='sigmoid')  
 ])  
 return model  
  
# Build and compile the model  
model = build\_model(vocab\_size)  
model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
model.summary()  
  
# Train and evaluate the model  
history = model.fit(train\_data, train\_labels, epochs=10, batch\_size=32, validation\_data=(test\_data, test\_labels), verbose=2)  
test\_loss, test\_acc = model.evaluate(test\_data, test\_labels, verbose=2)  
print(f"Test Accuracy: {test\_acc}, Test Loss: {test\_loss}")  
  
# Decode review function  
word\_index = imdb.get\_word\_index()  
def decode\_review(encoded\_review):  
 reverse\_word\_index = dict([(value, key) for (key, value) in word\_index.items()])  
 return ' '.join([reverse\_word\_index.get(i - 3, '?') for i in encoded\_review if i >= 3])  
  
# Display incorrect predictions  
def display\_incorrect\_predictions(test\_data, test\_labels, predictions, num\_examples=3):  
 predicted\_classes = (predictions > 0.5).astype(int)  
 incorrect\_indices = np.where(predicted\_classes.flatten() != test\_labels)[0]  
 for i, idx in enumerate(incorrect\_indices[:num\_examples]):  
 print(f"Incorrect Prediction {i+1}:")  
 print(f"Review: {decode\_review(test\_data[idx])}")  
 print(f"Actual Sentiment: {'Positive' if test\_labels[idx] == 1 else 'Negative'}")  
 print(f"Predicted Sentiment: {'Positive' if predicted\_classes[idx][0] == 1 else 'Negative'}")  
 print("--------------------------------------------------------------------------------\n")  
  
predictions = model.predict(test\_data)  
display\_incorrect\_predictions(test\_data, test\_labels, predictions)  
  
# Predict sentiments for sample reviews and display them  
def predict\_and\_display\_reviews(reviews):  
 sequences = [[word\_index.get(word, 2) for word in review.lower().split()] for review in reviews]  
 padded\_sequences = preprocess\_data(sequences)  
 sample\_predictions = model.predict(padded\_sequences)  
 sample\_predicted\_classes = (sample\_predictions > 0.5).astype(int)  
 for i, review in enumerate(reviews):  
 print(f"Review {i+1}: {review}")  
 print(f'Predicted Score: {sample\_predictions[i]}')  
 print(f"Predicted Sentiment: {'Positive' if sample\_predicted\_classes[i][0] == 1 else 'Negative'}")  
 print("--------------------------------------------------------------------------------\n")  
  
# Sample movie reviews  
reviews = [  
 "This movie was an excellent portrayal of character development and had stellar acting.",  
 "I found the movie to be predictable with a lackluster script.",  
 "The cinematography was magnificent, and the pacing was perfect. Highly recommend watching.",  
 "It was a terrible movie that wasted two hours of my life. The plot made no sense.",  
 "An absolute masterpiece, with a gripping story and profound performances."  
]  
  
predict\_and\_display\_reviews(reviews)

Output:

Test Accuracy: 0.8620399832725525, Test Loss: 0.45465412735939026  
782/782 [==============================] - 1s 2ms/step  
Incorrect Prediction 1:  
Review: i generally love this type of movie however this time i found myself wanting to kick the screen since i can't do that i will just complain about it this was absolutely idiotic the things that happen with the dead kids are very cool but the alive people are absolute idiots i am a grown man pretty big and i can defend myself well however i would not do half the stuff the little girl does in this movie also the mother in this movie is reckless with her children to the point of neglect i wish i wasn't so angry about her and her actions because i would have otherwise enjoyed the flick what a number she was take my advise and fast forward through everything you see her do until the end also is anyone else getting sick of watching movies that are filmed so dark anymore one can hardly see what is being filmed as an audience we are involved with the actions on the screen so then why the hell can't we have night vision  
Actual Sentiment: Negative  
Predicted Sentiment: Positive  
--------------------------------------------------------------------------------  
  
Incorrect Prediction 2:  
Review: hollywood had a long love affair with bogus nights tales but few of these products have stood the test of time the most memorable were the jon hall maria films which have long since become camp this one is filled with dubbed songs and slapstick it's a truly crop of corn and pretty near today it was nominated for its imaginative special effects which are almost in this day and age mainly of trick photography the only outstanding positive feature which survives is its beautiful color and clarity sad to say of the many films made in this genre few of them come up to alexander original thief of almost any other nights film is superior to this one though it's a loser  
Actual Sentiment: Negative  
Predicted Sentiment: Positive  
--------------------------------------------------------------------------------  
  
Incorrect Prediction 3:  
Review: ed mitchell is a teenager who lives for his job at good a small but friendly neighborhood stand while his buddy thompson also works there but lack single minded devotion to his job he's there because he accidentally destroyed the car of his teacher mr and has to raise money to pay the when a fast foot chain opens across the street it looks like good is history until ed a secret that brings hundreds of new customers to their door however the manager of kurt jan is determined to get his hands on the and put good out of business meanwhile ed and must rescue the world's oldest fast food employee from the demented hills asylum and ed might just find love with jackson if he could take his mind off the long enough to pay attention to her good is a comedy directed for kids decent story acting and overall a pretty harmless kids movie  
Actual Sentiment: Negative  
Predicted Sentiment: Positive  
--------------------------------------------------------------------------------  
  
1/1 [==============================] - 0s 19ms/step  
Review 1: This movie was an excellent portrayal of character development and had stellar acting.  
Predicted Score: [0.79064614]  
Predicted Sentiment: Positive  
--------------------------------------------------------------------------------  
  
Review 2: I found the movie to be predictable with a lackluster script.  
Predicted Score: [0.6444884]  
Predicted Sentiment: Positive  
--------------------------------------------------------------------------------  
  
Review 3: The cinematography was magnificent, and the pacing was perfect. Highly recommend watching.  
Predicted Score: [0.2016313]  
Predicted Sentiment: Negative

This code demonstrates how to preprocess data, define a neural network architecture, and use it for sentiment analysis.

**Conclusion**

Sentiment analysis is a powerful tool in the arsenal of data scientists, offering insights into the public’s perceptions and feelings. By leveraging Keras/TensorFlow and the IMDB dataset, we’ve shown how to build and train a model capable of classifying sentiments in movie reviews. The field of sentiment analysis is vast and continuously evolving, with new techniques and models emerging regularly. This example serves as a starting point, and there’s much more to explore and experiment with in this exciting domain.

**[Having Fun with TypeScript: Zod](https://medium.com/@weidagang/zod-schema-validation-made-easy-195f86d82d44?source=author_recirc-----7944e1e97e9d----1---------------------b80f02c9_3699_4ca9_99a7_19746027b872-------)**

[Schema Validation Made Easy](https://medium.com/@weidagang/zod-schema-validation-made-easy-195f86d82d44?source=author_recirc-----7944e1e97e9d----1---------------------b80f02c9_3699_4ca9_99a7_19746027b872-------)

**[Linux Beyond the Basics: How Linux Redirects I/O Streams](https://medium.com/@weidagang/linux-beyond-the-basics-i-o-redirection-d2362e2bed0b?source=author_recirc-----7944e1e97e9d----2---------------------b80f02c9_3699_4ca9_99a7_19746027b872-------)**

[File Descriptors and Tables](https://medium.com/@weidagang/linux-beyond-the-basics-i-o-redirection-d2362e2bed0b?source=author_recirc-----7944e1e97e9d----2---------------------b80f02c9_3699_4ca9_99a7_19746027b872-------)

**[Demystifying Neural Networks: Anomaly Detection with AutoEncoder](https://medium.com/@weidagang/demystifying-anomaly-detection-with-autoencoder-neural-networks-1e235840d879?source=author_recirc-----7944e1e97e9d----3---------------------b80f02c9_3699_4ca9_99a7_19746027b872-------)**

[This article is part of the series Demystifying Neural Networks.](https://medium.com/@weidagang/demystifying-anomaly-detection-with-autoencoder-neural-networks-1e235840d879?source=author_recirc-----7944e1e97e9d----3---------------------b80f02c9_3699_4ca9_99a7_19746027b872-------)

**[LLM Architectures Explained: NLP Fundamentals (Part 1)](https://medium.com/@vipra_singh/llm-architectures-explained-nlp-fundamentals-part-1-de5bf75e553a?source=read_next_recirc-----7944e1e97e9d----0---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)**

[Deep Dive into the architecture & building of real-world applications leveraging NLP Models starting from RNN to the Transformers.](https://medium.com/@vipra_singh/llm-architectures-explained-nlp-fundamentals-part-1-de5bf75e553a?source=read_next_recirc-----7944e1e97e9d----0---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)

**[Deep Learning Journey #1 → Simple Neural Network](https://medium.com/gitconnected/deep-learning-journey-1-simple-neural-network-141f269651d1?source=read_next_recirc-----7944e1e97e9d----1---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)**

[Let’s build a simple neural network with Python](https://medium.com/gitconnected/deep-learning-journey-1-simple-neural-network-141f269651d1?source=read_next_recirc-----7944e1e97e9d----1---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)

[Predictive Modeling w/ Python](https://medium.com/@ben.putney/list/predictive-modeling-w-python-e3668ea008e1?source=read_next_recirc-----7944e1e97e9d--------------------------------)

[20 stories·1454 saves](https://medium.com/@ben.putney/list/predictive-modeling-w-python-e3668ea008e1?source=read_next_recirc-----7944e1e97e9d--------------------------------)

[Practical Guides to Machine Learning](https://destingong.medium.com/list/practical-guides-to-machine-learning-a877c2a39884?source=read_next_recirc-----7944e1e97e9d--------------------------------)

[10 stories·1770 saves](https://destingong.medium.com/list/practical-guides-to-machine-learning-a877c2a39884?source=read_next_recirc-----7944e1e97e9d--------------------------------)

[Natural Language Processing](https://medium.com/@AMGAS14/list/natural-language-processing-0a856388a93a?source=read_next_recirc-----7944e1e97e9d--------------------------------)

[1651 stories·1226 saves](https://medium.com/@AMGAS14/list/natural-language-processing-0a856388a93a?source=read_next_recirc-----7944e1e97e9d--------------------------------)

[The New Chatbots: ChatGPT, Bard, and Beyond](https://medium.com/@MediumStaff/list/the-new-chatbots-chatgpt-bard-and-beyond-5969c7449b7f?source=read_next_recirc-----7944e1e97e9d--------------------------------)

**[LLM Fine-Tuning For Text Classification Using QLoRA](https://medium.com/@vmn11/llm-fine-tuning-for-text-classification-using-qlora-13a7d3a256f6?source=read_next_recirc-----7944e1e97e9d----0---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)**

[Large Language Models have been the hottest topic in the machine learning world for some time now. With ChatGPT reaching 100 millions users…](https://medium.com/@vmn11/llm-fine-tuning-for-text-classification-using-qlora-13a7d3a256f6?source=read_next_recirc-----7944e1e97e9d----0---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)

**[Social Media Sentiment Analysis](https://medium.com/@your-data-guy/social-media-sentiment-analysis-a8fcd1c9860e?source=read_next_recirc-----7944e1e97e9d----1---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)**

[Monitoring sentiment across social media platforms associated with a brand/product](https://medium.com/@your-data-guy/social-media-sentiment-analysis-a8fcd1c9860e?source=read_next_recirc-----7944e1e97e9d----1---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)

**[Sentiment Analysis using Deep Learning (BERT)](https://medium.com/@girish9851/sentiment-analysis-using-deep-learning-bert-adf975232da2?source=read_next_recirc-----7944e1e97e9d----2---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)**

[Sentiment analysis is one of the classic machine learning problems which finds use cases across industries. For example, it can help us in…](https://medium.com/@girish9851/sentiment-analysis-using-deep-learning-bert-adf975232da2?source=read_next_recirc-----7944e1e97e9d----2---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)

**[Building Sequential Models with RNN(Recurrent Neural Networks) Using Keras API](https://medium.com/@rakesh.ganya28/building-sequential-models-with-rnn-recurrent-neural-networks-using-keras-api-d32a83d7b6b5?source=read_next_recirc-----7944e1e97e9d----3---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)**

[In this blog, we will delve into the world of Sequential Data Modelling using Recurrent Neural Networks (RNN) with the Keras API. We will…](https://medium.com/@rakesh.ganya28/building-sequential-models-with-rnn-recurrent-neural-networks-using-keras-api-d32a83d7b6b5?source=read_next_recirc-----7944e1e97e9d----3---------------------559b3d9f_a212_48da_85ee_80d2ad51b712-------)

**Clustering sentence embeddings to identify intents in short text**

Hyperparameter tuning of UMAP + HDBSCAN to determine the number of clusters in unlabeled text data

([Clustering sentence embeddings to identify intents in short text | by David Borrelli | Towards Data Science](https://towardsdatascience.com/clustering-sentence-embeddings-to-identify-intents-in-short-text-48d22d3bf02e))

[David Borrelli](https://medium.com/@dborrelli3?source=post_page-----48d22d3bf02e--------------------------------)

Photo by [Mike Tinnion](https://unsplash.com/@m15ky?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

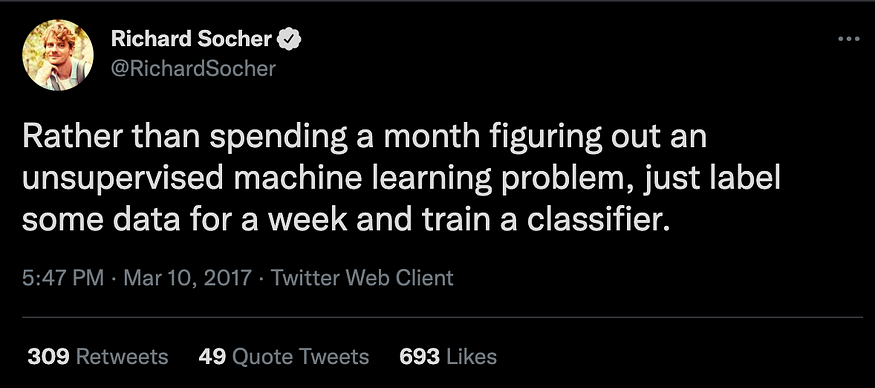
***TL;DR****The unsupervised learning problem of clustering short-text messages can be turned into a constrained optimization problem to automatically tune UMAP + HDBSCAN hyperparameters. The [chatintents](https://github.com/dborrelli/chat-intents" \t "_blank) package makes it easy to implement this tuning process.*

**Introduction**

User dialogue interactions can be a tremendous source of information on how to improve products or services. Understanding why people are reaching out to customer service is also an important first step in automating some or all of the replies (for example, with a chatbot). There are several ways to analyze dialogue interaction data to extract useful insights, and it is common to characterize the interactions by topics discussed, sentiment and intent.

Determining the intent is particularly useful from the perspective of wanting to improve products or services because it answers the question: why are people reaching out to begin with? However, a major hurdle to leveraging user message intent is that determining it is typically treated as a [classification problem](https://rasa.com/blog/rasa-nlu-in-depth-part-1-intent-classification/). This means you usually need to already have a significant amount of labeled data to get started. For example, [Microsoft’s LUIS](https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-intent) and [Google’s Dialogflow](https://cloud.google.com/dialogflow/es/docs/intents-overview) both start with the premise that you can either use prebuilt domain labeled data or that you already have labeled data.

But what if you don’t yet have any labeled data and you don’t think any publicly-available labeled data is relevant (as is often the case)? On top of the challenge of having an unsupervised learning problem, the messages containing the intent are typically quite short (less than 15 words). I was recently tasked with this challenge, with one additional hurdle: we only had about 1,000 samples total. I immediately remembered some sage advice I had seen years ago:



Source: <https://twitter.com/RichardSocher/status/840333380130553856>

After thinking about my particular problem for a bit and making a few unfruitful attempts at treating this as an unsupervised learning problem, I ultimately manually labeled the data (it took about a week…). Doing the labeling manually gave me a helpful appreciation and intuition for the data. But at the same time it made me very curious to see if there is a way to get most of the way to those labeled intents in a much more automated way. This post will provide an approach I learned that can automatically cluster short-text message data to identify and extract intents.

**Defining the goal**

Before we go further, let’s first define what we’re trying to do. Here I’m interested in answering the question:

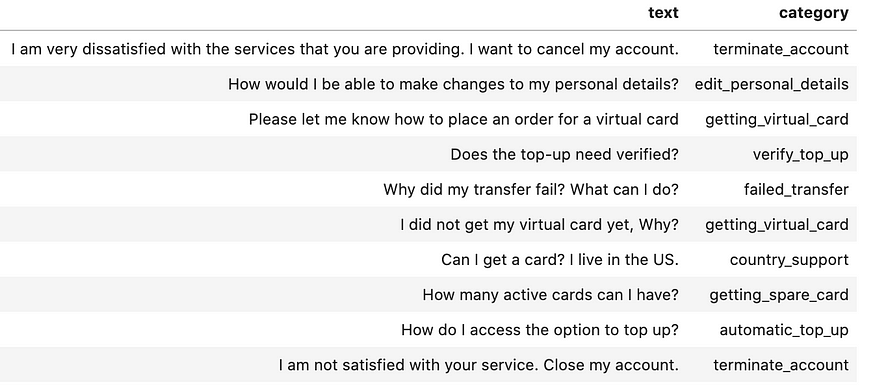
*given an unlabeled set of dialogues between a user and some company representative, is there a way to obtain a helpful labeling of user intents in an automated way?*

As this is an unsupervised problem and labeling intents can be quite subjective, I wouldn’t expect to be able to find a perfect solution. But, similar to how [auto-EDA libraries](https://github.com/pandas-profiling/pandas-profiling) aren’t exhaustive but can provide a helpful starting point when confronted with new data, can we do something to provide initial insights before committing to time-consuming manual labeling? It’s possible that the automated results could be good enough, saving someone from a week or more of manually labeling data. Alternatively, it could speed up the labeling process by providing a helpful starting point.

**Data**

Obviously, I’m not able to share the original dataset that inspired this article, so I set out to find something as similar as I could that is publicly available. While several dialogue datasets exist that have labeled intents, a major limitation in many of them is the small number of intents represented (often ~10). Having a small number of intents, or classes, will make the problem too simple. Although the manual labeling process is inherently subjective, I found there to easily be more than 50 intents, unequally represented, for the data I was working with. That seems to be fairly common for real-world applications.

Thankfully the PolyAI team published the [banking77 dataset](https://github.com/PolyAI-LDN/task-specific-datasets), which contains 77 intents represented unequally:



Sample data from bank77 dataset. Image by the author.

The full dataset contains 10,0003 messages in the training dataset across 77 intents. The maximum and minimum category counts are 187 and 35, respectively.

In order to more closely match the challenge I previously faced, I’ll randomly sample only 1,000 of the 10,000 total samples from the training set:

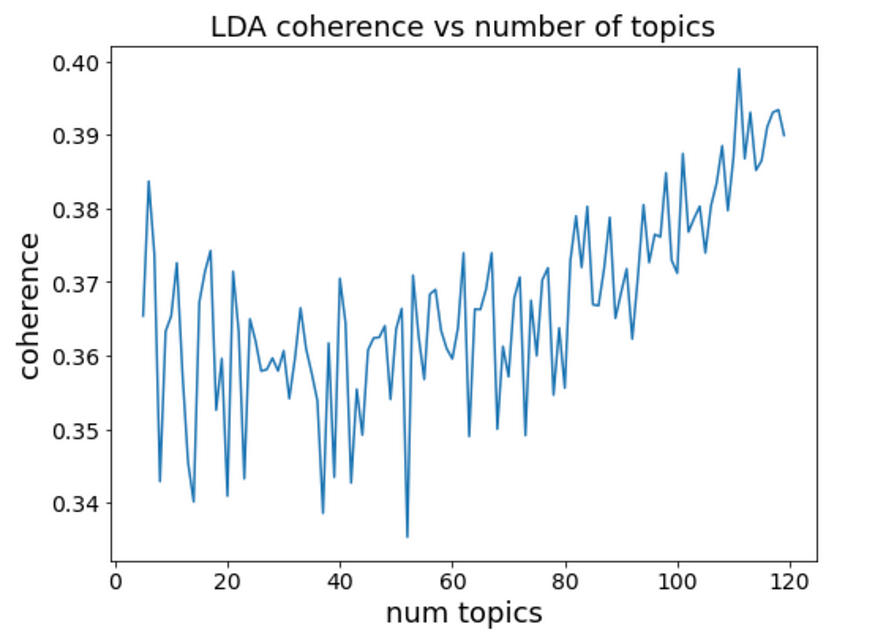
data\_sample = data\_full.sample(1000, random\_state=123)

Note that while this dataset is useful for this exercise for demonstration purposes, this is still somewhat artificial and you’ll face additional challenges in a real-world setting. For example, you’ll first need to identify what message or sentence in a full dialogue sequence actually relates to the intent, as well as handle random system error messages, typos and nonsensical messages. [This post](https://medium.com/airbnb-engineering/discovering-and-classifying-in-app-message-intent-at-airbnb-6a55f5400a0c) on discovering and classifying AirBnB message intents touches on some of the real-world challenges.

**Attempting topic modeling**

There are several ways to approach an unsupervised learning problem like this. [Topic modeling](https://en.wikipedia.org/wiki/Topic_model) was the first method that came to mind when confronted with this problem. It’s a technique used to discover latent topics in a collection of documents.

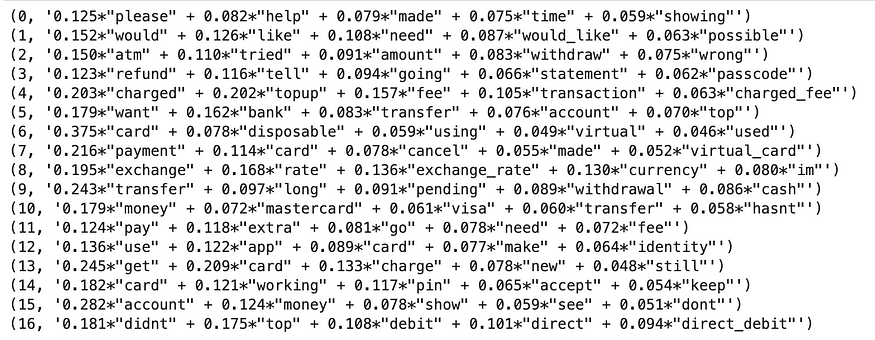
Many algorithms can be used to perform topic modeling, but one very common one is [Latent Dirichlet Allocation (LDA)](https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf). LDA is a generative probabilistic model that assumes that each document is made up of a distribution of a fixed number of topics and each topic is made up of a distribution of words. A big challenge when trying to use LDA (and many other topic modeling algorithms) is deciding how many topics to actually use, which is a necessary model hyperparameter. Obviously, if that is what we’re hoping to get out of the analysis then this is a problem. [Coherence](http://svn.aksw.org/papers/2015/WSDM_Topic_Evaluation/public.pdf) is one way to assess the quality of the learned topics by measuring how similar the words are in each topic, and a higher coherence score is better. [Gensim](https://radimrehurek.com/gensim/index.html" \t "_blank), a very popular library for topic modeling, makes it easy to calculate [model coherence](https://radimrehurek.com/gensim/models/coherencemodel.html). Unfortunately, for the short texts we’re working with here, it isn’t obvious how many topics to pick using coherence:



LDA coherence as a function of number of topics applied to a sample of the bank77 dataset. Image by the author.

It appears as though increasing the number of topics continues to increase the coherence for this dataset, giving us little guidance as to how many topics to choose.

Adding to the challenges, topic models can be hard to interpret. For example, consider the identified topics below:



LDA topics extracted from a sample of the bank77 dataset. Image by the author.

While some of the topics make sense, many are hard to interpret. [This article series](https://medium.com/pew-research-center-decoded/making-sense-of-topic-models-953a5e42854e) about work at Pew Research does a great job walking through the challenges of interpreting topic models.

Ultimately, the biggest issue is that intents are more nuanced than topics. A limitation in LDA and other topic modeling approaches is that they treat the vocabulary in the documents as a bag of words, where the order doesn’t matter. This works well for longer documents (and a larger corpus), where identifying words that co-occur can provide a good picture of the topics. Additionally, there are often a relatively small number of topics, and the topics are fairly distinct. However, short-text intents create challenges, such as two phrases having nearly identical words but very different intents or having the same intent but almost no words in common. This severely limits the usefulness of standard topic modeling approaches for identifying intents in short text.

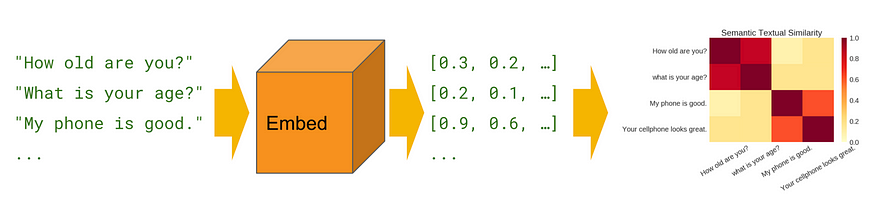
**Clustering embeddings**

Aside from topic modeling, clustering is another very common approach to unsupervised learning problems. In order to be able to cluster text data, we’ll need to make multiple decisions, including how to process the data and what algorithms to use.

**Selecting embeddings**

First, it is necessary to represent our text data numerically. One approach is to create embeddings, or vector representations, of each word to use for the clustering. [This article](https://towardsdatascience.com/introduction-to-word-embeddings-4cf857b12edc) gives a good overview of various ways of embedding words. Since each message consists of several words, one option is to then simply average the individual word embeddings of all the words in each message. This has worked well enough for some applications, but it would be better to directly calculate an embedding for the full sentences to more effectively take meaning into account. Especially given how short each message is, this will help avoid some of the pitfalls of the topic modeling algorithms described above.

It turns out that there are many ways to find a single vector representation of a full message or sentence. [This article](https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d) gives a great overview of the various methods to achieve this. Google’s [Universal Sentence Encoder](https://tfhub.dev/google/universal-sentence-encoder/4) (USE), first published by [Cer et al](https://arxiv.org/abs/1803.11175" \t "_blank) in 2018, is a popular sentence embedding model. The USE model was trained on a variety of data, including Wikipedia, web news, web question-answer pages and discussion forums, and it performs well on sentence semantic similarity tasks.



Universal Sentence Encoder model. Source: TensorFlow Hub <https://tfhub.dev/google/universal-sentence-encoder/4>

In 2019, Reimers and Gurevych published a [paper](https://arxiv.org/pdf/1908.10084.pdf) introducing Sentence-BERT, a “modification of the pretrained BERT network that use siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity”. They also released a [Python implementation](https://github.com/UKPLab/sentence-transformers) that makes it easy to download and use many different [pre-trained models](https://www.sbert.net/docs/pretrained_models.html).

Given how small our dataset is, using a pre-trained model is preferable here. For this analysis, I’ll compare the results of four pre-trained sentence embedding models: USE and three different sentence-BERT models (**all-mpnet-base-v2**, **all-MiniLM-L6-v2** and **all-distilroberta-v1**).

Converting our messages into sentence embeddings is then straightforward:



Embedding shapes using various pre-trained embedding models. Image by the author.

**Dimensionality Reduction**

As seen above, all of our sentence embeddings have high dimensionality (>500 features each). A manifestation of the Curse of Dimensionality is that distance measures, such as Euclidean and Manhattan, needed for clustering become meaningless at such high dimensions (see for example “[On the Surprising Behavior of Distance Metrics in High Dimensional Space](https://bib.dbvis.de/uploadedFiles/155.pdf)” by Aggarwal et al for more details). While some of the sentence-transformer pre-trained models were created in a way to preserve the usefulness of some distance measures, dimensionality reduction before clustering will greatly improve the results. (To prove this to myself, I [briefly explored](https://github.com/dborrelli/chat-intents/blob/main/notebooks/03-clustering_without_dim_reduction.ipynb) using different clustering algorithms and embeddings without dimensionality reduction first.)

Uniform Manifold Approximation and Projection for Dimension Reduction ([UMAP](https://umap-learn.readthedocs.io/en/latest/)), introduced by [McInnes et al](https://arxiv.org/pdf/1802.03426.pdf) in 2020, has quickly grown in popularity as a dimensionality reduction technique. UMAP is much faster and more scalable than t-SNE, while also preserving the global structure of the data much better. This makes it useful for both visualization and as a preprocessing dimensionality reduction step to use before clustering. We’ll use it here for both.

**Selecting a clustering algorithm**

The Scikit-learn documentation has a [helpful overview](https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods) of the many different clustering algorithms it supports and when each performs best. For our current application, it is preferable to use an algorithm that does not require specifying the number of clusters upfront and can also tolerate noisy data. Density-based algorithms are a good option here as they do not require specifying the number of clusters and are indifferent to cluster shape. Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) has become popular since it has fewer and more intuitive hyperparameters than DBSCAN and is robust to variable-density clusters. The HDBSCAN documentation provides a [helpful comparison](https://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html#comparing-python-clustering-algorithms) of different clustering algorithms. HDBSCAN worked best for the current problem, so we’ll focus on it for this post.

**Generating clusters from UMAP + HDBSCAN**

There are at least two packages (and likely many more) already available to chain UMAP and HDBSCAN together for the purposes of topic modeling: Top2Vec ([github](https://github.com/ddangelov/Top2Vec" \t "_blank) and [paper](https://arxiv.org/pdf/2008.09470.pdf)) and BERTopic ([github](https://github.com/MaartenGr/BERTopic" \t "_blank) and [article](https://towardsdatascience.com/topic-modeling-with-bert-779f7db187e6)). However, the default hyperparameters used in both packages do not work well for problems like the current one with short text and a small corpus (most of the data ends up being classified as noise and only three clusters total are found). To make it easier to tailor to our current problem of intent extraction, we’ll instead directly use the UMAP and HDBSCAN packages for hyperparameter tuning.

UMAP has several hyperparameters that control how it performs dimensionality reduction, but two of the most important are n\_neighbors and n\_components. The n\_neighbors parameter controls how UMAP balances [local versus global structure](https://umap-learn.readthedocs.io/en/latest/parameters.html#n-neighbors) in the data. This parameter controls the size of the neighborhood UMAP looks to learn the manifold structure, and so lower values of n\_neighbors will focus more on the very local structure. The n\_components parameter controls the dimensionality of the final embedded data after performing dimensionality reduction on the input data. Unfortunately, there is [no obvious way to pick](https://github.com/lmcinnes/umap/issues/182) the best UMAP parameters by itself without ground truth labels. Here we do have the labels, which we’ll use at the end to determine how well we did. But the point of this work is to identify a methodology to use when we have unlabeled data. In Angelov’s Top2Vec paper he mentions that n\_neighbors = 15 and n\_components = 5 worked best for his downstream tasks, but it is unlikely this would always be the case for any dataset.

HDBSCAN also has several important hyperparameters, but the [most important one](https://hdbscan.readthedocs.io/en/latest/parameter_selection.html#selecting-min-cluster-size) to consider is min\_cluster\_size. Intuitively, this controls the smallest grouping you want to consider as a cluster. In addition, the min\_samples parameter, which defaults to being equal to min\_cluster\_size if unspecified, controls how conservative the clustering is. The larger it is, the more points are discarded as noise/outliers. Decoupling the two hyperparameters and having a smaller min\_samples than min\_cluster\_size will essentially keep points that would have been labeled as outliers by merging them with their most similar neighboring clusters. This isn’t exactly what we want to happen if we’re trying to uncover the number of clusters. Thus, here I’ll only consider directly modifying the min\_cluster\_size parameter:

Note that UMAP is a [stochastic algorithm](https://umap-learn.readthedocs.io/en/latest/reproducibility.html), using randomness to speed up approximation steps and perform the optimization. Thus, we’ll set the random seed state to a constant value to get consistent results for a given set of UMAP hyperparameters.

**Defining the scoring function**

We now have a pipeline with three hyperparameters (n\_neighbors, n\_components, and min\_cluster\_size) that we want to tune. Next, we need to decide how to actually evaluate our clusters to select the best hyperparameters. Although commonly used with various clustering algorithms, Silhouette Score is not a good validation metric for density-based algorithms like DBSCAN and HDBSCAN since it assumes all points are assigned a group and can’t appropriately handle noise/outliers. [Density Based Cluster Validation (DBCV)](https://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.96) has been proposed and used by some for tuning HDBSCAN hyperparameters. While it likely works well for several applications, for this current problem it favored having a smaller number of clusters and leaving too many samples in the “noise” category.

Instead, we’ll leverage the useful probabilities\_ HDBSCAN attribute, which from the documentation is:

*The strength with which each sample is a member of its assigned cluster. Noise points have probability zero; points in clusters have values assigned proportional to the degree that they persist as part of the cluster.*

[This article](https://towardsdatascience.com/how-to-cluster-in-high-dimensions-4ef693bacc6) by Nikolay Oskolkov provides a very intuitive and logical solution of simply defining our cost function that we want to minimize as:

Cost = percent of dataset with < 5% cluster label confidence

This will help ensure that we assign as many data points as we can to actual clusters instead of labeling them as noise. But what’s stopping us from setting the hyperparameters to make every individual point a “cluster”, or just making one giant cluster?

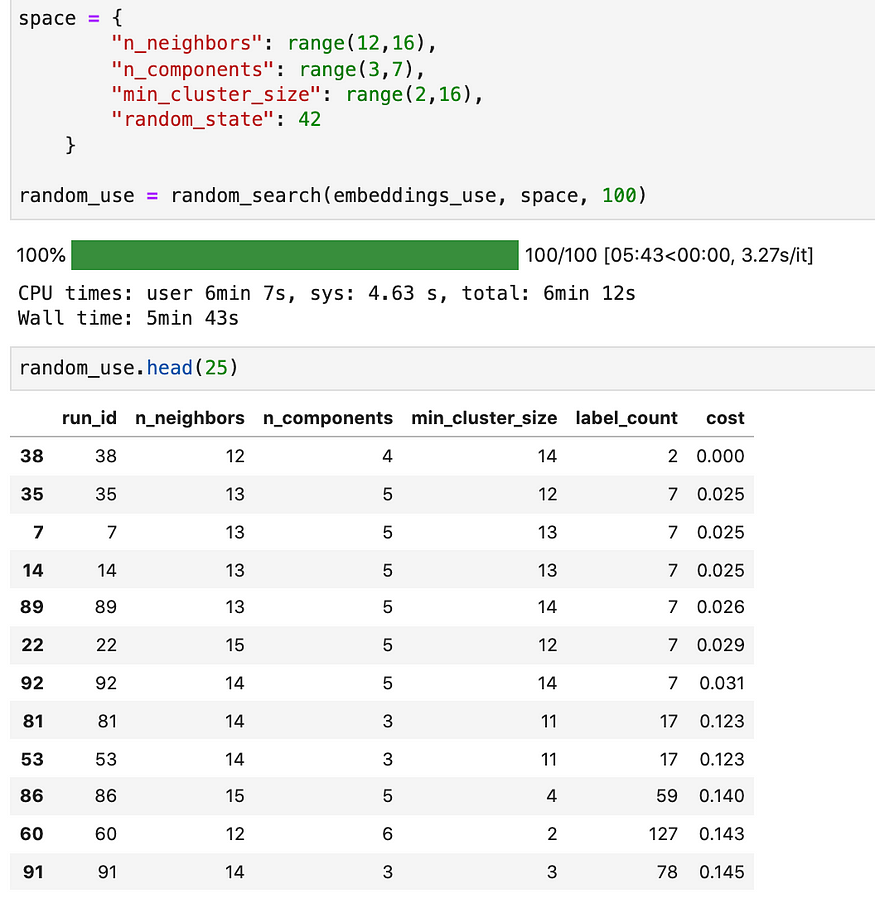
**Here we have to use some domain knowledge to apply constraints.** For this problem, based on my experience with this kind of data, I expected there to be at least 30 labels, but no more than 100. So our objective function becomes a constrained optimization problem:

minimize(Cost = percent of dataset with < 5% cluster label   
 confidence)Subject to: 30 < num\_clusters < 100

**Random hyperparameter search**

With the current dataset size of only 1,000 samples, it still takes about 3 seconds to generate the clusters and score them for a given set of inputs. Attempting to do a full grid search of, for example, a 10 x 10 x 10 hyperparameter search space would take almost an hour. Larger dataset sizes would take even longer. I care about finding the right hyperparameters, but not *that* much. Performing a random search instead of a full grid search is a pretty effective alternative strategy:

Running the search for 100 randomly-selected hyperparameter values yields:



Results of randomly searching hyperparameter space for UMAP+HDBSCAN. Image by the author.

We see that the runs with the lowest cost also have less than 10 clusters total. The first entry with a number of clusters between 30 and 100 has 59 clusters and a cost of 0.140 (i.e. about 14% of the data was labeled as outliers or low confidence). It only took 6 minutes to run too. Not bad.

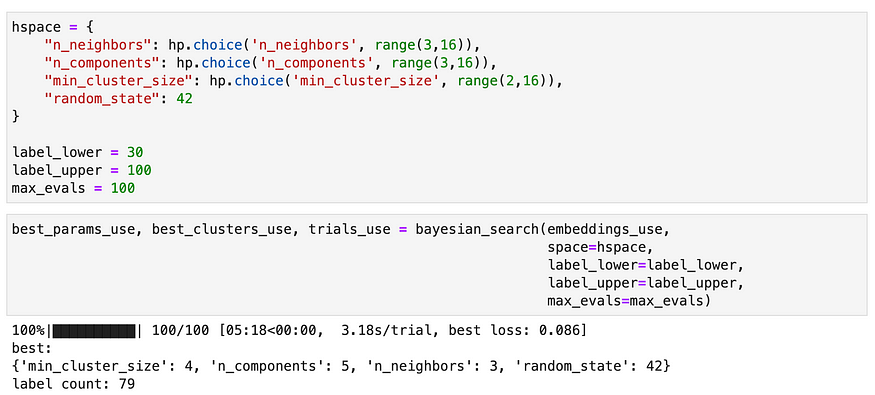
**Bayesian optimization with Hyperopt**

Randomly searching the hyperparameter space works reasonably well, but there is a better option: Bayesian optimization. Here we’ll leverage the popular [hyperopt package](https://github.com/hyperopt/hyperopt" \t "_blank) to do so. If you’re unfamiliar with hyperopt and Bayesian optimization, [this article](https://medium.com/district-data-labs/parameter-tuning-with-hyperopt-faa86acdfdce) provides a good overview.

First, define the objective function that we want to minimize. The optimization constraints are included within the objective function by adding a penalty term if the number of clusters falls outside of the desired range:

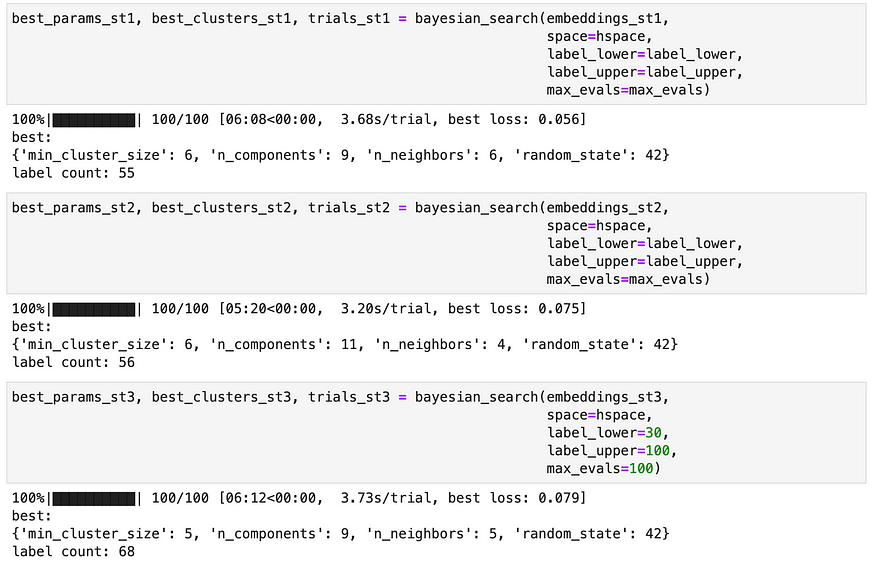
Then minimize the objective function over the hyperparameter search space using the Tree-structured Parzen Estimator (TPE) algorithm:

Running the Bayesian search with 100 max evaluations over our parameter space yields slightly better results than random search:



Results of performing bayesian search of hyperparameter space of UMAP+HDBSCAN. Image by the author.

It’s then easy to run the pipeline using embeddings from multiple different models:



Best hyperparameters found for each model being considered here. Image by the author.

At this point we could do a few more things, like visualize the clusters or manually inspect some of them to make sure they make sense. But ultimately we’re trying to find the “best” clustering results from the best analysis pipeline. If we trust our loss function, then it makes sense to pick the configuration with the lowest loss. Of the combinations tried above, it seems that we should go with sentence-transformer #1 (**all-mpnet-base-v2**), which generated 55 clusters using n\_neighbors = 6, n\_comonents= 9, and min\_cluster\_size = 6.

**Evaluating performance, knowing the ground truth labels**

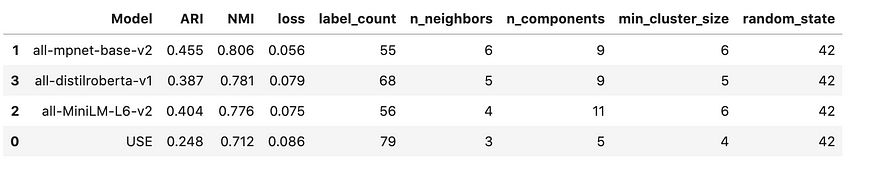
In this case, we happen to also know the ground truth labels so we can see how well our loss function correlates with performance. We can manually inspect how well the models did on some of the ground truth clusters:



Example results for single ground-truth category for four different models. Image by the author.

As shown above, all models seem to do relatively well on placing most of messages in the card\_about\_to\_expire ground-truth group in the same clusters. At least for this category, the first sentence-transformer model seems to stand out in correctly assigning all messages to the same cluster.

Rather than manually inspecting all groups, we can also quantitatively assess the model performances. Two commonly-used metrics for evaluating text clustering are the [Normalized Mutual Information](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html) and [Adjusted Rand Index](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html). Both metrics have values ranging from 0 to 1, where larger is better. Calculating these metrics for the best hyperparameters of the four models under consideration yields:

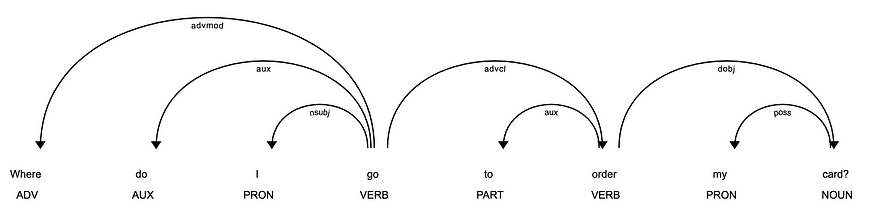


Comparison of four model results and evaluation against ground truth labels. Image by the author.

In agreement with our previous conclusion, sentence-transformer #1 does in fact perform the best, with an ARI of 0.46 and an NMI of 0.81. However, the performance ordering for some of the other models do not follow the order expected from their cost function values. Thus, our scoring method for hyperparameter tuning isn’t perfect, but it is clearly still useful for the current application.

**Automatic cluster labeling**

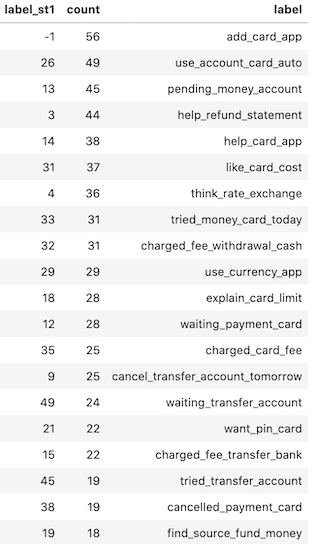
To make the results even more helpful, we can also automatically apply descriptive labels to the clusters we found. A [paper](https://arxiv.org/pdf/2104.12114.pdf) by Liu et al provides an interesting approach to doing this by extracting the most common action-object pair from the phrases in each cluster as the cluster label (e.g. “book-flight”). The bank77 dataset we’re considering here is a little more complicated than the dataset in that paper, but we can do something similar. Here we’ll concatenate the most common verb, direct object, and top two nouns from each cluster. The [spaCy](https://spacy.io/" \t "_blank) package has a powerful syntactic dependency parser that we can use for this:



Result of applying spaCy’s syntactic dependency parser on an example sentence. Image by the author.

We can write a simple function to extract these labels for each cluster:

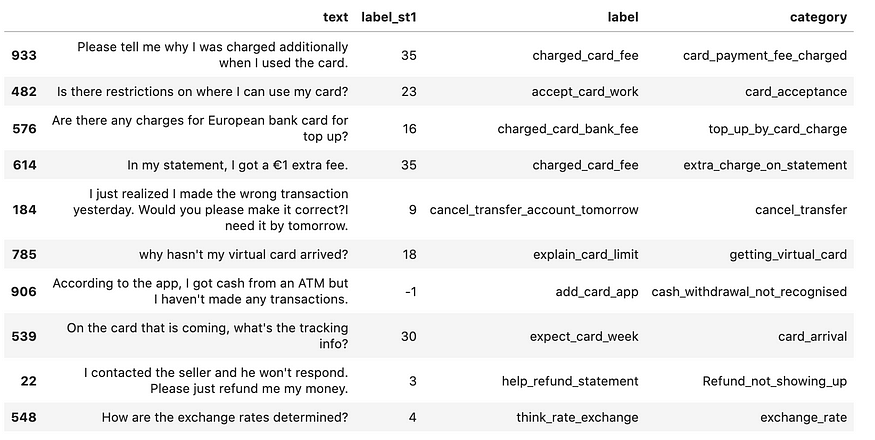
Applying these labels to each of the clusters that our best model found yields our final result:



Summary of extracted clusters with counts and descriptive labels. Image by the author.

Using our tuning method, we’ve automatically extracted and applied descriptive labels to 55 clusters in our dataset.

Since in this case we know the ground truth labels for each document, we can also inspect a few documents to see how well our derived labels match the ground truth labels:



Sample of original data, with derived descriptive label and original ground truth labels (“category” field). Image by the author.

They aren’t perfect, but the extracted labels match the ground category labels pretty well!

**Summary**

In this post, I outlined a framework for leveraging domain knowledge to create a constrained optimization problem to automatically tune UMAP and HDBSCAN hyperparameters. This allows us to easily cluster short-text documents and apply descriptive labels. The focus for this article was on a small dataset, but the same method can be applied to larger datasets as well. The clustering results provide helpful insights of unlabeled text data in a very short amount of time, before deciding or needing to complete time-intensive manual labeling.

All code examples from this article, along with the chatintents python package I created to make applying these concepts easier, can be found here:

**[GitHub - dborrelli/chat-intents: Clustering sentence embeddings to extract message intent](https://github.com/dborrelli/chat-intents?source=post_page-----48d22d3bf02e--------------------------------" \t "_blank)**

[Clustering sentence embeddings to extract message intent - GitHub - dborrelli/chat-intents: Clustering sentence…](https://github.com/dborrelli/chat-intents?source=post_page-----48d22d3bf02e--------------------------------" \t "_blank)

[github.com](https://github.com/dborrelli/chat-intents?source=post_page-----48d22d3bf02e--------------------------------" \t "_blank)

Thanks for reading. If you found this article helpful, drop a comment below or connect on [LinkedIn](http://www.linkedin.com/in/borrellidavid).