Prepared by Group 4

QMBU-420

Predicting Mental Health Problems from Social Media Posts

Introduction

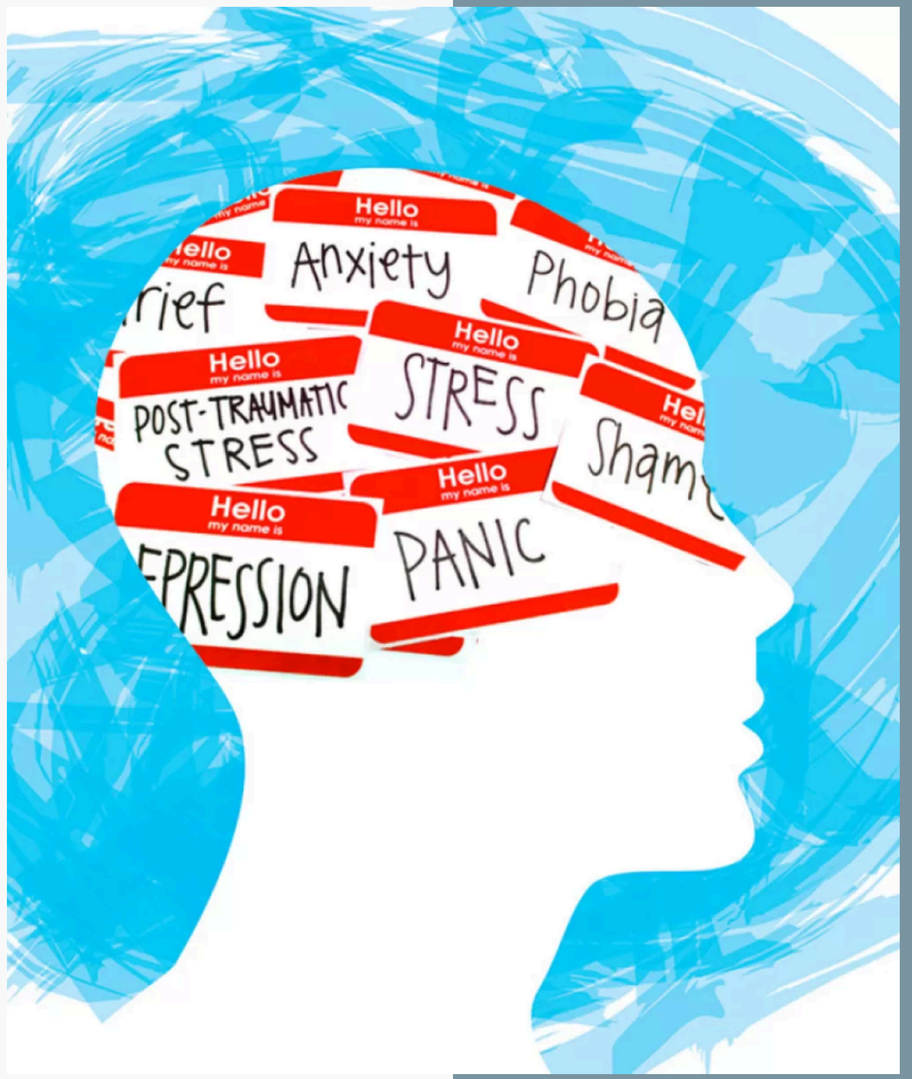
Business & Public Sector

Motivation

- Mental health affects millions globally.
- Early detection through social media text can support public health.
- Goal: Build a predictive model to identify signs of mental health issues.

Research Objective

- Binary classification: Depression vs. Normal
- Multiclass classification: Depression, Anxiety, Bipolar, etc.
- Understand language patterns & support awareness.




feel
life
time
help
hate
bad
hope
love
die
talk
restless
sleep
reason
school
live
hard
pain
night
kill
world
person
wish
health
shit
lot
tell
miss
job
stop
hurt
week
told
look
matter
afraid
clean
own
nice
hear
understand
heart
ago
mental
left
fuck
cold
stupid
lost
don
guess
stuff
care
mind
soon
close
sad
gone
bed
try
start
haha
plan
deal
wrong
sick
dead
bit
month
late
head
home
cry
friend
stress
death
phone
doctor
normal
wake
twitter
worst
mean
forward

Dataset Overview

Source
Kaggle “Sentiment Analysis for Mental Health”

Total:
51,000 labeled social media posts

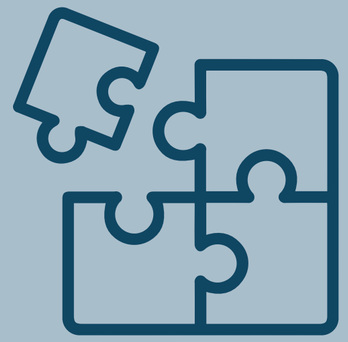
Labels:
depression, anxiety, stress, bipolar, suicidal thoughts, etc.:

Key challenge:
Class imbalance & informal language

Data Preparation Pipeline

Text Preprocessing:

- lowercasing
- emoji removal
- stemming



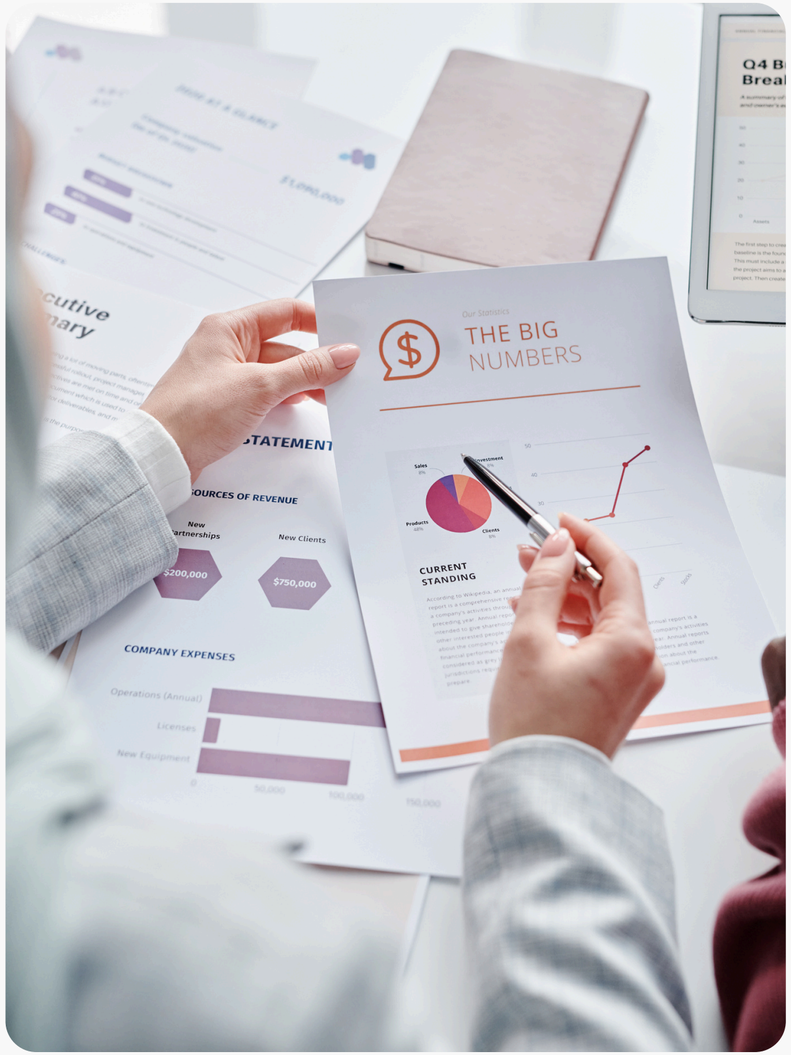
Vectorization:

- TF-IDF & Word2Vec

Class Balancing:

- Used subset of 5,000 rows

Tool Stack



KNIME:

- Visual, low-code environment ideal for non-programmers.
- Easy to chain preprocessing steps
- Great for transparency: Every step is a visible node.

Python:

- Used to get F1 scores.
- Also, to implement BERT modeling

May shift to Scala if performance is a bottleneck

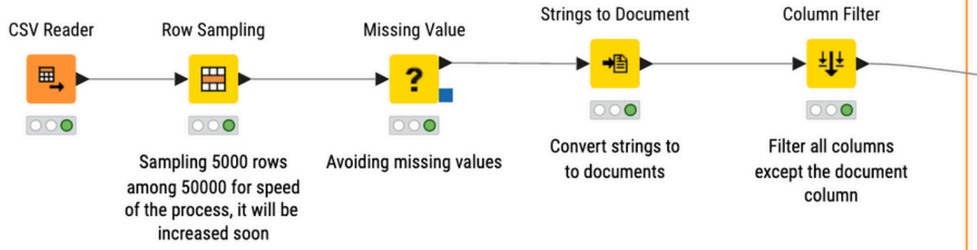
Modeling Techniques

Models Implemented So Far:

- **Decision Tree:** Simple and interpretable; good for understanding feature importance.
- **XGBoost:** Ensemble method; strong performance; handles non-linear relationships better.
- **BERT:** Bidirectional encoder representations from transformers

Reading Textual Data and Parsing

Purpose: Ingest raw data and convert it into KNIME's Document format.



Enrichment

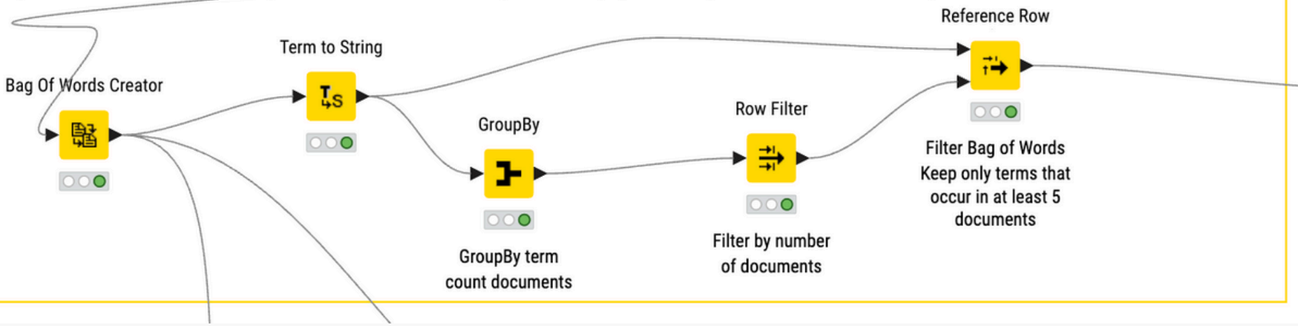
Purpose: Add Part-of-Speech tags to enable filtering by grammatical role (e.g., adjectives,

Preprocessing

Purpose: Clean and normalize the text for consistent downstream processing.

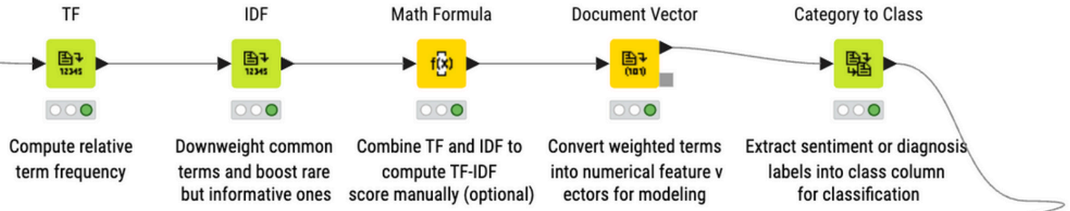
Preprocessing II

Purpose: Convert documents to Bag of Words and filter out infrequent terms (e.g., occurring in fewer than 5 documents).



Transformation

Purpose: Transform text into numerical vectors and extract class labels.



Most Frequent Words Cloud

GroupBy

GroupBy term count documents

Row Filter

Filter by number of documents

Reference Row

Filter Bag of Words
Keep only terms that occur in at least 5 documents

TF

Term Frequency

Tag Cloud

Observing words used more than 5 times in the document

Modeling & Evaluation

Purpose: Train and evaluate the machine learning model using text features.

Color Manager

Assign distinct colors to each class for ROC/visualization nodes"

Partitioning

Training / test set

Decision Tree Learner

Train model

Decision Tree Predictor

Apply decision tree model

Scorer

Column Filter

Filter metric columns to prepare for model comparison

XGBoost Tree Ensemble Learner

Train Model

XGBoost Predictor

Apply xgboost tree model

Scorer

Column Filter

Filter metric columns to prepare for model comparison

Comparison & Documentation

Purpose: This section automates the comparison and documentation of machine learning models by consolidating their evaluation metrics (e.g., F-score), labeling them by model type, and exporting the results to Excel for reporting or further review.

Python Script (1⇒1)

F-Score

Concatenate (Optional in) (deprecated)

Constant Value

Decision Tree Label rows with model type

Concatenate (Optional in) (deprecated)

Python Script (1⇒1)

F-Score

Concatenate (Optional in) (deprecated)

Constant Value

XGBoost Label rows with model type

Concatenate (Optional in) (deprecated)

Excel Sheet Appender (XLS) (deprecated)

Model_Comparison.xlsx

Excel Sheet Appender (XLS) (deprecated)

Model_Comparison.xlsx

Column Filter

Rule-based Row

Sorter

Sort models by evaluation metric to identify top performer

Evaluation Metrics

Modeling – Decision Tree

Results:

Best Recall: Normal (0.90)
F1 Score: 0.585 → **0.619**
Accuracy: 59.8% → **62.54%**
Cohen's Kappa: 0.4726 → **50.96%**

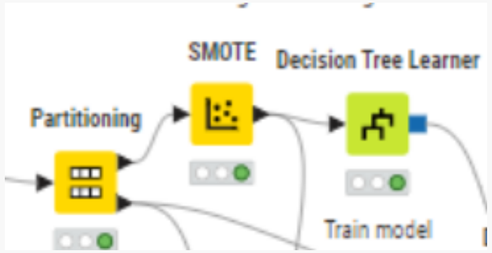
Modeling – XGBoost

Results:

Best Recall: Normal (0.92)
F1 Score: 0.677 → **0.70**
Accuracy: 68.6% → **70.7%**
Cohen's Kappa: 0.5859 → **61.13%**

In mental health, missing a true positive (e.g., not detecting suicidal intent) is much more serious than a false positive. That's why recall and F1 score are prioritized.

XGBoost outperforms Decision Tree on all major metrics.



SAMPLE SIZES

Depressio	Anxiety	Bipolar	Normal	Suicidal	Stress	Personalit
417	**102**	**80**	**491**	**281**	**68**	**16**

Data Analysis

Both models are good at predicting “Normal” posts likely because there are more of them (class imbalance).

Class	XGBoost Recall	Decision Tree Recall
Depression	0.669 → 0.669	0.533 → 0.566
Anxiety	0.673 → 0.63	0.57 → 0.51
Bipolar	0.611 → 0.709	0.429 → 0.519
Normal	**0.923 → 0.916**	**0.903 → 0.874**
Suicidal Thoughts	0.518 → 0.615	0.448 → 0.485
Stress	0.391 → 0.356	0.297 → 0.397
Personality Disorder	0.323 → 0.548	0.161 → 0.451

XGBoost has better overall performance:

F1 Score: 0.677 vs. DTree’s 0.585

Accuracy: 68.6% vs. 59.8%

Cohen’s κ: 0.586 vs. 0.472

Decision Tree is a single model, which can easily overfit or underperform on complex datasets.

XGBoost uses gradient boosting, combining many weak learners (trees) sequentially.

Each tree learns from the errors of the previous ones.

This leads to better generalization and error correction.

- XGBoost has built-in mechanisms **(like weighted classes and learning objectives)** that help mitigate imbalance. This makes it more effective at **learning from minority classes** (e.g., “Personality Disorder” or “Suicidal Thoughts”).

- XGBoost automatically calculates **feature importance**, and its structure handles **noisy and high-dimensional data** better. It likely dealt more effectively with the informal language and sparse text features from the social media dataset.

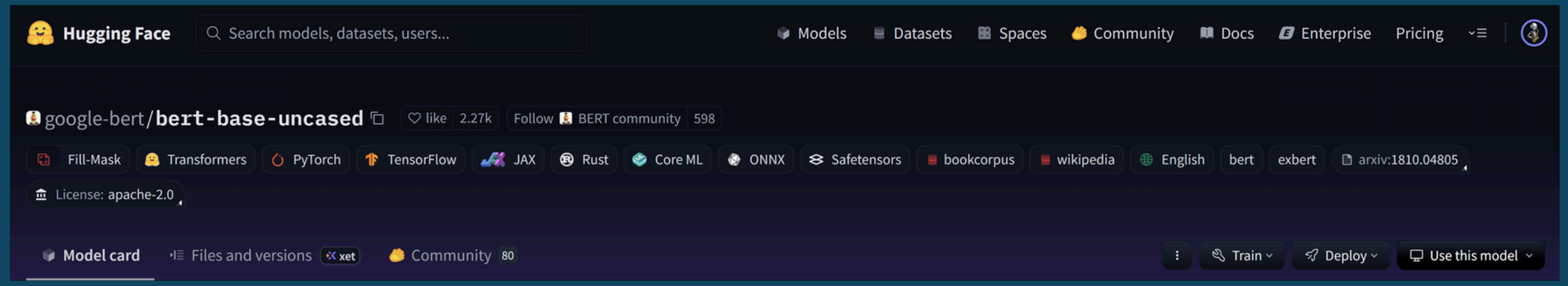
- XGBoost minimizes a custom loss function using **gradient descent**, optimizing performance with each iteration. Decision Trees split data based on heuristic criteria like **Gini** or **entropy**, which might not find globally optimal solutions.

What About Using BERT?

- **Bidirectional encoder representations from transformers** (BERT) is a language model introduced in October 2018 by researchers at Google.

- It learns to represent text as a **sequence of vectors** using **self-supervised** learning. It uses the **encoder-only transformer architecture.**

- BERT dramatically improved the state-of-the-art for large language models. As of 2020, BERT is a ubiquitous baseline in natural language processing (NLP) experiments.




```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
import torch

# Load your dataset
df = pd.read_csv('data.csv')  # replace with your path

# Encode labels
label_encoder = LabelEncoder()
df['label_encoded'] = label_encoder.fit_transform(df['status'])

# Split
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['statement'], df['label_encoded'], test_size=0.3, random_state=42
)
train_texts = train_texts.astype(str).tolist()
val_texts = val_texts.astype(str).tolist()

# Tokenize
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

train_encodings = tokenizer(list(train_texts), truncation=True, padding=True, max_length=128)
val_encodings = tokenizer(list(val_texts), truncation=True, padding=True, max_length=128)

# Create dataset
class MentalHealthDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __len__(self):
        return len(self.labels)
    def __getitem__(self, idx):
        return {
            'input_ids': torch.tensor(self.encodings['input_ids'][idx]),
            'attention_mask': torch.tensor(self.encodings['attention_mask'][idx]),
            'labels': torch.tensor(self.labels[idx])
        }

train_dataset = MentalHealthDataset(train_encodings, train_labels.tolist())
val_dataset = MentalHealthDataset(val_encodings, val_labels.tolist())

# Create model
num_labels = len(label_encoder.classes_)
print(f"Number of labels: {num_labels}")

model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=num_labels)

training_args = TrainingArguments(
    output_dir='./results',
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=1,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10  # Add if you want some logging during training
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)

trainer.train()

# Evaluate the model
eval_result = trainer.evaluate()
print(eval_result)

# Get predictions on the validation set
predictions_output = trainer.predict(val_dataset)
preds = predictions_output.predictions.argmax(-1)  # Get predicted class indices
labels = predictions_output.label_ids              # True labels

# Print or save predictions and true labels
df_preds = pd.DataFrame({
    'true_label': labels,
    'predicted_label': preds
})
print(df_preds.head())

# Optionally, save to CSV for further analysis
df_preds.to_csv('val_predictions.csv', index=False)
```

Results for BERT

Modeling – BERT

Results:

Best Recall: Normal (0.96)

F1 Score: 0.825

Accuracy: 82.5%

Cohen’s Kappa: 0.7722

Modeling – XGBoost

Results:

Best Recall: Normal (0.92)

F1 Score: 0.677

Accuracy: 68.6%

Cohen’s Kappa: 0.5859

BERT (even with 1 epoch of finetuning) outperforms XGBoost on all major metrics.

Recall Scores per Class for BERT

Class	**BERT Recall**	XGBoost Recall
Depression	**0.762**	0.669 → 0.669
Anxiety	**0.859**	0.673 → 0.63
Bipolar	**0.794**	0.611 → 0.709
Normal	**0.962**	0.923 → 0.916
Suicidal Thoughts	**0.731**	0.518 → 0.615
Stress	**0.747**	0.391 → 0.356
Personality Disorder	**0.684**	0.323 → 0.548

Challenges

1. Class Imbalance

- “Normal” class dominates dataset
- We sampled down to 5,000 rows due to compute limits

2. Noisy Text

- Informal language, abbreviations, emojis make it hard to detect signals.
- Preprocessing steps help but don’t eliminate all issues.

3. Ethical Considerations

- Sensitive topic: user-generated mental health data.
- Important to prevent misuse and avoid stigmatizing language.

Future Directions

- Assessing the reliability of current models on binary classification task (Normal versus Not-Normal).
- Checking for Specificity in this folllowing experiment (let’s not diagnose everyone with a disorder eh?)
- Implementing other methods and enhancing the current results.
  - Different transformer based architectures
  - Different hyperparameters
  - Random Forest
  - Naive Bayes
  - Logistic Regression
  - SVM
  - LSTM networks for sequential text
  - BERT like sentence encoders

Thank you