Multi-Dimensional Sentiment Analysis

By Jenna Kudaimi, Roger Nguyen, Aniket Shirodkar

Happy

Sad

Angry

RESEARCH QUESTION:

How much more can sentiment analysis infer to predict emotions (rather than positive/negative) within a given English text, regardless of their background and level of education?

Love

Joy

Fear

DATABASES:

Our goal is to be able to predict a tone based on a given text.

"Sentiment Analysis Word Lists Dataset"

By PRAJWAL KANADE

This kaggle database contains 2 files: a list of negative words and a list of positive words.

"Emotion Dataset for Emotion Recognition Tasks"

By PARUL PANDEY

This kaggle database contains one file with text & an emotion associated with the text; there are 6 emotions to be identified.

Taking the K # of nearest neighbors (supervised)

kNN

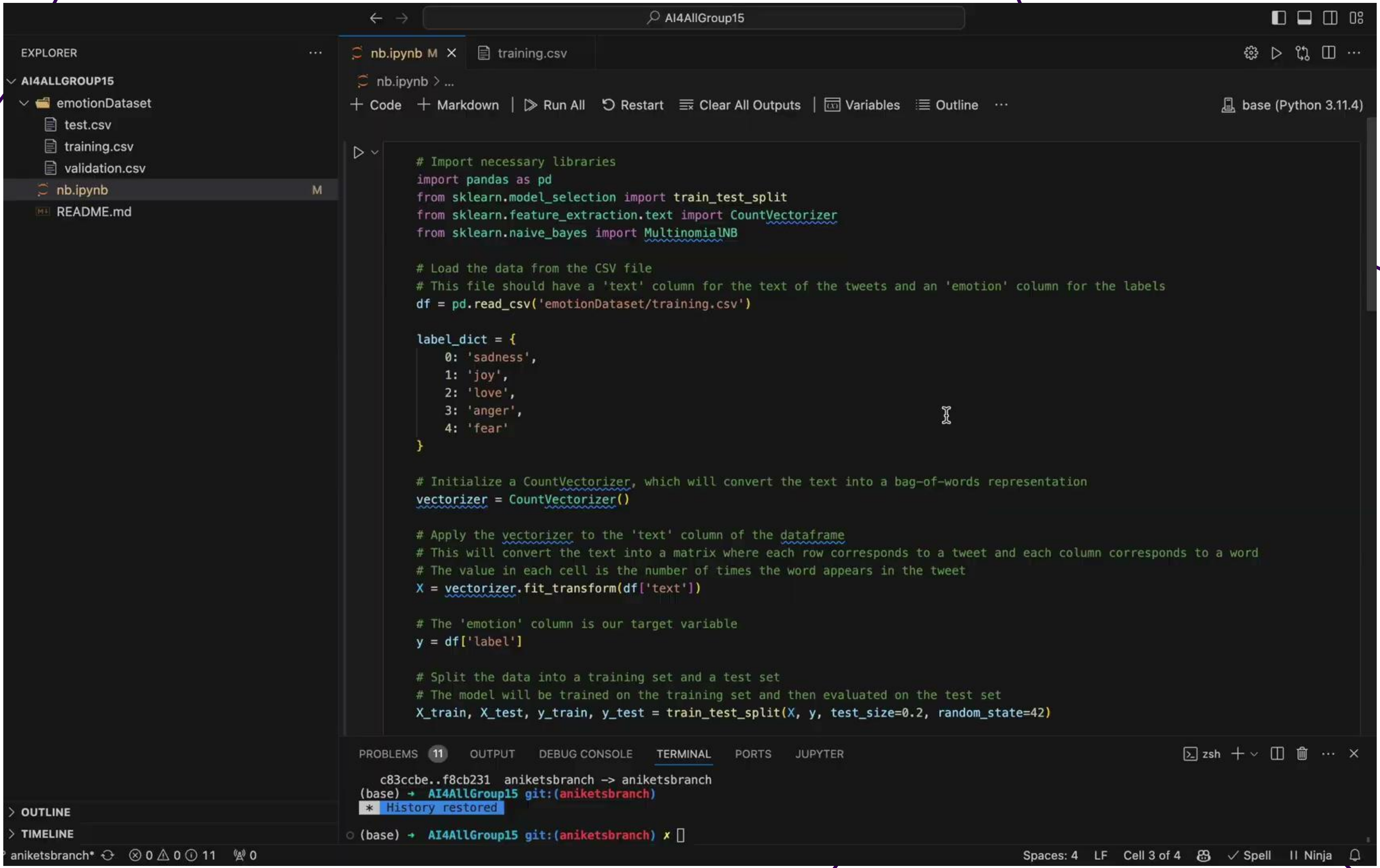
Random Forests

Naive Bayes

Hashing & mapping to a column (Supervised)

Modeling the distribution of inputs (Supervised)

Method 1:
Naive Bayes
(Aniket)


AI4AllGroup15
EXPLORER
AI4ALLGROUP15
emotionDataset
test.csv
training.csv
validation.csv
nb.ipynb M
README.md
nb.ipynb M
training.csv
nb.ipynb > ...
+ Code + Markdown Run All Restart Clear All Outputs Variables Outline
base (Python 3.11.4)
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
# Load the data from the CSV file
# This file should have a 'text' column for the text of the tweets and an 'emotion' column for the labels
df = pd.read_csv('emotionDataset/training.csv')
label_dict = {
0: 'sadness',
1: 'joy',
2: 'love',
3: 'anger',
4: 'fear'
}
# Initialize a CountVectorizer, which will convert the text into a bag-of-words representation
vectorizer = CountVectorizer()
# Apply the vectorizer to the 'text' column of the dataframe
# This will convert the text into a matrix where each row corresponds to a tweet and each column corresponds to a word
# The value in each cell is the number of times the word appears in the tweet
X = vectorizer.fit_transform(df['text'])
# The 'emotion' column is our target variable
y = df['label']
# Split the data into a training set and a test set
# The model will be trained on the training set and then evaluated on the test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
zsh
c83ccbe..f8cb231 aniketsbranch -> aniketsbranch
(base) → AI4AllGroup15 git:(aniketsbranch)
* History restored
(base) → AI4AllGroup15 git:(aniketsbranch) ✗
OUTLINE
TIMELINE
aniketsbranch*
Spaces: 4 LF Cell 3 of 4 Spell Ninja

Method 2: Random Forests (Roger)

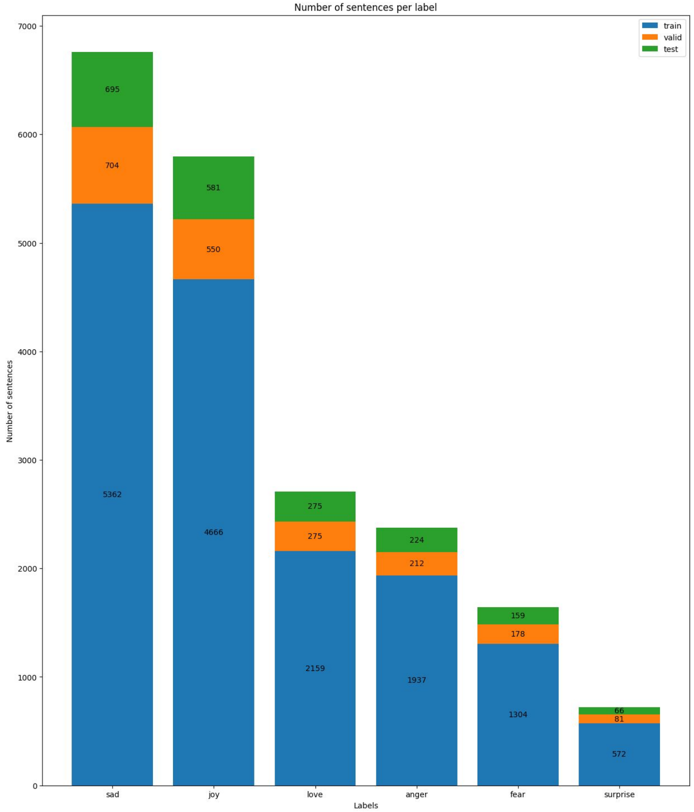
Step 0. Library Import

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import MaxAbsScaler
from scipy.sparse import hstack
from scipy.sparse import csr_matrix
from sklearn.ensemble import RandomForestClassifier
import joblib
import re
```

Python

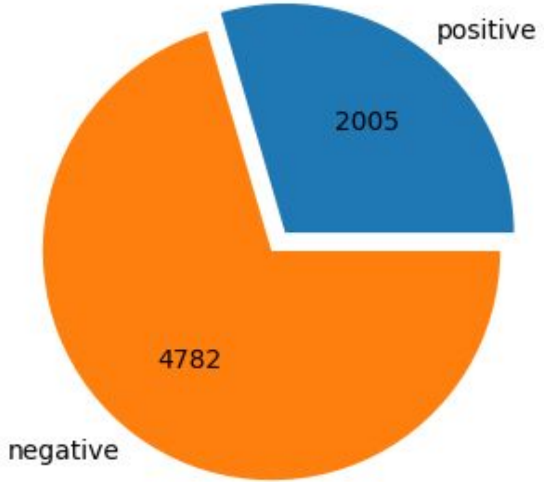
Step 1. Data Visualization & Preparation

Dataset #1. Emotion Dataset for Emotion Recognition Tasks
(16,000 training, 2,000 validation, and 2,000 test tweets with corresponding labels)



Step 1. Data Visualization & Preparation

Dataset #2. Sentiment Analysis Word Lists Dataset (Lists of positive and negative words)



```python
# Load the first datasets (lists of setencnces)

# Load the training data
data = pd.read_csv('dataset/training.csv')

# Load the validation data
valid_data = pd.read_csv('dataset/validation.csv')

# Load the test data
test_data = pd.read_csv('dataset/test.csv')

# 0 = sad
# 1 = joy
# 2 = love
# 3 = anger
# 4 = fear
# 5 = surprise

# Load and preprocess the second datasets (lists of words for sentiment analysis)
negativewords = pd.read_csv('words/negative-words.csv')
negativewords = negativewords.rename(columns={'2-faced': 'neg'})
negativewords['val'] = [-1 for _ in range(len(negativewords))]
negative_words = set(negativewords['neg'].str.lower())

positivewords = pd.read_csv('words/positive-words.csv')
positivewords = positivewords.rename(columns={'a+': 'pos'})
positivewords['val'] = [1 for _ in range(len(positivewords))]
positive_words = set(positivewords['pos'].str.lower())
```
Python

```python
fig, ax = plt.subplots(figsize=(15, 18))
bottom = np.zeros(6)
emotion = ['sad', 'joy', 'love', 'anger', 'fear', 'surprise']

dataset_counts = {
    'train': data['label'].value_counts(),
    'valid': valid_data['label'].value_counts(),
    'test': test_data['label'].value_counts()
}

for dataset, counts in dataset_counts.items():
    p = ax.bar(emotion, counts, label=dataset, bottom=bottom)
    bottom += counts
    ax.bar_label(p, label_type='center')

plt.legend(loc='upper right', labels=['train', 'valid', 'test'])
plt.title('Number of sentences per label')
plt.xlabel('Labels')
plt.ylabel('Number of sentences')
plt.show()
```

Python

```python
pltfig, ax = plt.subplots(figsize=(5,4))
labels = ['positive', 'negative']
sizes = [len(positivewords), len(negativewords)]

def absolute_value(val):
    total_size = sum(sizes)
    absolute_val = int(round(val/100.*total_size))
    return f"{absolute_val:d}"

ax.pie(sizes, labels=labels, explode = (0.1, 0), autopct=absolute_value)
plt.show()
```

Python

Step 2. Data Transformation

- **Feature Selection:** sentiment score, text length, and word count
- **Feature Engineering:**
  - Convert text data into numerical feature vectors
  - Normalize the text length and word count data through Min/Max scaling

```python
# helper function to compute the sentiment of a sentence
def parsing_words(s):
    words = re.findall(r'\b\w+\b', s.lower())
    val = []
    for word in words:
        if word in negative_words:
            val.append(-1)
        if word in positive_words:
            val.append(1)
    if len(val) != 0:
        return sum(val) / len(val)
    return 0
```

Python

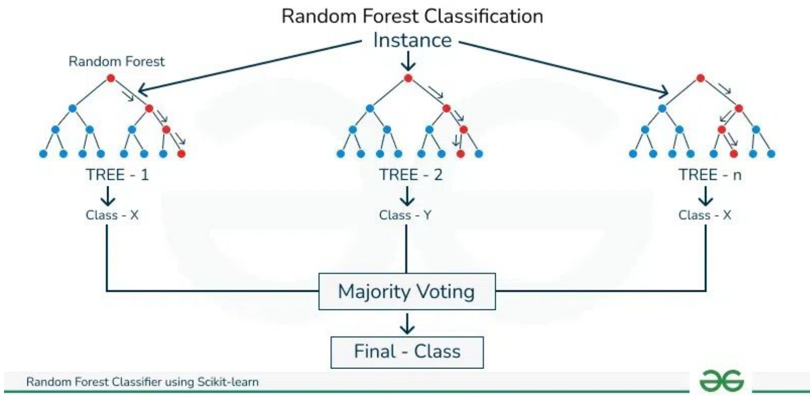
```python
# Global variables for vectorizer and scaler to ensure they are fitted only once
vectorizer = CountVectorizer(ngram_range=(1, 1), max_features=5000, min_df=5)
vectorizer.fit(data['text'])

scaler = MaxAbsScaler()
data['text_length'] = data['text'].apply(len)
data['word_count'] = data['text'].apply(lambda x: len(x.split()))
scaler.fit(data[['text_length', 'word_count']])

# Initialize vectorizer for word counts, with some limitations to avoid memory issues
# Function to create additional features without converting to dense array
def create_features(df):
    # Text length and word count
    df['text_length'] = df['text'].apply(len)
    df['word_count'] = df['text'].apply(lambda x: len(x.split()))

    # Sentiment score
    df['sentiment_score'] = df['text'].apply(parsing_words)

    # Transform text with vectorizer
    X_vect = vectorizer.transform(df['text'])

    # Scale text_length and word_count
    scaled_features = scaler.transform(df[['text_length', 'word_count']])

    all_features = np.hstack((scaled_features, df[['sentiment_score']].to_numpy()))
    all_features_sparse = csr_matrix(all_features)

    # Combine the additional features with the vectorized text (all in sparse format)
    X_combined = hstack((X_vect, all_features_sparse))

    return X_combined
```

Python

Step 3. Model Training

- **Supervised Learning Models for Classification:** Random Forest Classifier

an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time



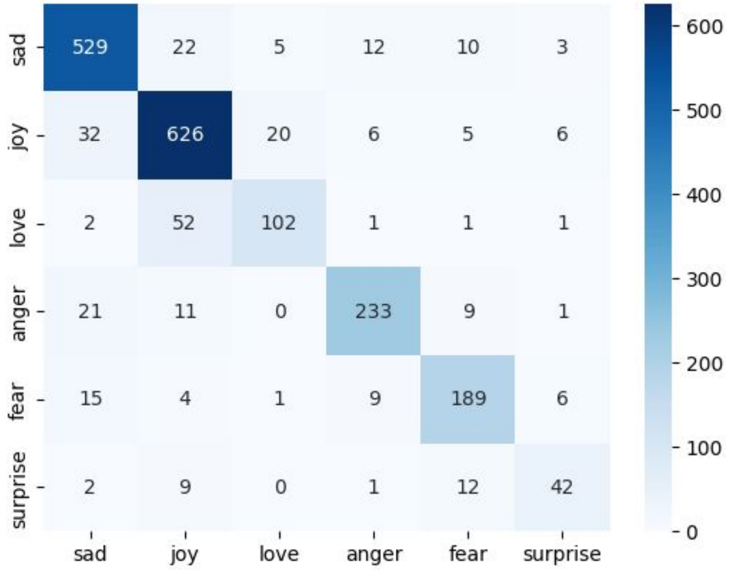
```python
# Assuming 'data' has a 'text' column containing the text and a 'label' column for the labels
y_train = data['label']

# Apply the feature creation to training data
X_train = create_features(data)

# Save the vectorizer and scaler
joblib.dump(vectorizer, 'vectorizer.pkl')
joblib.dump(scaler, 'scaler.pkl')

# Proceed with training the RandomForestClassifier as before
# Initialize the classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the sparse matrix
clf.fit(X_train, y_train)

# Save the trained model
joblib.dump(clf, 'sentiment_classifier.pkl')
```

Python

```
['sentiment_classifier.pkl']
```

Step 4. Model Evaluation

- **Model Development:** Predict the emotion label on validation and test data, Generate the classification report
- **Model Validation:** Confusion Matrix, Accuracy, F1-score (Precision, Recall)



```python
# Load the trained classifier
clf = joblib.load('sentiment_classifier.pkl')

# Load the vectorizer and scaler
vectorizer = joblib.load('vectorizer.pkl')
scaler = joblib.load('scaler.pkl')
```

```python
# Apply the feature creation to test data
X_test_combined = create_features(test_data)

# Predict using the classifier
y_pred = clf.predict(X_test_combined)

# Assuming 'label' is the column with actual labels in test data
y_test = test_data['label']

# Output the classification report
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred, average='weighted'))
print(classification_report(y_test, y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', xticklabels=emotion, yticklabels=emotion)
```

Python

Step 5. Live Demo

Method 3: k-Nearest Neighbors (Jenna)

Step 1: Import & Read Files

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import matplotlib
import matplotlib.pyplot as plt

# Load your dataset here
data = pd.read_csv('training.csv')
negativewords = pd.read_csv('negative-words.csv')
negativewords = negativewords.rename(columns={'2-faced': 'neg'})
negativewords['val'] = [-1 for _ in range(len(negativewords))]




positivewords = pd.read_csv('positive-words.csv')
positivewords = positivewords.rename(columns={'a+': 'pos'})
positivewords['val'] = [1 for _ in range(len(positivewords))]
```

COLAB!
https://colab.research.google.com/drive/1pSA8nwyFSEX97sBmnbMEwTpYW9Ck5jh7?usp=sharing

Step 2: Averaging Sentences

```
[ ] # data = data.sort_values(by=['label'])

    def parsingwords(s):
      words = s.split(' ')
      val = []
      for n in list(negativewords['neg']):
        if s.find(str(n)) != -1:
          val.append(-1)
      for p in positivewords['pos']:
        if s.find(p) != -1:
          val.append(1)
      if len(val) != 0:
        return sum(val)/len(val)
      return 0

    vals = []
    for row in range(len(data)) :
      #print(data['text'][row], data[])
      vals.append(parsingwords(data['text'][row]))
      #print(vals[row], '\n\n')
    data['vals'] = [parsingwords(data['text'][row]) for row in range(len(data))]
    data.head(25)
    #print([parsingwords(data['text'][row]) for row in range(2)])
    #print([data['text'][row] for row in range(2)])
    # 0 = sad
    # 1 = joy
    # 2 = love
    # 3 = anger
    # 4 = fear
    # 5 = surprise
```

	text	label	vals
0	i didnt feel humiliated	0	-1.000000
1	i can go from feeling so hopeless to so damned...	0	-0.500000
2	im grabbing a minute to post i feel greedy wrong	3	-1.000000
3	i am ever feeling nostalgic about the fireplac...	2	0.000000
4	i am feeling grouchy	3	-1.000000
5	ive been feeling a little burdened lately wasn...	0	-1.000000
6	ive been taking or milligrams or times recomme...	5	0.400000
7	i feel as confused about life as a teenager or...	4	-1.000000
8	i have been with petronas for years i feel tha...	1	0.333333
9	i feel romantic too	2	1.000000
10	i feel like i have to make the suffering i m s...	0	-0.333333
11	i do feel that running is a divine experience ...	1	1.000000
12	i think it s the easiest time of year to feel ...	3	0.000000
13	i feel low energy i m just thirsty	0	-1.000000
14	i have immense sympathy with the general point...	1	0.000000
15	i do not feel reassured anxiety is on each side	1	0.333333
16	i didnt really feel that embarrassed	0	-1.000000

Step 3: Build kNN Class

```
[ ]  class KNearestNeighborsClassifier():
         def __init__ (self, k):
             self.number_k = k
             self.depend_var = None
             self.dataframe = None
         def fit (self, dataframe, dependent_variable):
             self.dataframe = dataframe
             self.depend_var = dependent_variable
         def compute_distances (self, sample):
             new_df = self.dataframe.copy()
             independ_vars = [var for var in list(new_df.columns) if var != self.depend_var]
             dist_data = []
             for index in range(new_df[self.depend_var].size):
                 distance = 0
                 for var in independ_vars:
                     distance += (new_df.to_dict()[var][index] - sample[var]) ** 2
                 dist_data.append(math.sqrt(distance))
             new_df['Distance'] = dist_data
             return new_df
         def nearest_neighbors (self, sample):
             sorted_df = self.compute_distances(sample)
             sorted_df = sorted_df.sort_values('Distance')
             sorted_df = sorted_df[['Distance', self.depend_var]]
             return sorted_df
         def classify (self, sample):
             sorted_df = self.nearest_neighbors(sample)
             sorted_df = sorted_df[:self.number_k]
             sorted_dict = sorted_df.to_dict('list')
             counting_dict = {var: {'count' : 0, 'avg_dist' : 0} for var in sorted_dict[self.depend_var]}
             for index in range(sorted_df[self.depend_var].size):
```

```
[ ]          sorted_df = self.nearest_neighbors(sample)
             sorted_df = sorted_df[:self.number_k]
             sorted_dict = sorted_df.to_dict('list')
             counting_dict = {var: {'count' : 0, 'avg_dist' : 0} for var in sorted_dict[self.depend_var]}
             for index in range(sorted_df[self.depend_var].size):
                 key = sorted_dict[self.depend_var][index]
                 counting_dict[key]['count'] += 1
                 counting_dict[key]['avg_dist'] += sorted_dict['Distance'][index]
             highest_count = 0
             lowest_dist = 100000
             by_dist = False
             for var in counting_dict:
                 section = counting_dict[var]
                 section['avg_dist'] = section['avg_dist']/section['count']
                 if section['count'] > highest_count:
                     count_choice = var
                     highest_count = section['count']
                 elif section['count'] == highest_count:
                     by_dist = True
                 if section['avg_dist'] < lowest_dist:
                     dist_choice = var
                     lowest_dist = section['avg_dist']
             if by_dist:
                 return dist_choice
             return count_choice
```

```
[ ]  knn = KNearestNeighborsClassifier(k=5)
     knn.fit(data, dependent_variable = '')
```

Challenges & Issues:

Some issues/challenges we ran into:

- Ambition
- Time constraints
- Data Bias *(Data Visualization previous slides)*

Positive Impact VS Negative Impact

1. Helping Autistic/Neurodivergent People
2. Accessible AI for all
3. Social Media Analysis
4. Business side: being able to interpret if customers are happy or not

1. Socio-Cultural Factors
2. Biases based on text patterns/false interpretations
3. Class Imbalances
4. Business side: could cause stocks to go down, etcetera

ESSENTIAL QUESTION:

How can the development processes of responsible AI/ML solutions address and mitigate the technology’s negative impacts while strengthening its positive effects?

Mitigation & Strengthening in a Cycle

01: Active Consciousness

Through <u>ACTIVE CONSCIOUSNESS,</u> development of an AI/ML, one will keep an open mind and continue to monitor an AI based on these solutions rather than not

02: Quick Solutions

Once having an active consciousness, it allows one to have quick solutions to negative AI impacts in testing phasing before the AI is released

03: Maintaining a *GOOD* AI

After releasing the AI, one can continuously monitor it and its use; being sure to keep tackling biases and negative impact issues so in the future, if it does cause negative impact, it can be fixed.

Next steps?

Summary of our next paths

01 Improving the model with better datasets/more to train (~next few months)

02 Continue learning/researching more about AI (2-4yrs, some of us are Juniors, some of us are Freshman)

03 Working on personal AI projects (~forever!!!!!!!)

04 And more!

Thanks!

Github link:

https://github.com/jkudaimi/AI4AllGroup15OtherCopy

Citations on next slide! :)

Citations

- A Survey of Sentiment Analysis: Approaches, Datasets, and Future Research
- Multi-Dimensional Sentiment Analysis with Learned Representations
- A review on sentiment analysis and emotion detection from text
- Emotion Detection and Sentiment Analysis in Text Corpus: A Differential Study with Informal and Formal Writing Styles
- Current State of Text Sentiment Analysis from Opinion to Emotion Mining
- Sentiment Analysis vs Emotion Detection: what is the difference?
- Beyond Sentiment Analysis: A Review of Recent Trends in Text Based Sentiment Analysis and Emotion Detection