UNIVERSITY OF OULU

TEXT TO EMOTION PROJECT

YOHAN FRANCOIS, FRANZISKA DREIER, ANH NGUYEN HOANG

CONTENT

INTRODUCTION

UNIVERSITY OF OULU

Our objective was to demonstrate how a computer can extract emotions from a given text

Using nltk and scikit-learn libraries, we were able to analyze and preprocess the data in order to train a classifier on this data

SHARE OF WORK

Anh

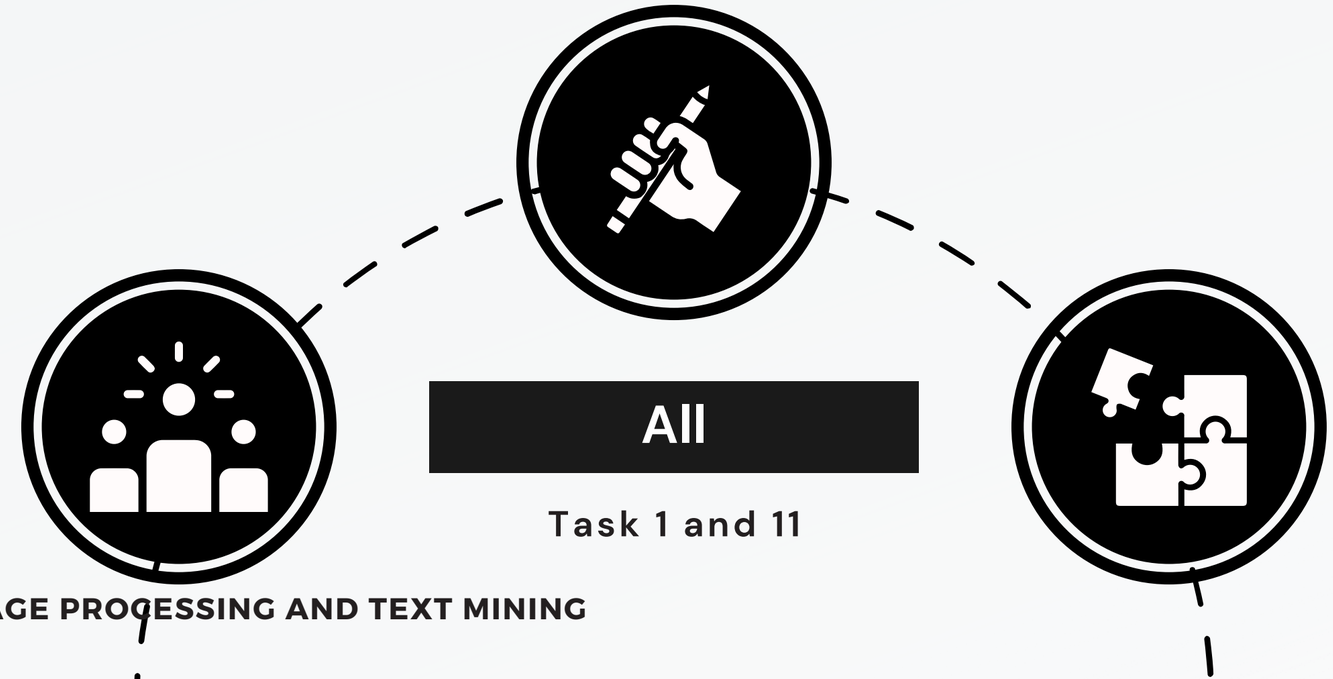
Worked on tasks 5 and 10, with wordclouds and performance improvement

Yohan

Worked on the tasks 6, 7 and 8, about negations and classifier models with TF-IDF vectorization

Franziska

Worked on tasks 2, 3, 4 and 9, about LDA, linguistic quality and DeepMoji



All

Task 1 and 11

FIRST TASK

DEMONSTRATION

UNIVERSITY OF OULU

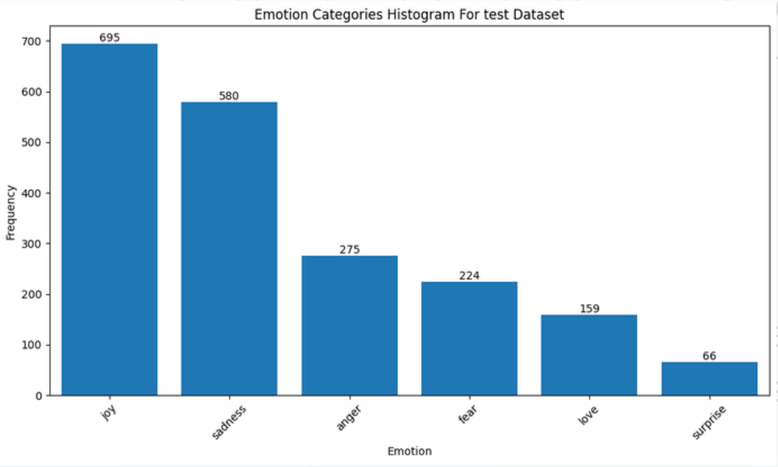
- In the first task, the goal was to show the emotion distribution in the dataset

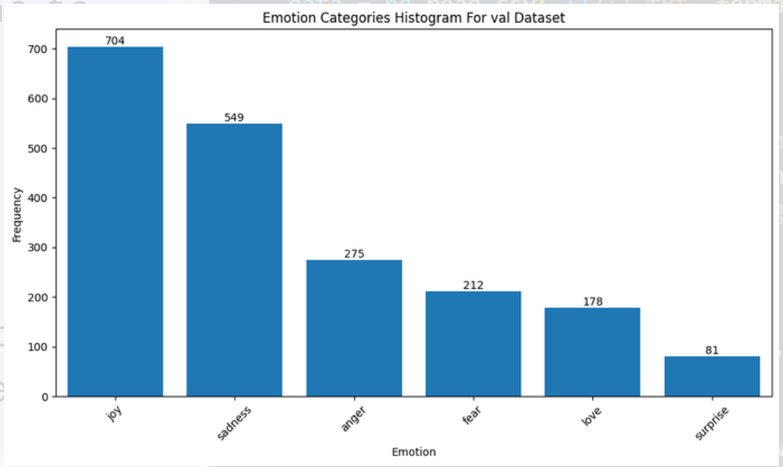
- In order to do this, we created a function that turns a dataset into a bar plot

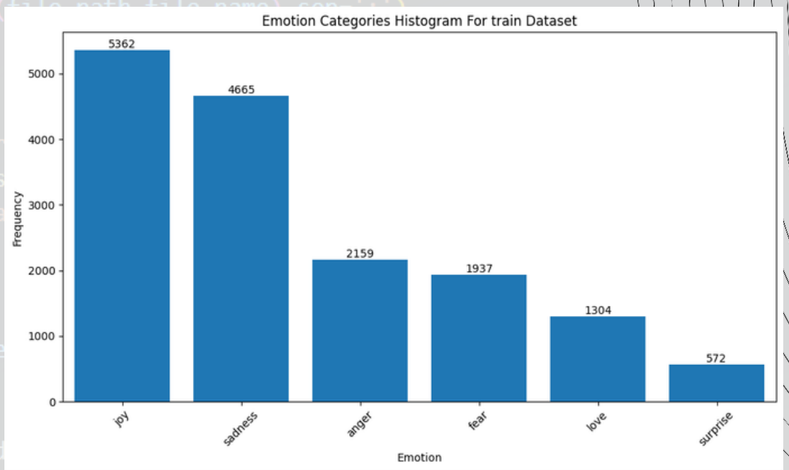
```python
import pandas as pd
import matplotlib.pyplot as plt

def show_historgram(file_path, file_name):
    data = pd.read_csv("{}/{}.txt".format(file_path,file_name),sep=';')
    emotions = data.iloc[:, 1]
    plt.figure(figsize=(10, 6))

    emotions.value_counts().plot(kind='bar', color='skyblue')
    emotion_counts = emotions.value_counts()
    plt.title('Emotion Categories Histogram For {} Dataset'.format(file_name))
    plt.xlabel('Emotion')
    plt.ylabel('Frequency')
    plt.xticks(rotation=45)
    bars = plt.bar(emotion_counts.index, emotion_counts)
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width() / 2, height, int(height), ha='center', va='bottom')

    plt.tight_layout()

    plt.show()

show_historgram("data","test")
show_historgram("data","train")
show_historgram("data","val")
```

FIRST TASK

DEMONSTRATION



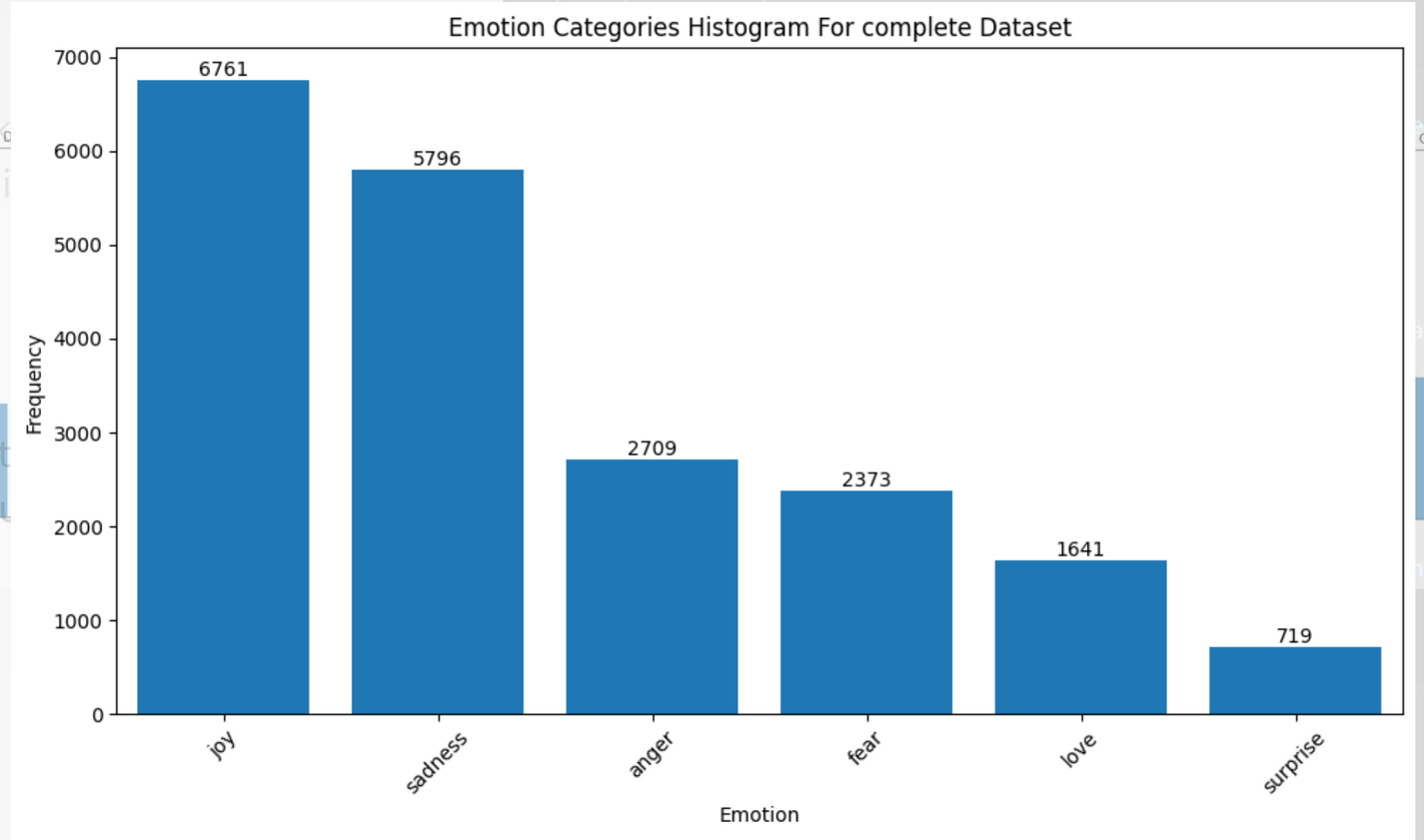





```
import pandas as pd
import matplotlib.pyplot as plt

def show_historgram(file_path, file_name):
    plt.tight_layout()

    plt.show()

show_historgram("data","test")
show_historgram("data","train")
show_historgram("data","val")
```

FIRST TASK

DEMONSTRATION



SECOND TASK

DEMONSTRATION

UNIVERSITY OF OULU

Objectives

- create and store DataFrames for future use
- perform Latent Dirichlet Allocation
- Extra: DataFrames with preprocessed Tweets

Approach

- one function for each objective
- LDA with 3 topics and 4 keywords per topic

```python
# function to read and concatenate a file and save it as dataframe into a .csv
def concat_categories_to_file(file_path, file_name, export_path):
```

```
Dataframes [key: sentences] for test were successfully created and stored as .csv
Dataframes [key: sentences] for val were successfully created and stored as .csv
Dataframes [key: sentences] for train were successfully created and stored as .csv
Dataframes [key: sentences] for complete were successfully created and stored as .csv
```

```python
# function to allocate topics and store results a .csv file
def perform_lda(category: str, data: list, shoud_preprocess: bool,
                num_topis: int, num_words: int):
```

```
anger:
[(0, '0.083*"i" + 0.032*"and" + 0.029*"feel" + 0.024*"the"'),
 (1, '0.070*"i" + 0.034*"the" + 0.032*"to" + 0.026*"feel"'),
 (2, '0.083*"i" + 0.040*"feel" + 0.026*"to" + 0.026*"and"')]
File for "anger" successfully created at: ./results/lda/lda_ana_anger.csv
```

```
[(0, '0.105*"feel" + 0.017*"like" + 0.008*"get" + 0.007*"time"'),
(1, '0.101*"feel" + 0.018*"like" + 0.008*"get" + 0.006*"time"'),
(2, '0.084*"feel" + 0.015*"like" + 0.007*"get" + 0.006*"know"')]
```

THIRD TASK

DEMONSTRATION

UNIVERSITY OF OULU

Objective

Closeness of the topics:

- I1) the total amount of common keywords
- I2) cosine similarity of the average word2vec embedding per topic

Approach

Function that calculates the inter-topic distances

- Gensim Word2Vec model trained with each categories data

```python
# calculate the total amount of common keywords (measure1) and the cosine similarity(measure2)
def calculate_closeness(topic1, topic2, model, vector_size):
    topic1_words = clean_keywords(topic1.split('+'))
    topic2_words = clean_keywords(topic2.split('+'))

    #### I1 - count common keywords of topics ####
    i1 = 0;
    for word in topic1_words:
        if word in topic2_words:
            i1 += 1

    # transform total to relative measure
    #measure1 = points/len(topic1 + topic2) # get percentage

    #### I2 - semantic similarity ####
    # calculate avg. word2vec embeddings per topic
    avg_topic_vector1 = np.empty(vector_size)
    for word in topic1_words:
        avg_topic_vector1 += model.wv[word]
    avg_topic_vector1 /= len(topic1_words)

    avg_topic_vector2 = np.empty(vector_size)
    for word in topic2_words:
        avg_topic_vector2 += model.wv[word]
    avg_topic_vector2 /= len(topic2_words)

    # calculate cosine similarity
    i2 = 1 - distance.cosine(avg_topic_vector1, avg_topic_vector2)

    return i1, i2
```

THIRD TASK

DEMONSTRATION

UNIVERSITY OF OULU

```
# calculate the total amount of common keywords (measure1) and the cosine similarity (measure2)
def calculate_closeness(topic1, topic2, model, vector_size):
```

	category	topics	l1	l2
0	anger	T0&T1	3	0.999999
1	anger	T0&T2	3	0.999999
2	anger	T1&T2	3	0.999884
3	anger	max_values	3	0.999999

	category	topics	l1	l2
3	fear	T0&T1	3	0.999576
4	fear	T0&T2	3	0.999999
5	fear	T1&T2	3	0.999190
6	fear	max_values	3	0.999999

	category	topics	l1	l2
6	joy	T0&T1	3	0.999815
7	joy	T0&T2	3	0.999873
8	joy	T1&T2	3	0.999362
9	joy	max_values	3	0.999873

	category	topics	l1	l2
9	love	T0&T1	2	0.999999
10	love	T0&T2	4	1.000000
11	love	T1&T2	2	0.999837
12	love	max_values	4	1.000000

	category	topics	l1	l2
12	sadness	T0&T1	3	0.999921
13	sadness	T0&T2	3	0.999941
14	sadness	T1&T2	3	0.999845
15	sadness	max_values	3	0.999941

	category	topics	l1	l2
15	surprise	T0&T1	2	0.999998
16	surprise	T0&T2	3	1.000000
17	surprise	T1&T2	3	0.999861
18	surprise	max_values	3	1.000000

```
    # calculate cosine similarity
    i2 = 1 - distance.cosine(avg_topic_vector1, avg_topic_vector2)

    return i1, i2
```

FOURTH TASK

DEMONSTRATION

UNIVERSITY OF OULU

Objective

- assess the quality of the dataset by calculating the linguistic quality ratio

- In order to do this, we created a function that turns a dataset into a bar plot

```
# define the set of English stopwords
english_stopwords = set(stopwords.words('english'))

# define the set of English words in WordNet
english_words = set(words.words())

# check each token for
for token in tokens:
    # stopwords
    if token in english_stopwords:
        stopwords_count += 1
    # not in WordNet
    if token not in english_words:
        not_in_wordnet_count += 1
    # symbols
    if not token.isalnum():
        symbols_count += 1
    # links
    if token.startswith("http://") or token.startswith("https://"):
        links_count += 1
    # numerals
    if token.isnumeric():
        numerals_count += 1

# calculate linguistic quality ratio
cleaned_tokens = (total_tokens - stopwords_count -
                  not_in_wordnet_count - symbols_count - links_count - numerals_count)
linguistic_quality_ratio = cleaned_tokens / total_tokens if total_tokens > 0 else 0.0

return (total_tokens, linguistic_quality_ratio, stopwords_count,
        not_in_wordnet_count, symbols_count + links_count + numerals_count)
```

FOURTH TASK

DEMONSTRATION

UNIVERSITY OF OULU

	Total tokens	Linguistic Quality Ratio	stopwords	not in wordnet	symbols/numbers/etc
anger	52177	0.390057	26879	4946	0
fear	44498	0.398782	22763	3990	0
joy	131401	0.401565	67405	11230	0
love	33833	0.389590	17521	3131	0
sadness	106784	0.400547	54413	9599	0
surprise	14184	0.384729	7260	1467	0

FIFTH TASK

DEMONSTRATION

UNIVERSITY OF OULU

- In the fifth task, the objective was to analyze the parts of speech (nouns, verbs, adjectives, adverbs) in text using POS tagging.

POS COUNT FOR DF complete_preprocessed

Emotion	Noun	Verb	Adjective	Adverb
anger	8080	6245	3134	1024
fear	6883	5544	2690	928
joy	21010	15452	7995	2558
love	5352	3880	1869	612
sadness	16574	13099	6498	2241
surprise	2185	1666	711	310

FIFTH TASK

DEMONSTRATION

UNIVERSITY OF OULU

- Additionally, we also generated a word cloud image which providing a visual representation of the most significant words within each sentence's context.

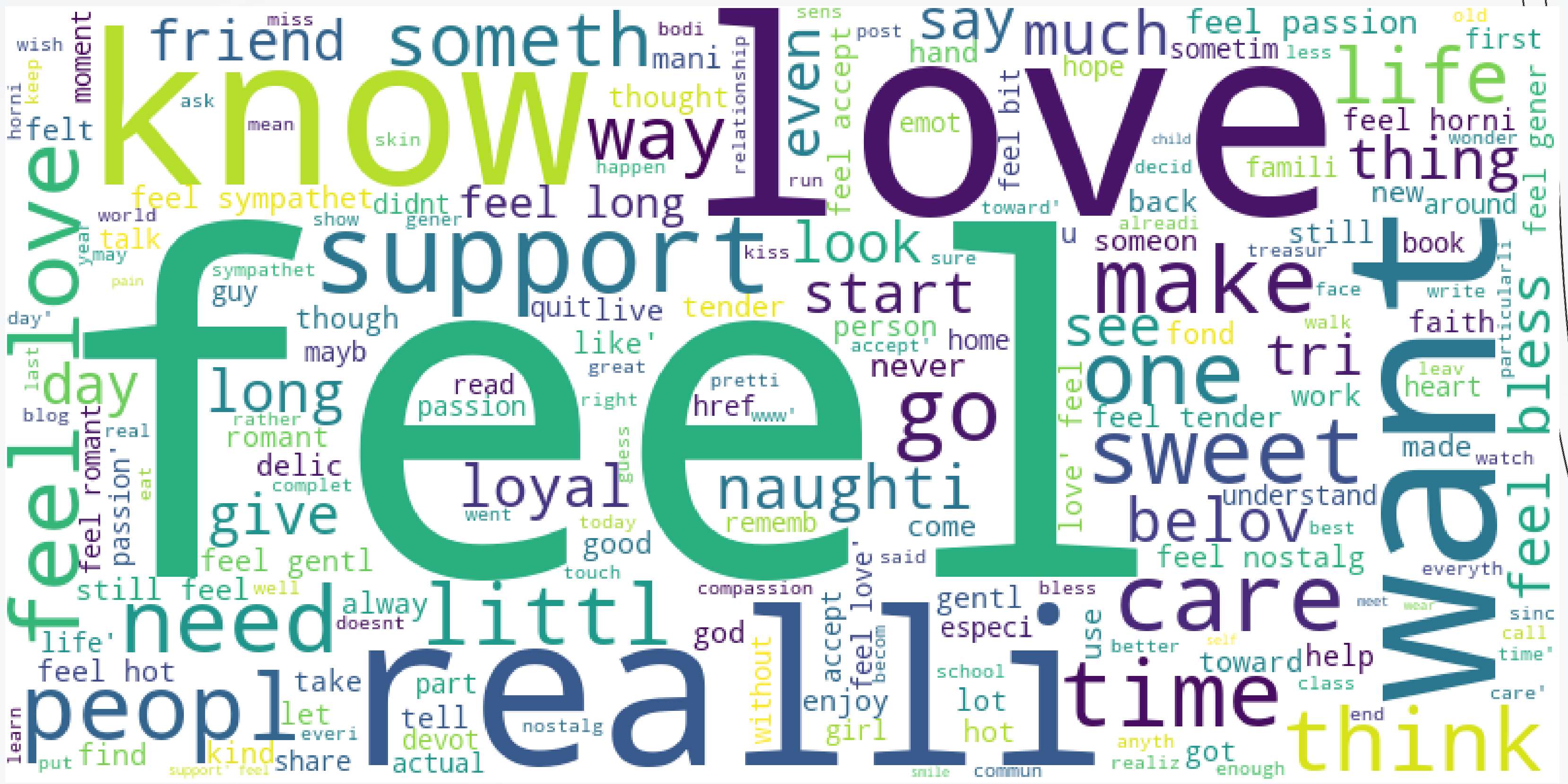


Image for category "Love"

SIXTH TASK

DEMONSTRATION

UNIVERSITY OF OULU

Objective

- Find the average number of negation in each category

Approach

- Function that calculate the average number of negation by category and add that number into the concatenated tweets dataset

```python
import pandas as pd
import nltk

negation_words = {'no', 'not', 'neither', 'never', 'no one', 'nobody', 'none', 'nor', 'nothing', 'nowhere'}
prefixes = {'un', 'im', 'in', 'il', 'ir', 'dis'}
suffix = 'less'

def find_average_number_of_negation_in_each_category(file_path, file_name):
    average_number_of_negation_in_each_category = []
    df = pd.read_csv("{}/{}.csv".format(file_path, file_name))
    for tweets in df["Concatenated_Tweets"]:
        negation_count = 0
        separated_tweets = tweets.replace("[", "").replace("]", "").split(',')
        for tweet in separated_tweets:
            tokens = nltk.word_tokenize(tweet)
            negation_count += sum(1 for token in tokens if token in negation_words)
            negation_count += sum(1 for token in tokens if any(token.startswith(prefix) for prefix in prefixes))
            negation_count += sum(1 for token in tokens if token.endswith(suffix))
        average_number_of_negation_in_each_category.append(round(negation_count/len(separated_tweets), 3))
    df["Average_Number_Of_Negation"]=average_number_of_negation_in_each_category
    return df



test_with_negation = find_average_number_of_negation_in_each_category('categories','test')
train_with_negation = find_average_number_of_negation_in_each_category('categories','train')
val_with_negation = find_average_number_of_negation_in_each_category('categories','val')
```

SIXTH TASK

DEMONSTRATION

Objective
- Find the
negation

Approach
- Function
average
category and add that number into the concatenated tweets dataset

	Category	Concatenated_Tweets	Average_Number_Of_Negation
0	sadness	['im updating my blog because i feel shitty', ...	0.843
1	joy	['i left with my bouquet of red and yellow tul...	0.755
2	fear	['i cant walk into a shop anywhere where i do ...	0.906
3	anger	['i felt anger when at the end of a telephone ...	0.949
4	love	['i find myself in the odd position of feeling...	0.686
5	surprise	['i feel a little stunned but can t imagine wh...	0.773

```
import pandas as pd
import nltk

negation_words = {'no', 'not', 'neither', 'never', 'no one', 'nobody', 'none', 'nor', 'nothing', 'nowhere'}

    return df


test_with_negation = find_average_number_of_negation_in_each_category('categories','test')
train_with_negation = find_average_number_of_negation_in_each_category('categories','train')
val_with_negation = find_average_number_of_negation_in_each_category('categories','val')
```

SEVENTH TASK

DEMONSTRATION

UNIVERSITY OF OULU

Objectives

- Apply a n-gram TF-IDF character vecotrization
- Train a SVM classifier on this data

Approach

- fit_transform the train data
- transform the test data
- fit the SVM model on train data
- predict with the test data

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import f1_score
from sklearn.svm import SVC

# import the python file
import functions_and_variables as fs

def read_txt_data_and_split(file_path, file_name, should_preprocess=False):···

X_train, y_train = read_txt_data_and_split("data", "train", True)
X_test, y_test = read_txt_data_and_split("data", "test", True)
print("Data imported")

tfidf_vectorizer = TfidfVectorizer(ngram_range=(2,3), analyzer='char', max_features=500)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
print("Vectorization done")

svm_classifier = SVC()
print("Blank model created")

svm_classifier.fit(X_train_tfidf, y_train)
print("Model trained")

y_pred = svm_classifier.predict(X_test_tfidf)
print("Prediction done")

f1_svm = f1_score(y_test, y_pred, average=None)
category_names = svm_classifier.classes_

for category, score in zip(category_names, f1_svm):
    print(f"F1-score for {category}: {score}")
```

SEVENTH TASK

DEMONSTRATION

UNIVERSITY OF OULU

Objectives

- Apply a n-gram TF-IDF
  vecotrization
- Train a SVM classifier o

Approach

- fit_transform the train
- transform the test data
- fit the SVM model on tr
- predict with the test da

```
Data imported
Vectorization done
Blank model created
Model trained
Prediction done
F1-score for anger: 0.5505376344086022
F1-score for fear: 0.5891472868217054
F1-score for joy: 0.7242848447961048
F1-score for love: 0.285714285714285 75
F1-score for sadness: 0.7054963084495488
F1-score for surprise: 0.23684210526315788
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import f1_score
from sklearn.svm import SVC

hould_preprocess=False):···

"train", True)
est", True)

), analyzer='char', max_features=500)
ain)

f1_svm = f1_score(y_test, y_pred, average=None)
category_names = svm_classifier.classes_

for category, score in zip(category_names, f1_svm):
    print(f"F1-score for {category}: {score}")
```

EIGHTH TASK

DEMONSTRATION

Objectives

- Apply a bag-of-word TF-IDF vectorization
- Train a classifier on this data

Approach

- fit_transform the train data
- transform the test data
- fit the model on train data
- predict with the test data

```
from sklearn.ensemble import RandomForestClassifier

X_train, y_train = read_txt_data_and_split("data", "train", True)
X_test, y_test = read_txt_data_and_split("data", "test", True)
print("Data imported")

tfidf_vectorizer = TfidfVectorizer(max_features=500)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
print("Vectorization done")

random_forest = RandomForestClassifier()
print("Blank model created")

random_forest.fit(X_train_tfidf, y_train)
print("Model trained")

y_pred = random_forest.predict(X_test_tfidf)
print("Prediction done")

f1_forest = f1_score(y_test, y_pred, average=None)
category_names = random_forest.classes_

for category, score in zip(category_names, f1_forest):
    print(f"F1-score for {category}: {score}")
```

EIGHTH TASK

DEMONSTRATION

UNIVERSITY OF OULU

Objectives

- Apply a bag-of-word TF
  vectorization
- Train a classifier on this

Approach

- fit_transform the train
- transform the test data
- fit the model on train da
- predict with the test da

```
Data imported
Vectorization done
Blank model created
Model trained
Prediction done
F1-score for anger: 0.6467065868263473
F1-score for fear: 0.6447058823529412
F1-score for joy: 0.7083906464924346
F1-score for love: 0.29230769230769227
F1-score for sadness: 0.6606260296540363
F1-score for surprise: 0.6027397260273972
```

SEVENTH AND EIGHTH COMPARAISON

DEMONSTRATION

UNIVERSITY OF OULU

Objectives

- Verifiy how the TF-IDF vectorization method change the F1-score results

```
svm_classifier = SVC()
print("Blank model created")

svm_classifier.fit(X_train_tfidf, y_train)
print("Model trained")

y_pred = svm_classifier.predict(X_test_tfidf)
print("Prediction done")

f1_svm_2 = f1_score(y_test, y_pred, average=None)
category_names = svm_classifier.classes_

for category, score_svm_1, score_svm_2 in zip(category_names, f1_svm, f1_svm_2):
    print(f"F1-score for {category}: {score_svm_1}, {score_svm_2}")
```

SEVENTH AND EIGHTH COMPARAISON

DEMONSTRATION

```
Blank model created
Model trained
Prediction done
F1-score for anger: 0.5505376344086022, 0.6411889596602972
F1-score for fear: 0.5891472868217054, 0.6518518518518519
F1-score for joy: 0.7242848447961048, 0.7139107611548556
F1-score for love: 0.28571428571428575, 0.2418604651162791
F1-score for sadness: 0.7054963084495488, 0.6656126482213438
F1-score for surprise: 0.23684210526315788, 0.55
```

Objectives

- Verifiy how the
  vectorization m
  the F1-score res

```
for category, score_svm_1, score_svm_2 in zip(category_names, f1_svm, f1_svm_2):
    print(f"F1-score for {category}: {score_svm_1}, {score_svm_2}")
```

Conclusion

- The F1-scores increase in certain categories but decrease in others
- There is no optimal solution between thos rwo TF-IDF vectorization methods

NINETH TASK

DEMONSTRATION

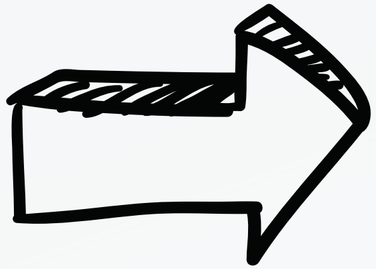
UNIVERSITY OF OULU

Objective

State-of-the-art DeepMoji emotion recognition implementation:

- calculate the F1-score for our dataset
- possibly retrain the model on our training data

- could not run the package because of multiple version issues with nosetest



- create an environment with older versions and try again

Or

- look for a different emotion recognition implementation

TENTH TASK

DEMONSTRATION

UNIVERSITY OF OULU

Objective

- We want to the quality of data augmentation on classification performance.
- I use love and sadness as categories to gather additional dataset and add it to the current data.

- After merging data, we test the SVM classifier and check the F1 score to see if it would improve or not.

TENTH TASK

DEMONSTRATION

```
Comparing to task 7, we can see that the F1 score has improved.
For the task 10:
F1-score for love: 0.594595
F1-score for sadness: 0.651163

For the task 7:
F1-score for love: 0.21649
F1-score for sadness: 0.6379
```

UNIVERSITY OF OULU

THANKS