Memotion Analysis: Emotion Analysis using Internet Memes

Sajib Biswas, Arunima Mandal, Amit Kumar Nath
Department of Computer Science
Florida State University
Emails: {sb19t, am19cf, an17d}@my.fsu.edu

***Abstract*—In the current age of social media based information exchange, the availability of information comes in various modalities including textual, visual and audio formats. For studying social media, most of the time only one significant modality is utilized. However, such is not the case for internet memes. Memes are becoming more popular and widely used every day across almost all social media platforms including Facebook, Instagram, and Twitter. A meme is generally a humorous image (with some text written on the image in most cases) that gets copied in its original form or with some slight variations and then spread throughout social media by users. To the best of our knowledge, Meme emotion (Memotion) analysis has not been explored much. Memotion analysis is a challenging task as we would need to use a hybrid approach for performing computational processing of the memes. The initial research problem is sentiment classification where we have to classify a given internet meme as a positive, negative or neutral one. The next problem is humor classification in which we need to identify the type of humor (sarcastic, humorous, offensive, other) expressed by a given meme. This step is more challenging as a meme can fall into multiple categories. The final problem is quantifying the extent to which a particular effect is being expressed. We perform extensive preprocessing operations followed by feature extraction and finally training models to complete these three tasks. Using four classic algorithms - Naive Bayes Classifier, Linear SVM, Logistic Regression and Random Forest Classifier, we build models to accomplish the tasks. We get moderate accuracy results for each model, with linear SVM providing the best result.**

***Index Terms*—Meme, sentiment classification, humor quantification, machine learning, data mining.**

I. Introduction

With the increasing popularity of social media it has become an absolute necessity to understand and study various popular and interesting aspects and trends of social media. The information available through social media consist of diverse modalities including textual, visual and audio. In most of the cases, only one selected modality is studied for NLP and Computer Vision based studies. However, studying some increasingly popular social media trends like internet memes require a hybrid approach. The extensive use of internet memes in various social media platforms like Facebook, Instagram and Twitter indicates the importance of multimodal social media analysis.

A meme is generally a humorous image - with some text written on the image in most cases that gets copied in its original form or with some slight variations and becomes viral (spread throughout social media by users). Memes can be used to convey a wide range of emotions. The term 'meme' was first coined by Richard Dawkins on his 1976 book titled 'The Selfish Gene'. However, memes as we know these came into popularity after 2010's with the globally popular songs and videos like the gangnam style. Images were taken from those videos and texts added to convey a specific meaning. The word 'meme' is now one of the most typed English words [1]. Memes are often derived from popular culture including a movie or a TV series or a quote from a popular character.

Meme 01: A sarcastic and humorous meme on a feminist men. Here, the text is enough to get humor punchline.

Meme 02: A sarcastic meme on unavailability of Deep Learning based OCR materials on Internet. The extreme shortage of tutorials is conveyed by the man in the meme through the imagery of trying to read a small piece of paper.

Fig. 1. Analysis of Meme Context

A huge amount of photos are shared each day on various social media platforms. Taking into account this immense amount of images and variations in the text, in case of memes, understanding the text in the images is quite different from that done by traditional optical character recognition (OCR) systems. OCR generally recognizes the characters but doesn't understand the context of the image that is associated [2].

As an example, in meme-01 of figure-01, the caption is sufficient enough to sense the sarcasm or dislike expressed towards a feminist man, the image isn't a must to realize the emotion. But in case of meme-02, the overall emotion of the meme is depending on the facial expression of the man in the image, his expression helps to realize the struggle to find OCR tutorials online. In this case a connection needs to be established between the text and the image. Only processing the image would result in losing the humor and the actual intended meaning.

Traditionally when something offensive has been posted

by someone on social media, it has to be seen and flagged by at least one human who can be either an user or a paid worker. Facebook and Twitter rely on outside human contractors like CrowdFlower for these tasks. However, with the immense growth in the volume of multimodal social media, the scaling of these tasks is becoming exponential. As a result offensive conetent detection on social media continues to be an ongoing struggle. OffensEval which is a shared task arranged with SemEval [3], focuses on this offensive content detection. However, detection of offensive memes is quite complicated since it requires visual cue and understanding of language. To address this issue, SemEval introduced the memotion analysis challenge [4], [5]. In this project we tried to address the research challenges provided by the memotion analysis challenge and solve these problems efficiently using data mining and machine learning based approaches.

In this project, we perform the following tasks:

- Sentiment Classification: The initial research problem is sentiment classification where a given internet meme needs to be classified as a positive, negative or neutral one.
- Humor Classification: The next problem is humor classification in which the type of humor (sarcastic, humorous, offensive, motivational) expressed by a given meme needs to be identified
- Quantification: The final problem is quantification i.e., quantifying the extent to which a particular effect is being expressed.

This paper has the following organization. In the next section, we discuss the related works. Section III provides a detailed description of our methodology. In section IV, we discuss the implementation. Sections V and VI contain the evaluation and the conclusion respectively.

II. Related Works

Gal et al. [6] describes memes as performative acts involving a conscious decision to support or reject an ongoing social debate or discussion. Online hate which an extremely brutal and destructive issue, must be detected and prevented at any cost and it's a societal responsibility for many social media companies. With the latest extensive use of internet memes, detecting online hate has become even more challenging as memes can be used to express various kinds of emotions [7]. These observations strengthens the requirement of analyzing internet memes.

Some researchers have explored the idea of automating the meme generation [8], [9] process. They have used deep learning and other relevant approaches for generating memes. Some researchers have tried to extract the inherent sentiment of the memes [10]. In this paper the author explored the correlation between the implied semantic meaning of image-based memes and the textual content of discussions in social media. The author demonstrated that memes are used to emphasize the semantic content in social media communications and that the meanings of memes correlate to the topics of the discussion threads. While there are a number of research works on the semantic importance of internet memes and sentiment classification of regular images (not memes) actual analysis on classifying the emotion expressed by the meme hasn't been explored much. There is a significant scope in this case to incorporate more work for distinguishing the finer aspects of memes such as type of humor or offense.

III. Methodology

To complete the tasks, initially we decided to approach each task individually. Due to time and resource constraints, we decided to work only on the captions (i.e., the OCR extracted and then corrected text from the memes) from memes. Also we didn't find any relevant existing work on memes which consider both the image and the text.

Our first target was completing task-01 i.e., identify a given meme as a positive, negative or neutral one. However, the data set that was provided didn't exactly match with the problem description and had the quantified labels (very positive, positive, neutral, negative, very negative). Hence initially we decided to perform the classification according to the given data set. However, recently the organizers of SemEval informed us that for task 1, we should convert very positive to positive and very negative to negative which will result into three classes negative, positive and neutral. Accordingly we updated the data set and performed task-01 i.e., sentiment classification of the memes. Then we focused on task-02 and task-03, humor classification and quantification, respectively. After analyzing the data set and the problem descriptions, we realized that we can address these two problems together by associating a numeric value to the different subcategories under each category (class) in the given data set. Originally we had the following classes in the given data set - humor, sarcasm, offense, motivation, overall_sentiment. Each class has a set of sub-classes. For humor classification and quantification purposes, we've organized the classes as -

- Humor: not_funny (2), funny (0), very_funny (3), hilarious (1).
- Sarcasm: not_sarcastic (1), general (0), twisted_meaning (2), very_twisted (3).
- Offense: not_offensive (1), slight (2), very_offensive (3), hateful_offensive (0).
- Motivation: not_motivational (1), motivational (0).
- Overall Sentiment: very_negative (3), negative (0), neutral (1), positive (2), very_positive (4).

To complete the tasks we followed the following steps:

A. Step 01: Data Preprocessing

Before performing the tasks, initially we needed to process the data set through several steps to make it usable with necessary information in required form. First we imported the data set (in csv format) and then dropped the unnecessary columns.

Since the machine learning models are mathematical models, we need to convert the textual data into mathematical data. First we bring uniformity to the text by making everything lowercase, removing punctuation an stop words. For gaining

proper insight, we also need to get all words to their root form. We perform lemmatisation to convert the variants of a word to the base word it represents. Along with that, we added code for reverting repetition of letters in a word due to the assumption that hardly any word has letters repeated more than twice.

Then we needed to consider the idea that words which appear only once in the entire sample of the data, aren't most likely to have any significant influence in determining the sentiment of the text. Hence we removed all the rarely occurring words from the data set. These are generally proper nouns and some other insignificant words with respect to the context.

B. Step 02: Feature Extraction

After performing all the data preprocessing operations, we have clean, error-free and precise data, represented in the form of a group of keywords. The next step is performing the feature extraction operation. In this step we extracted some parameters from data to be presented numerically. This step is quite crucial as labeling the features is an absolute must for humor classification and quantification. For our implementation we have used towo features - TF-IDF and Count vectors. Also we split the data into training and testing parts before performing the feature extraction operations.

TF-IDF provides the relative importance of a term in the data and is a measure of the frequency of its appearance in the text. We also considered count vectors to transform the meme captions into an array having the count of appearances of each word in it. The intuition behind using this feature is that the text that conveys similar emoticons might have the same words repeated multiple times, which is a more direct approach.

C. Step 03: Training the models

After preparing the captions, we can directly use those as inputs for some selected machine learning models. For emotion analysis of a meme we developed models using the following algorithms - Multinomial Naive Bayes Classifier, Linear SVM, Logistic Regression, and Random Forest Classifier.

IV. Implementation

According to the course requirements, we implemented our approach entirely in python. We started with the preprocessing operations. After preparing the data for use, we trained the models. In this section we would discuss about the implementation details.

A. Implementation of Preprocessing Operations

1) Reading Dataset and Dropping Unnecessary Columns: Initially we read the data set which is in csv format. Then we dropped the unnecessary columns.

```
### Importing and reading the data set
datapath = os.getcwd()
datapath = datapath + "/data_6512_new.csv"
if not os.path.exists(datapath):
    print ("Data cannot be located!")
data = pd.read_csv(datapath)

### Dropping unnecessary columns
data = data.drop('Image_name', axis=1)
data = data.drop('Image_URL', axis=1)
data = data.drop('OCR_extracted_text', axis=1)
```

2) Lowercase Conversion, Stop Word Removal and Lemmatisation:

```
### Making all letters lowercase
data['Corrected_text'] = data['Corrected_text'].
    apply(lambda x: " ".join(x.lower() for x in x.
    split()))
#Removing Punctuation, Symbols
data['Corrected_text'] = data['Corrected_text'].str.
    replace('[^\w\s]',' ')

### Removing Stop Words using NLTK
from nltk.corpus import stopwords
stop = stopwords.words('english')
data['Corrected_text'] = data['Corrected_text'].
    apply(lambda x: " ".join(x for x in x.split() if
     x not in stop))

### Lemmatisation
from textblob import Word
data['Corrected_text'] = data['Corrected_text'].
    apply(lambda x: " ".join([Word(word).lemmatize()
     for word in x.split()]))
```

3) Correcting Letter Repetitions and Finding Rarest Words:

```
### Correcting Letter Repetitions
import re
def de_repeat(text):
    pattern = re.compile(r"(.)\1{2,}")
    return pattern.sub(r"\1\1", text)

data['Corrected_text'] = data['Corrected_text'].
    apply(lambda x: " ".join(de_repeat(x) for x in x
    .split()))

# Code to find the top 10,000 rarest words appearing
     in the data
freq = pd.Series(' '.join(data['Corrected_text']).
    split()).value_counts()[-10000:]
# Removing all those rarely appearing words from the
     data
freq = list(freq.index)
data['Corrected_text'] = data['Corrected_text'].
    apply(lambda x: " ".join(x for x in x.split() if
     x not in freq))
```

B. Implementation of Feature Extraction Operations

1) Encoding Output Labels:

```
#Encoding output labels 'sadness' as '1' & '
    happiness' as '0'
from sklearn import preprocessing
lbl_enc = preprocessing.LabelEncoder()
#y = lbl_enc.fit_transform(data.Humour.["not funny
    "],data.Humour.["funny"],data.Humour.["very
    funny"])

humor_col = lbl_enc.fit_transform(data.Humour.values
    )
sentiments = np.array(humor_col)
sentiments = np.reshape(sentiments, (-1, 1))

sarcasm_col = lbl_enc.fit_transform(data.Sarcasm.
    values)
sarcasm_np = np.array(sarcasm_col)
sarcasm_np = np.reshape(sarcasm_np, (-1,1))
```

```
sentiments = np.append(sentiments, sarcasm_np, axis = 1)

offensive_col = lbl_enc.fit_transform(data.Offensive.values)
offensive_np = np.array(offensive_col)
offensive_np = np.reshape(offensive_np, (-1,1))
sentiments = np.append(sentiments, offensive_np, axis = 1)

motivational_col = lbl_enc.fit_transform(data.Motivational.values)
motivational_np = np.array(motivational_col)
motivational_np = np.reshape(motivational_np, (-1,1))
sentiments = np.append(sentiments, motivational_np, axis = 1)

overall_col = lbl_enc.fit_transform(data.Overall_Sentiment.values)
overall_np = np.array(overall_col)
overall_np = np.reshape(overall_np, (-1,1))
sentiments = np.append(sentiments, overall_np, axis = 1)
```

2) Splitting the dataset:

```
X_train, X_test, y_train, y_test = train_test_split(data.Corrected_text.values, sentiments, random_state=42, test_size=0.1, shuffle=True)
print("X-Train Shape: ",X_train.shape)
print("X-test Shape: ",X_test.shape)
print("Y-Train Shape: ",y_train.shape)
print("y-test Shape: ",y_test.shape)
print("\n\n")
```

3) Extracting TF-IDf and Count Vectors:

```
##### Extracting TF-IDF parameters
tfidf = TfidfVectorizer(max_features=1000, analyzer='word',ngram_range=(1,3))
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.fit_transform(X_test)

#### Extracting Count Vectors Parameters
count_vect = CountVectorizer(analyzer='word')
count_vect.fit(data['Corrected_text'])
X_train_count_vect = count_vect.transform(X_train)
X_test_count_vect =  count_vect.transform(X_test)
```

C. Training the Models

1) Training Multinomial Naive Bayes Classifier:

```
def runMultinomialNB(X_train_extra, X_test_extra):

    # Model 1: Multinomial Naive Bayes Classifier

    nb1 = MultinomialNB()
    nb2 = MultinomialNB()
    nb3 = MultinomialNB()
    nb4 = MultinomialNB()
    nb5 = MultinomialNB()

    nb1.fit(X_train_extra, y_train[:,0])
    nb2.fit(X_train_extra, y_train[:,1])
    nb3.fit(X_train_extra, y_train[:,2])
    nb4.fit(X_train_extra, y_train[:,3])
    nb5.fit(X_train_extra, y_train[:,4])

    Accuracies=[]
    rows = X_test_extra.shape[0]
    for i in range(rows):
        Predictions = []
        pred1 = nb1.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred1)
        pred2 = nb2.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred2)
        pred3 = nb3.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred3)
        pred4 = nb4.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred4)
        pred5 = nb5.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred5)
        ##print(Predictions)
        intersection = np.count_nonzero(np.bitwise_and(y_test[i,:],Predictions), axis=0)
        union = np.count_nonzero(np.bitwise_or(y_test[i,:],Predictions), axis=0)
        # print (intersecTP,"  ",unionTP)
        Accuracies.append(float(intersection)/float(union))

    #Average this value over all the test samples to compute the final test accuracy
    percent_avg = sum(Accuracies)/rows*100
    print ("Avg accuracy - MultinomialNB: %",end=" " ) #percent_avg,"%")
    print('%0.2f'% percent_avg)
```

2) Training Linear SVM Classifier:

```
def runSGDClassifier(X_train_extra, X_test_extra):

    # Model 2: Linear SVM
    lsvm1 = SGDClassifier(alpha=0.001, random_state=5, max_iter=15, tol=None)
    lsvm2 = SGDClassifier(alpha=0.001, random_state=5, max_iter=15, tol=None)
    lsvm3 = SGDClassifier(alpha=0.001, random_state=5, max_iter=15, tol=None)
    lsvm4 = SGDClassifier(alpha=0.001, random_state=5, max_iter=15, tol=None)
    lsvm5 = SGDClassifier(alpha=0.001, random_state=5, max_iter=15, tol=None)

    lsvm1.fit(X_train_extra, y_train[:,0])
    lsvm2.fit(X_train_extra, y_train[:,1])
    lsvm3.fit(X_train_extra, y_train[:,2])
    lsvm4.fit(X_train_extra, y_train[:,3])
    lsvm5.fit(X_train_extra, y_train[:,4])

    Accuracies=[]
    rows = X_test_extra.shape[0]
    for i in range(rows):
        Predictions = []
        pred1 = lsvm1.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred1)
        pred2 = lsvm2.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred2)
        pred3 = lsvm3.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred3)
        pred4 = lsvm4.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred4)
        pred5 = lsvm5.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred5)
        ##print(Predictions)
        intersection = np.count_nonzero(np.bitwise_and(y_test[i,:],Predictions), axis=0)
```

```
        union = np.count_nonzero(np.bitwise_or(y_test[i,:],Predictions), axis=0)
        # print (intersecTP,"  ",unionTP)
        Accuracies.append(float(intersection)/float(union))

    #Average this value over all the test samples to compute the final test accuracy
    percent_avg = sum(Accuracies)/rows*100
    print ("Avg accuracy - LinearSVM: %",end=" " ) #percent_avg,"%")
    print('%0.2f'% percent_avg)
```

3) Training Logistic Regression Classifier:

```
def runLogisticRegression(X_train_extra, X_test_extra):
    # Model 3: logistic regression

    logreg1 = LogisticRegression(solver='lbfgs', multi_class='auto', max_iter=4000)
    logreg2 = LogisticRegression(solver='lbfgs', multi_class='auto', max_iter=4000)
    logreg3 = LogisticRegression(solver='lbfgs', multi_class='auto', max_iter=4000)
    logreg4 = LogisticRegression(solver='lbfgs', multi_class='auto', max_iter=4000)
    logreg5 = LogisticRegression(solver='lbfgs', multi_class='auto', max_iter=4000)

    logreg1.fit(X_train_extra, y_train[:,0])
    logreg2.fit(X_train_extra, y_train[:,1])
    logreg3.fit(X_train_extra, y_train[:,2])
    logreg4.fit(X_train_extra, y_train[:,3])
    logreg5.fit(X_train_extra, y_train[:,4])

    Accuracies=[]
    rows = X_test_tfidf.shape[0]
    for i in range(rows):
        Predictions = []
        pred1 = logreg1.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred1)
        pred2 = logreg2.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred2)
        pred3 = logreg3.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred3)
        pred4 = logreg4.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred4)
        pred5 = logreg5.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred5)
        ##print(Predictions)
        intersection = np.count_nonzero(np.bitwise_and(y_test[i,:],Predictions), axis=0)
        union = np.count_nonzero(np.bitwise_or(y_test[i,:],Predictions), axis=0)
        # print (intersecTP,"  ",unionTP)
        Accuracies.append(float(intersection)/float(union))

    #Average this value over all the test samples to compute the final test accuracy
    percent_avg = sum(Accuracies)/rows*100
    print ("Avg accuracy-Logistic Regresssion: %", end=" " ) #percent_avg,"%")
    print('%0.2f'% percent_avg)
```

4) Training Random Forest Classifier:

```
def runRandomForestClassifier(X_train_extra, X_test_extra):
    # Model 4: Random Forest Classifier

    rf1 = RandomForestClassifier(n_estimators=500)
    rf2 = RandomForestClassifier(n_estimators=500)
    rf3 = RandomForestClassifier(n_estimators=500)
    rf4 = RandomForestClassifier(n_estimators=500)
    rf5 = RandomForestClassifier(n_estimators=500)

    rf1.fit(X_train_extra, y_train[:,0])
    rf2.fit(X_train_extra, y_train[:,1])
    rf3.fit(X_train_extra, y_train[:,2])
    rf4.fit(X_train_extra, y_train[:,3])
    rf5.fit(X_train_extra, y_train[:,4])

    Accuracies=[]
    rows = X_test_tfidf.shape[0]
    for i in range(rows):
        Predictions = []
        pred1 = rf1.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred1)
        pred2 = rf2.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred2)
        pred3 = rf3.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred3)
        pred4 = rf4.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred4)
        pred5 = rf5.predict(X_test_extra[i,:].reshape(1,-1))[0]
        Predictions.append(pred5)
        ##print(Predictions)
        intersection = np.count_nonzero(np.bitwise_and(y_test[i,:],Predictions), axis=0)
        union = np.count_nonzero(np.bitwise_or(y_test[i,:],Predictions), axis=0)
        # print (intersecTP,"  ",unionTP)
        Accuracies.append(float(intersection)/float(union))

    #Average this value over all the test samples to compute the final test accuracy
    percent_avg = sum(Accuracies)/rows*100
    print ("Avg accuracy - Random Forest: %",end=" " ) #percent_avg,"%")
    print('%0.2f'% percent_avg)
```

5) Running Classifiers:

```
print("for TF-IDF:")
runMultinomialNB(X_train_tfidf, X_test_tfidf)
runSGDClassifier(X_train_tfidf, X_test_tfidf)
runLogisticRegression(X_train_tfidf, X_test_tfidf)
runRandomForestClassifier(X_train_tfidf, X_test_tfidf)
print("\nfor count vector:")
runMultinomialNB(X_train_count_vect, X_test_count_vect)
runSGDClassifier(X_train_count_vect, X_test_count_vect)
runLogisticRegression(X_train_count_vect, X_test_count_vect)
runRandomForestClassifier(X_train_count_vect, X_test_count_vect)
```

V. Evaluation

The performance of our approach is evaluated in this section. Here we compare the performance of the different algorithms which have been used to train the models.

A. Experimental Setup

Our experimental setup included-

- Windows 10 machine
- Intel Core i7 2.3 GHz processor.
- 8 GB RAM

1) Dataset: We registered as a team in SemEval-2020 (International Workshop on Semantic Evaluation 2020) competition [3]. Evolving from the SensEval word sense disambiguation evaluation series, SemEval provides several tasks belonging to different categories. Among them we've selected the task called Memotion Analysis which is relevant to our project. SemEval released 7K annotated memes - with human annotated tags namely sentiment, and type of humor that is, sarcastic, humorous, or offensive. The humor types are further quantified on a scale. For performing meme emotion analysis both both textual and visual cues required. To address this issue, SemEval would also provide the caption from the memes as a part of the dataset. The caption extraction process is performed using the Google OCR system and then manually corrected with the help of crowdsourcing services [1].

We're using the training dataset (in csv format) from the challenge for both our training and testing purposes. The csv file has the following format - Image_name, Image_url, OCR_extracted_text, Corrected_text, Humour, Sarcasm, Offense, Motivation, Overall_sentiment, Basis_of_classification. The arrangement of the dataset didn't properly match with the original problem description. We had to go through the entire dataset to address the ambiguities and re-organized the categories and sub-categories. We've performed extensive preprocessing operations on the dataset and eliminated the noise.

- **View of the dataset:**

	A	B	C	D	E	F	G	H	I	J
1	Image_name	Image_URL	OCR_extractec	Corrected_text	Humour	Sarcasm	Offensive	Motivational	Overall_Sentiment	
2	10_year_2r94rv	https://i.imgflip	LOOK THERE M	LOOK THERE MY	hilarious	general	not_offensive	not_motivatio	very_positive	
3	10_year_10-yea	https://spiderin	The best of #1(	The best of #10 \	not_funny	general	not_offensive	motivational	very_positive	
4	10_year_10year	https://www.lif	Sam Thorne @	Sam Thorne @St	very_funny	not_sarcastic	not_offensive	not_motivatio	positive	
5	10_year_10-yea	https://pics.con	10 Year Challer	10 Year Challeng	very_funny	twisted_mea	very_offensiv	motivational	positive	
6	10_year_10-yea	https://pics.me	10 YEAR CHALL	10 YEAR CHALLEI	hilarious	very_twisted	very_offensiv	not_motivatio	neutral	

- **Classes and Ordinality:**

Humor:		Sarcasm:		Offense:		Motivation:		Overall_sentiment:	
not_funny	(2)	not_sarcastic	(1)	not_offensive	(1)	Not_motivational	(1)	very_negative	(3)
funny	(0)	general	(0)	slight	(2)	motivational	(0)	negative	(0)
very_funny	(3)	Twisted_meaning	(2)	very_offensive	(3)			neutral	(1)
hilarious	(1)	very_twisted	(3)	hateful_offensive	(0)			positive	(2)
								very_positive	(4)

Fig. 2. View of the Dataset and Ordanilty of the Classes

2) Accuracy Comparison: We used accuracy as the performance metric to compare the performances of the different algorithms. First we analyzed the accuracy of the models for task-01. In case of task-01, for TF-IDF based trained models, svm provided the best result at 61.96%. For count vector based models also SVM performs best with 59.66% accuracy. On the other hand, for combined task-02 and task-03, we also notice that SVM outperforms the other models by getting accuracy of 52.43% and 51.89% for TF-IDF and count vector models respectively.

```
######## Task 01 ########
Accuracy of Training Models using TF-IDF:
Multinomial Naive Bayes: % 59.82
Linear SVM: % 61.96
Logistic Regresssion: % 55.06
Random Forest: % 52.76


Accuracy of Training Models using Count Vector:
Multinomial Naive Bayes: % 54.29
Linear SVM: % 59.36
Logistic Regresssion: % 54.29
Random Forest: % 53.83
```

Fig. 3. Accuracy of task-01

```
######## Task 02 & 03 ########
Average Accuracy for TF-IDF:
Avg accuracy - MultinomialNB: % 50.59
Avg accuracy - LinearSVM: % 51.51
Avg accuracy-Logistic Regresssion: % 48.91
Avg accuracy - Random Forest: % 47.29

Average Accuracy for Count Vector:
Avg accuracy - MultinomialNB: % 49.36
Avg accuracy - LinearSVM: % 52.19
Avg accuracy-Logistic Regresssion: % 49.93
Avg accuracy - Random Forest: % 47.60
```

Fig. 4. Accuracy of task-02 and task-03

We have also included some graphs for better and easier understanding of the comparison. In Figure 5 and 6 we are showing the accuracy vs model graph for Task 01. Figure 7 and 8 shows the same type of graph for Taks 02 and 03.

VI. Future Works

The most challenging part of the tasks was quantifying the labels to the extent that it is being precisely expressed i.e. very funny, hilarious, not funny, etc. So this problem falls under the multi-class quantification problem, to be more specific - Ordinal Quantification problem. For that, we wanted to apply OQT (Ordinal Quantification Tree) which exploits the apparently good performance of PCC (Probabilistic Classify and Count) for sentiment quantification to build an ordinal quantification model composed by a collection of binary PCC's arranged in a decision tree [11]. But due to the time constraint, we were unable to implement that.

We explored another technique namely Ranking SVM. The Ranking SVM algorithm is a learning retrieval function that

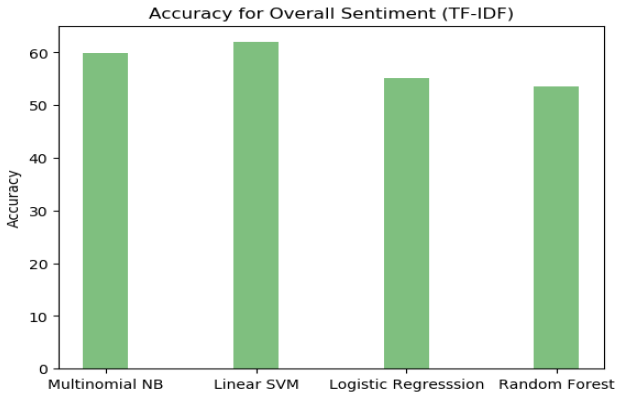

Fig. 5. Task-01 Accuracy vs Model graph for TF-IDF

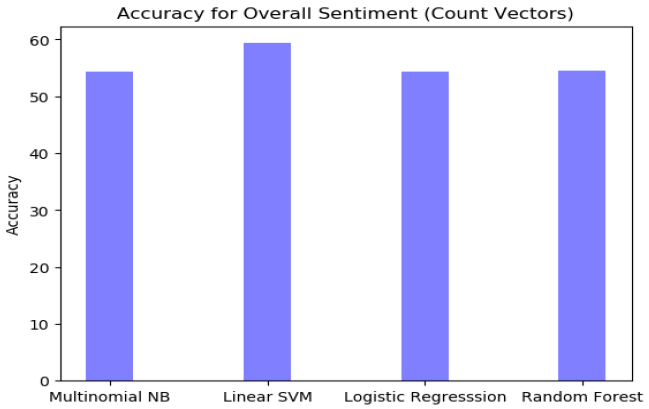

Fig. 6. Task-01 Accuracy vs Model graph for Count Vector

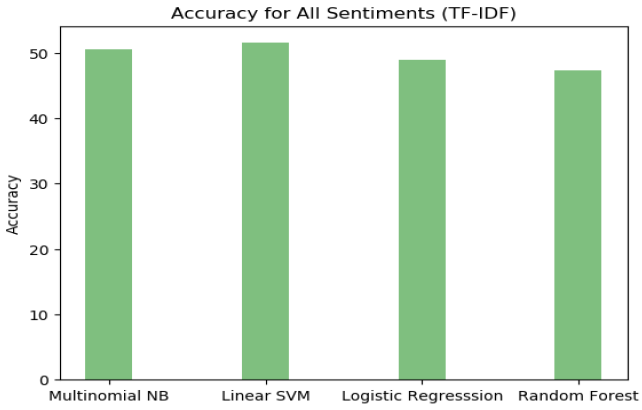

Fig. 7. Task-02 and Task-03 Accuracy vs Model graph for TF-IDF

employs pair-wise ranking methods to adaptively sort results based on how 'relevant' they are for a specific query. The Ranking SVM function uses a mapping function to describe the match between a search query and the features of each of the possible results. This mapping function projects each data pair (such as a search query and clicked web-page, for example) onto a feature space. These features are combined

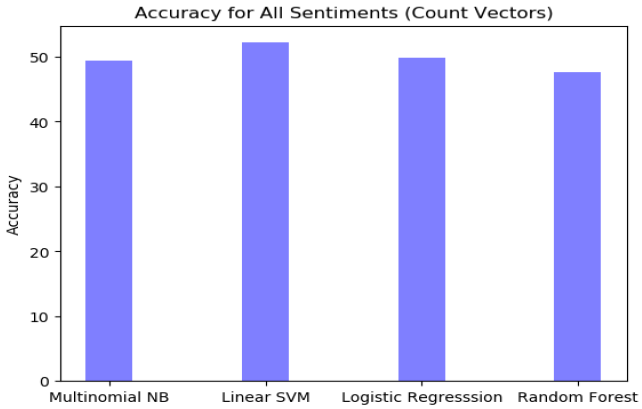

Fig. 8. Task-02 and Task 03 Accuracy vs Model graph for Count Vector

with the corresponding click-through data (which can act as a proxy for how relevant a page is for a specific query) and can then be used as the training data for the Ranking SVM algorithm [12].

SVM-Rank uses pairwise difference vectors to produce the rank list of items. So, given a list of items represented as a feature vector, if we want to order them then SVM-Rank is the right thing to use. So after analyzing the algorithm, we have realized that Ranking SVM is primarily to rank the ordinal values, which includes applications like ranking search engine results, ranking documents, etc.

We have analyzed another paper where a new method for document retrieval on the basis of learning to rank is used [13]. The whole working method of Ranking SVM is quite different from quantifying a sentiment which is way more complex. Because in case of sentiment analysis of memes, each sample test contains very small text size compared to tweet, social media status or any document or any review. And there is no question to compare one sample text meme with another sample text meme which is done in case of Ranking SVM.

We explored these existing and implemented particular algorithms in detail, particularly for quantifying and getting better accuracy. Due to time constraint ultimately we were able to implemented the classic algorithms. In future we will try to implement the specific algorithm OQT (Ordinal Quantification Tree) which was introduced by De San Martino et el., as this algorithm seems to be mostly relevant to our sentiment quantification problem. Also we will take into consideration the image context during analysis. Thus both text and image analysis will be combined to produce the overall emotion of the memes.

VII. Conclusion

In this project, we addressed the yet not so much explored problem of emotion analysis from internet memes. Classifying memes into positive, negative and neutral was our first step. And then we categorized the memes, not only according to different emotion they expresses but also quantified them. Due to time and resource constraints, we focused on the

text caption from the images, the images themselves were not analyzed. Our methodology provided moderate accuracy values. In future we would apply different methods to precisely quantify the labels. And also will try to include the images in the actual analysis, which could provide much more accurate results.

REFERENCES

[1] N. Sonnadh. The world's biggest meme is the word "meme" itself. [Online]. Available: https://qz.com/1324334/memes-around-the-world-the-worlds-biggest-meme-is-the-word-meme-itself/

[2] "Ocr." [Online]. Available: https://engineering.fb.com/ai-research/rosetta-understanding-text-in-images-and-videos-with-machine-learning/

[3] "Semeval." [Online]. Available: http://alt.qcri.org/semeval2020/

[4] "memotion1." [Online]. Available: https://competitions.codalab.org/competitions/20629

[5] "memotion1." [Online]. Available: http://www.amitavadas.com/Memotion.html

[6] N. Gal, L. Shifman, and Z. Kampf, ""it gets better": Internet memes and the construction of collective identity," *New Media Society*, vol. 18, pp. 1698–1714, 2016.

[7] A. Williams, C. Oliver, K. Aumer, and C. Meyers, "Racial microaggressions and perceptions of internet memes," *Comput. Hum. Behav.*, vol. 63, no. C, pp. 424–432, Oct. 2016. [Online]. Available: https://doi.org/10.1016/j.chb.2016.05.067

[8] V. AbelL.Peirson and E. M. Tolunay, "Dank learning: Generating memes using deep neural networks," *ArXiv*, vol. abs/1806.04510, 2018.

[9] H. Gonçalo Oliveira, D. Costa, and A. Pinto, "One does not simply produce funny memes! – explorations on the automatic generation of internet humor," 06 2016.

[10] J. H. French, "Image-based memes as sentiment predictors," in *2017 International Conference on Information Society (i-Society)*, July 2017, pp. 80–85.

[11] G. Da San Martino, W. Gao, and F. Sebastiani, "Ordinal text quantification," in *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '16. New York, NY, USA: ACM, 2016, pp. 937–940. [Online]. Available: http://doi.acm.org/10.1145/2911451.2914749

[12] Wikipedia contributors, "Ranking svm — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Ranking$_SVMoldid = 915187246, 2019, [$Online; accessed 10 - December - 2019].

[13] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon, "Adapting ranking svm to document retrieval," in *SIGIR '06 Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, August 2006, pp. 186–193.