“Final Project”
Persian Sentiment Analysis for Marketing

Advanced Machine Learning

Professor: Dr. Farahani

Group: Mehrdad Baradaran -
Hossein Yahyaei -
Katayoun Kobraei

Table of contents:

1. Introduction to Dataset

Snappfood is an online food delivery company in Iran. This dataset consists user comments containing 70,000 comments with two labels (i.e. polarity classification): Happy , Sad

Label	#
Negative	35000
Positive	35000

It consists of 3 features : 1. The comment feature 2. A label with values of ‘sad’ and ‘happy’ 3. The label_id feature which 1 meaning ‘sad’ and 0 meaning happy

The dataset can be accessed via the provided link: https://hooshvare.github.io/docs/datasets/sa#snappfood.

2. Abstract

In this project, Persian Sentiment Analysis for Marketing, we worked on specifically a 2-class classification on a dataset sourced from Snappfood, an online food delivery company. The dataset consists of 70,000 user comments, with sentiments categorized into two classes: Happy, Sad. We use machine learning methods and more advanced methods like neural networks to complete this task. Actually we are tasked with developing models that accurately classify user comments into the specified sentiment categories to gain insights into customer satisfaction and sentiment trends for marketing purposes. We are prohibited from utilizing pre-trained models but we are allowed to incorporate supplementary data from other sources. Our primary evaluation metric for this task would be the weighted F1 score. For this task first we cleaned our data and then did some feature engineering processes to reach better performance in training any model.

3. Preprocessing Part

First of all we will use a library called ‘Hazm’ in this part. We need to install this library using pip. "Hazm" is a Python library designed for natural language processing (NLP) tasks in the Persian language. It provides a set of tools and utilities for working with Persian text, including tokenization, stemming, lemmatization, and more.

First we see some samples of dataset:

	Unnamed: 0	comment	label	label_id
0	0	واقعا حیف وقت که بنویسم سرویس دهیتون شده افتضاح	SAD	1
1	1	قرار بود ا ساعته برسه ولی نیم ساعت زودتر از مو...	HAPPY	0
2	2	قیمت این مدل اصلا با کیفیتش سازگاری نداره، فقط...	SAD	1
3	3	عاللللی بود همه چه درست و به اندازه و کیفیت خوب...	HAPPY	0
4	4	شیرینی وانیلی فقط یک مدل بود.	HAPPY	0

```
data['comment'][7498]
```

```
'ای بود و حلیم هم بد نبودu200cآش خوشمزه'
```

We see comments need some preprocessing before tokenization like deleting english words because it is a persian NLP task.
We implement some functions to clean data.
We need to map some character:

- First mapping is to convert all persian numbers to english numbers. Then we converts arabic characters like 'ؤ', 'ة', 'أ', 'إ', 'ئ', 'ي', 'ك' to 'و', 'ه', 'ا', 'ا', 'ی', 'ک'. We also convert these specific smileys to a token or a word. We convert all ":(", ":|", ":-|", ":-/", ":/", ":-@", ":@" , ":-(" to ‘ناراحت’ token and ":)", ":-)", ":-D", ":D", ";-)", ";)" to ‘خوشحال’ token and just delete other smileys from comments.
- Then we handle emojis. First we needed to detect emojis in comments using the emojis library. To handle emojis we prefer to just convert all emojis to a null string. In fact we did not do any sentiment analysis in emojis. We could actually find unicode of important meaningful emojis then find them in comment and interpret them to a token. But the processing took too long and it was not worth it for just a few comments.

- Next step we detect urls in the comments using URLExtract library and then convert all urls in comments to a null string.
- We needed to convert all characters to its lowercase form.
- We also deleted whitespaces using the ‘strip()’ function. This method is used to remove leading and trailing whitespace (spaces, tabs, and newline characters) from a string. Then using regex r'[\s]{2,}' we reduced multiple consecutive whitespace characters to a single space in all comments.
- Then we deleted all these punctuations '[<>#.:()"\'!?؟،,@$%^&*_+\[\]/]' to a null string.
- Next we reduced consecutive sequences of the same word character repeated two or more times to a single occurrence of that character. For example we convert ‘عالللللی بود’ to ‘عالی بود’.
- After all those mapping we checked if all characters were in the persian usicodes list ([\u0600-\u06FF]) to work with it.

Our dataset had an unnecessary and unmeaningful column named 'Unnamed: 0' so we needed to drop it from the dataset.

As there are not any null values in the dataset in this part we just pass it.

In the next step, determine some stop words in the Persian language. We could use ‘Hazm’ library to handle it but it took too long for it to find and convert them so we just defined these stop words :

```
stopwords = [
    "می" ,"ای" ,"اند" ,"ها" ,"بر" ,"تا" ,"آن" ,"برای" ,"های" ,"با" ,"را" ,"این" ,"که" ,"از" ,"به" ,"در" ,"و",
    "شما" ,"او" ,"نیز" ,"درباره" ,"یک" ,"دیگر" ,"ما",
    "اگرچه" ,"همچنین" ,"ولی" ,"البته" ,"پس" ,"بنابراین" ,"وقتی" ,"حتی" ,"چون" ,"توسط" ,"اما" ,"چه" ,"اگر" ,"باید",
    "اخیر" ,"اول" ,"دیروقت" ,"چه" ,"بیشتر" ,"تاکنون" ,"ولی" ,"هر" ,"همه" ,"همین" ,"یکدیگر",
]
```

And delete them. Stop words are commonly used words that are often filtered out from text data during the preprocessing stage. These words are considered to be of little value in terms of providing meaningful information for various NLP tasks so their removal can improve the efficiency of text processing and analysis.

We could implement a stemming function using ‘Hazm’ library and also a text normalizer but for that we needed a better computational power (bigger than kaggle) to do this part. So we just skipped this part and continued with other cleaning functions.

After all this we output the csv file of the new dataset with our changes to work with in the next part.

Tokenizers and Models:

As we know, machine learning models do not understand any kind of text. So we need to convert all these comments to a vector consisting of numbers to input them into models. There are various methods to vectorize texts in nlp tasks and we implemented some of them.

➢ Countvectorizer & logistic Regression

We could explain countvectorizer with this example:
Consider these texts with this bag of words. We see 10 unique words in total:

Text	Vector
worth it	[1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
I hate it	[0, 1, 1, 1, 0, 0, 0, 0, 0, 0]
If you like it, you like it	[0, 2, 0, 0, 1, 2, 2, 0, 0, 0]
Expensive, but awesome	[0, 0, 0, 0, 0, 0, 0, 1, 1, 1]

As we have 10 unique values we make a vector with length 10 for each data. Then we count how many times each word is repeated in that data and put the number of it in the place for that word in the vector.

After vectorization we split our dataset into train and test data with random state 42 and test size 0.2. This vectorization method is a basic and simple method that may work well on some tasks. For or task it worked almost good with this scores:

```
Accuracy: 84.30088495575221
Weighted F1 Score: 84.29362188037433
```

But we can do better with some advanced methods. Because count vectorization is biased to words repeatedly used in the text and if we do not handle punctuations and common words, models will not work accurately.

➢ TF-IDF Vectorizer & Logistic Regression

First we explain the TF-IDF method with an example. Consider this comments:

Text
worth it
I hate it
If you like it, you like it
Expensive, but awesome

We should calculate two value for every single value: TF and IDF

$$tf = \frac{number\ of\ times\ term\ appear\ in\ the\ documnet}{total\ number\ of\ terms\ in\ the\ document}$$

For example TF for word ‘it’ in data one (‘ worth it’) in 1/2 and in data three (‘if you like it, you like it’) in 2/7.

Now for the IDF value:

$$idf = log(\frac{number\ of\ documnets\ in\ the\ corpus}{number\ of\ documnets\ in\ the\ corpus\ contain\ the\ term})$$

Now for the ‘it’ word in data one (‘worth’) it IDF would be log of number of all documents in the corpus that is 4 divided by number of documents in the corpus containing the term ‘it’ which is 3.

$$tf(it) = \frac{1}{2} \qquad \text{idf}(it) = log\frac{4}{3}$$

All these values are calculated for each single word in each comment using the TfidfVectorizer() function. Then we split these vectors into test and train data using

train_test_split() function with random state 42 and test size 0.2. Then we input these datasets to a simple logistic regression model. This is the results:

```
Accuracy: 85.39823008849558
Weighted F1 Score: 85.37010320929892
```

➢ TF-IDF Vectorizer & Neural network

In this part we implemented some more advanced deep learning models like neural networks. We used a simple neural network first to see the accuracy. The model architecture consists of three hidden layers with 1000, 500, and 50 neurons, respectively, all using the ReLU activation function. Dropout layers are included after each hidden layer to prevent overfitting by randomly dropping out units during training. The output layer has two neurons (nb_classes = 2) with softmax activation, suitable for binary classification. The model is compiled with categorical cross-entropy loss and the Adam optimizer.

If we check the result it would be:

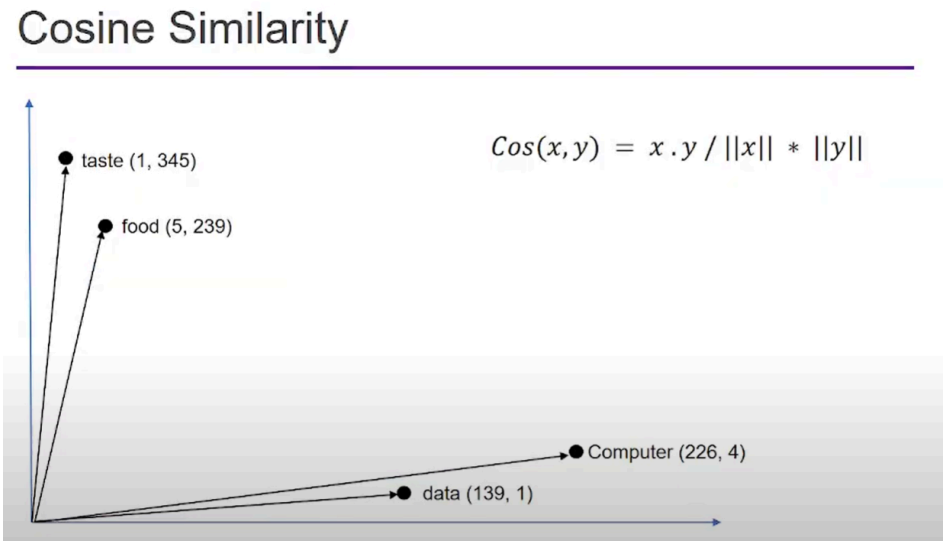
```
nDeep Neural Network - Test accuracy: 82.67
nDeep Neural Network - Train accuracy: 99.71
```

```
354/354 [==============================] - 3s 7ms/step
Deep Neural Network - Test F1 score: 82.67
```

We see our model is a little bit overfitted in the training data. We can simply tune hyperparameters or use early stopping or … and test again again to overcome this problem to reach higher scores but we prefer to use even more advanced methods both in model and vectorization.

➢ Word Embeddings & LSTM

Next advanced tokenizer is word embedding. But what is word embedding?
It converts words into vectors in a way that similar words have values close to each other in space. As an example, imagine we have only these four words in our vocabulary. For simplicity we show a two dimensional vector for each word.



Then we calculate similarity between words. We use cosine similarity.

$$Cos(taste, food) = \frac{(1*5)+(345*239)}{\sqrt{1^2+345^2}*\sqrt{5^2+239^2}} = \frac{82460}{82489} = 0.99$$

$$Cos(data, food) = \frac{(1*139)+(345*1)}{\sqrt{1^2+345^2}*\sqrt{1^2+139^2}} = \frac{484}{47955} = 0.01$$

So in this method we maintain a semantic connection between words using similarity.
Now we use tokenizer class from keras.preprocessing.text to tokenize words and we choose num_word=2000 to have 2000 most important words and maxlen=100 which means we choose a 100 dimensional vector for each word. So the input size would be (56498, 100). Then we split this input into test data and train with random state 42 and test size 0.2.
In this step we need to implement a LSTM model to get better results.
LSTM Architecture: 1.tf.keras.Sequential: This creates a linear stack of layers for building the model. 2.tf.keras.layers.Embedding: This layer is responsible

for embedding input sequences into dense vectors of fixed size.
3.tf.keras.layers.Bidirectional: This wrapper runs the input sequence both forward and backward through the LSTM layer and concatenates the outputs. Bidirectional LSTMs are useful for capturing information from both past and future contexts.
4.tf.keras.layers.LSTM: The LSTM layer, a type of recurrent layer, processes sequences and maintains context information. In this case, there are two LSTM layers, both wrapped in Bidirectional to capture bidirectional information.
5.tf.keras.layers.Dense: These are fully connected layers. The first dense layer has 64 units and uses the ReLU activation function. The second dense layer has 2 units with a softmax activation function, suitable for binary classification.

If we train this model we get this results:

```
nDeep Neural Network - Test accuracy: 85.03
nDeep Neural Network - Train accuracy: 89.64999999999999

Deep Neural Network - Test F1 score: 85.0
```

Next if we want to test our model with some random comments we see this result for ‘غذا خیلی خوب بود’ comment:

```
1/1 [==============================] - 0s 29ms/step
array([[0.90641296, 0.09358707]], dtype=float32)
```

This means 90% is a happy comment and 9% is a sad comment.

➢ Byte-Pair encoding tokenizer and LSTM

We tested a famous encoding called byte pair encoding. Byte Pair Encoding (BPE) is a compression algorithm that includes tokenization and subword segmentation. It was initially introduced for text compression but has found applications in various NLP tasks due to its effectiveness in handling rare words and out-of-vocabulary terms. Here's an overview of how BPE works:
First we calculate frequencies. we calculate the frequency of each symbol (character, subword, or word) in the dataset. Then we calculate pair frequency. we calculate the frequency of pairs of consecutive symbols. then we merge all together. we merge the most frequent pair of symbols into a new symbol. then we update the vocabulary with the merged symbol. We repeat this until a stopping criterion is met.

As an example let's consider a simple example with a vocabulary of characters:
Initial Vocabulary: {'a', 'b', 'c', 'd'}

Data: "abracadabra"

Step 1:
Symbol Frequency: {'a': 5, 'b': 2, 'c': 1, 'd': 1}
Pair Frequency: {('a', 'b'): 2, ('b', 'r'): 2, ('r', 'a'): 2, ('a', 'c'): 1, ('c', 'a'): 1, ('c', 'a'): 1, ('a', 'd'): 1, ('d', 'a'): 1, ('a', 'b'): 1}

Step 2:
Merge the most frequent pair: ('a', 'b')
Updated Vocabulary: {'ab', 'r', 'c', 'd'}

Then we repeat these steps until a desired vocabulary size or number of iterations is reached.

As we build input data we give it to a LSTM model which is a powerful neural network for simple nlp tasks. We get this result at first try and it

need more RAM if we want to do it for more iteration, but as our model is not overfitted we can reach better results in further attempts.

```
1766/1766 [==============================] - 4s 2ms/step
1766/1766 [==============================] - 4s 2ms/step
Deep Neural Network - Test accuracy: 58.96
Deep Neural Network - Train accuracy: 58.96
Deep Neural Network - Test F1 score: 58.040000000000006
```

5. Conclusion

In conclusion, in this task we explored various approaches for Persian Sentiment Analysis on the Snappfood dataset. While we reached reasonable results with traditional methods and basic neural networks, the integration of deep learning techniques, particularly Bidirectional LSTM models, showcased promising results with accuracy reaching 85.03%. However, the Byte-Pair Encoding tokenizer and LSTM model displayed comparatively lower accuracy at 58.96% which could be better with higher ram capacity. Of course we can enhance models' accuracy across the board in potential areas including hyperparameter tuning, addressing convergence warnings, and exploring more sophisticated neural network architectures. These insights contribute to a comprehensive understanding of sentiment analysis in the Persian language for marketing applications.