AML-2203 Advanced Python AI and ML tools

Final Project

Submitted To:

Vahid Hadavi

Team:

Mohammed Muzammil Lakdawala (C0872315)

Gurinder Singh (C0871566)

Maryam Shokrollahi (C0864597)

Mohammed Fuwad Anjum (C0867421)

Gursimran Kaur (C0867039)

Introduction

In recent years, social media has become a prominent platform for people to express their thoughts and opinions, and it has also become a vital source of information for businesses and individuals alike. Twitter, in particular, has gained immense popularity as a platform for news, politics, and entertainment. With over 330 million monthly active users, Twitter has become a valuable source of data for researchers and data scientists.

In this project, we aimed to predict the number of likes that a tweet from Elon Musk's Twitter account would receive. Elon Musk is a well-known entrepreneur, investor, and visionary who is known for his innovative ideas and outspoken opinions. His tweets often go viral, and we wanted to explore the factors that contribute to their popularity.

To achieve this goal, we collected data on Elon Musk's tweets and their corresponding number of likes using SNScrape. We then preprocessed the data and extracted relevant features such as the number of retweets, the length of the tweet, and the time of day it was posted. We used machine learning techniques such as regression analysis to build a predictive model to estimate the number of likes a tweet would receive.

The insights gained from this project could be useful for businesses and individuals who want to improve their social media strategy and increase their engagement on Twitter. By understanding the factors that contribute to tweet popularity, they can create more effective content and ultimately reach a wider audience.

In the following sections, we will discuss our data collection process, data preprocessing techniques, feature engineering, and model building process in detail. We will also present our results and discuss the implications of our findings.

Methodology

After scraping the data, it looks something like this

```
df.head()
```

✓ 0.0s Python

	Unnamed: 0	User	Date Created	Number of Likes	Source of Tweet	Tweet	ReTweet Count	Reply Count	View Count
0	0	elonmusk	2023-04-06 20:31:24+00:00	1418	Twitter for iPhone	@DirtyTesLa We are gradually reducing it, prop...	126	331	206454.0
1	1	elonmusk	2023-04-06 20:26:19+00:00	1174	Twitter for iPhone	@GRDecter Any highly successful investor with ...	84	142	59256.0
2	2	elonmusk	2023-04-06 20:17:48+00:00	1947	Twitter for iPhone	@GRDecter Probably Buffett. He could do it usi...	155	161	72716.0
3	3	elonmusk	2023-04-06 20:15:32+00:00	1604	Twitter for iPhone	@cb_doge Actually, will add this app to my nam...	114	283	78871.0
4	4	elonmusk	2023-04-06 20:13:36+00:00	9968	Twitter for iPhone	@KanekoaTheGreat Is this real?	1260	1524	1290796.0

We first prepare the data by cleaning and preprocessing the text and then made a function that creates a Bag of Words (BOW) representation of the data.

```
def prepareSentence(s):
    stemmer = LancasterStemmer()
    ignore_words = set(stopwords.words('english'))
    regpattern = re.compile('[\W_]+" "')
    s = re.sub('[^A-Za-z ]+', '', s)
    words = word_tokenize(s.lower())
    return [stemmer.stem(w.lower()) for w in words if w not in ignore_words]
```

✓ 0.1s

```
def toBOW(sentence, words):
    bag = []
    for word in words:
        bag.append(1) if word in sentence else bag.append(0)
    return bag
```

✓ 0.0s

Now we add a new column to the DataFrame called "Engagement" that is the sum of the "Number of Likes" and "ReTweet Count" columns.

```
df['Engagement'] = df['Number of Likes'] + df['ReTweet Count']
```

✓ 0.0s

We then extract the distinct words from the tweets using the prepareSentence function, which applies stemming, removes stop words, and removes non-alphabetic characters.

```
words = []

for index, tweet in df.iterrows():
    words.extend(prepareSentence(tweet['Tweet']))
print(len(words))
```

✓ 8.5s

```
167144
```

At this point we have too many words (1,67,144) to work with so we need to reduce it. So, we then set lower and upper threshold values for word counts and only include words that fall within this range.

```
distinct_words = set(words)

lower_threshold = 10
upper_threshold = 350
counts = []
final_words = []
for word in distinct_words:
    counts.append(words.count(word))
    if words.count(word) > lower_threshold and words.count(word) < upper_threshold:
        final_words.append(word)

print(len(words))
print(len(distinct_words))
print(len(final_words))
```

✓ 1m 8.3s

```
167144
19303
2124
```

The Bag of Words (BOW) representation is created by converting each tweet into a binary vector representing the presence or absence of the selected words.

We then split the data into inputs and outputs for training the neural network. Inputs are the BOW vectors and outputs are scores for each tweet that indicate whether it received any engagement (1) or not (0).

```
inputs = []
outputs = []

for index, tweet in df.iterrows():
    sentence = prepareSentence(tweet['Tweet'])
    bag = toBOW(sentence, final_words)
    inputs.append(bag)
    #Calculate a score, 1 if any engagement, 0 if none
    score = min(tweet["Engagement"],1)

    outputs.append(score)
```

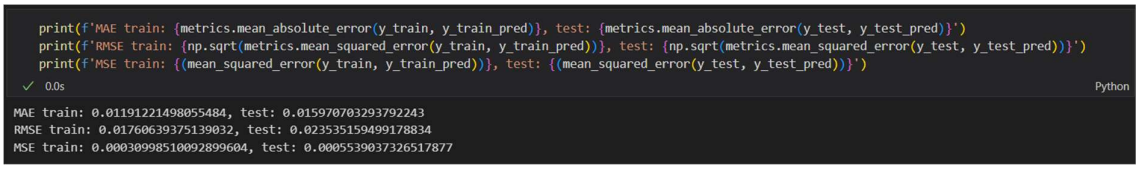
✓ 14.7s

The neural network (MLPRegressor) is then defined and trained using the training data with some of the following hyperparameters.

Finally, the trained neural network is used to predict the engagement of potential tweets in a list called feed. The potential_posts dictionary is used to store the predicted engagement score for each post, and the posts are sorted by their predicted engagement score to identify the top and bottom five posts.

```
Top 5 posts
("Cybertruck is going to be a game-changer in the world of electric trucks. Can't wait to start production!", 1.0163000704977478)
("Who needs a boring old house when you can live in a sustainable, high-tech Mars colony? #ColonizeMarsThe Falcon Heavy launch was absolutely epic. Can't wait
("The Hyperloop is going to change the way we travel forever. Who's ready for a 700 mph ride?", 1.0081185127648657)
('Boring Company flamethrowers are the ultimate tool for clearing brush and entertaining guests. #FlameOn', 1.0042685905810447)
('Just landed on Mars. Time to start building a new world!', 1.0040125732489746)
Bottom 5 posts
('Sustainable energy is the only way forward. We need to invest in clean technology now more than ever.', 0.980505997344562)
("SpaceX's Starlink internet service is going to connect the entire world. No more dead zones!", 0.9806338733758235)
("Life is too short to waste time on things that don't matter. Follow your dreams and make a difference.", 0.9876500442131579)
("If you're not pushing the limits of what's possible, you're not really living.", 0.9904203518552714)
('Neuralink is going to revolutionize the way we interact with technology. The possibilities are endless.', 0.991656579321063)
```

This was a success but to truly call this a success we need evidence and to provide it we calculated the errors for it. We calculated the Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Mean Square Error (MSE) and all of them are very low.



But this type of prediction is only useful to us when we have a list of tweets and we want to know which one will perform better than the others and it won't have the exact number of likes that tweet is going to receive. So, we did an alternative approach as well.

We first performed some preprocessing on the tweet column like removing the numbers, punctuation and small words as follows

```
# Removing Numbers From "Tweet" Column
df['Tweet'] =  [re.sub(r'\d+','', str(x)) for x in df['Tweet']]
```
✓ 0.1s

```
# Removing Punctuations From "Tweet" Column
df["Tweet"] = df['Tweet'].str.replace('[^\w\s]','')
```
✓ 0.1s

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_29376\2799319037.py:2: Fu
  df["Tweet"] = df['Tweet'].str.replace('[^\w\s]','')
```

```
# Removing Words In Tweets With Length Less Than 2
pat = re.compile(r'\b\w{1,2}\b')
df["Tweet"] = df["Tweet"].apply(lambda s: pat.sub('', s))
```
✓ 0.1s

We also performed sentiment analysis on it using SentimentIntensityAnalyzer which gives us 4 scores which are compound, negative, neutral and positive. We will be using this for training our model as well to get some more features.

Then, we used TFIDF Vectorizer on the tweet column, we set the max_features as 20,000 as we have a lot of tweets and going lower might cause us to lose a lot of our features.

```
# Using TFIDF Vectorizer
tfidf = TfidfVectorizer(max_features=20000, stop_words=stopwords.words('english'))
vectorized_data = tfidf.fit_transform(df['Tweet'].values)
```
✓ 0.3s

```
vectorized_dataframe = pd.DataFrame(vectorized_data.toarray(), index=df['Tweet'].index.tolist())
```
✓ 0.2s

We then used Truncated SVD to reduce the number of dimensions but we have to be careful as going too low might cause us to lose variance and a lot of information along with it.

```
# Using Truncated SVD to reduce the dimensions of the data
number_of_dimensions = 10000
svd = TruncatedSVD(n_components=number_of_dimensions)
reduced_data = svd.fit_transform(vectorized_dataframe)
print(reduced_data.shape)

# List of explained variances
tsvd_var_ratios = svd.explained_variance_ratio_

var_explained = svd.explained_variance_ratio_.sum()
print(var_explained)
```

✓ 25m 5.3s

```
(21601, 10000)
0.9661877116226325
```

We went with 10,000 features as this gave us 0.966 and when we tried with 5,000 rows, we were getting a variance of around 0.6 which was very low.

We then divide the DataFrame into target and features as follows

```
df_copy = df.copy(deep=True)
y = df_copy['Number of Likes']
x = df_copy[['neg', 'pos', 'compound' ,'neu']]
```

✓ 0.1s

```
x = pd.concat([x, vectorized_dataframe], axis=1, join="inner")
```

✓ 7.8s

Here, x are the features and y is the target.

We wanted to perform Polynomial Feature Extraction on it but due to hardware restrictions we were unable to do so as only a degree of 2 was requiring memory of 22 Terabytes.

```
poly = PolynomialFeatures(degree=2, include_bias=False)
x_train = poly.fit_transform(x_train)
x_test = poly.transform(x_test)
```

1.2s Python

```
c:\Users\Admin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\validation.py:1858: FutureWarning: Feature names only support names that are all strings. Got feature names wit
  warnings.warn(
c:\Users\Admin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\validation.py:1858: FutureWarning: Feature names only support names that are all strings. Got feature names wit
  warnings.warn(

Output exceeds the size limit. Open the full output data in a text editor
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
c:\Users\Admin\Desktop\Sem 2\AML 2304 2 Natural Language Processing\Assignment 2\project.ipynb Cell 39 in 2
      1 poly = PolynomialFeatures(degree=2, include_bias=False)
----> 2 x_train = poly.fit_transform(x_train)
      3 x_test = poly.transform(x_test)

File c:\Users\Admin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:867, in TransformerMixin.fit_transform(self, X, y, **fit_params)
    863 # non-optimized default implementation; override when a better
    864 # method is possible for a given clustering algorithm
    865 if y is None:
    866     # fit method of arity 1 (unsupervised transformation)
--> 867     return self.fit(X, **fit_params).transform(X)
    868 else:
    869     # fit method of arity 2 (supervised transformation)
    870     return self.fit(X, y, **fit_params).transform(X)

File c:\Users\Admin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\preprocessing\_polynomial.py:432, in PolynomialFeatures.transform(self, X)
    428     XP = sparse.hstack(columns, dtype=X.dtype).tocsc()
    429 else:
    430     # Do as if _min_degree = 0 and cut down array after the
    431     # computation, i.e. use _n_out_full instead of n_output_features_.
--> 432     XP = np.empty(
    433         shape=(n_samples, self._n_out_full), dtype=X.dtype, order=self.order
    434     )
...
    448
    449     # degree 0 term
    450     if self.include_bias:

MemoryError: Unable to allocate 22.0 TiB for an array with shape (15120, 200110014) and data type float64
```

So, we moved on to training our Regression model.

We trained our Random Forest Regressor on the train set.

```
regr = RandomForestRegressor(max_depth=2, random_state=0)
regr.fit(x_train, y_train)
```

✓ 1m 39.6s

```
c:\Users\Admin\AppData\Local\Programs\Python\Python310\lib\site
  warnings.warn(
```

RandomForestRegressor

```
RandomForestRegressor(max_depth=2, random_state=0)
```

The metrics that we got for it are as follows

```
print(f'MAE train: {metrics.mean_absolute_error(y_train, y_train_pred)}, test: {metrics.mean_absolute_error(y_test, y_test_pred)}')
print(f'RMSE train: {np.sqrt(metrics.mean_squared_error(y_train, y_train_pred))}, test: {np.sqrt(metrics.mean_squared_error(y_test, y_test_pred))}')
print(f'R^2 train: {(r2_score(y_train, y_train_pred))}, test: {(r2_score(y_test, y_test_pred))}')
print(f'MSE train: {(mean_squared_error(y_train, y_train_pred))}, test: {(mean_squared_error(y_test, y_test_pred))}')
```

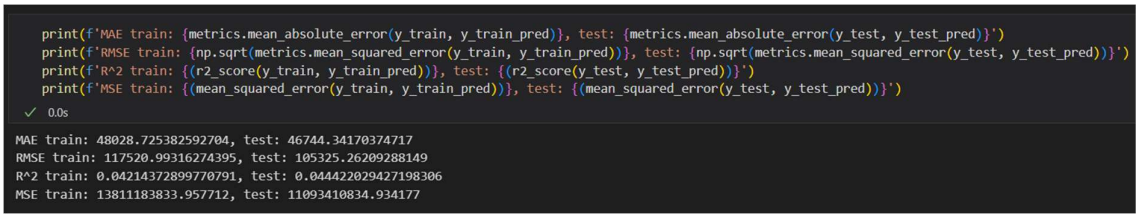
✓ 0.0s

```
MAE train: 51169.32904514804, test: 50413.39415588119
RMSE train: 110654.17187098303, test: 107664.16521526489
R^2 train: 0.1508096985523517, test: 0.001510765004574921
MSE train: 12244345752.453053, test: 11591572471.499855
```

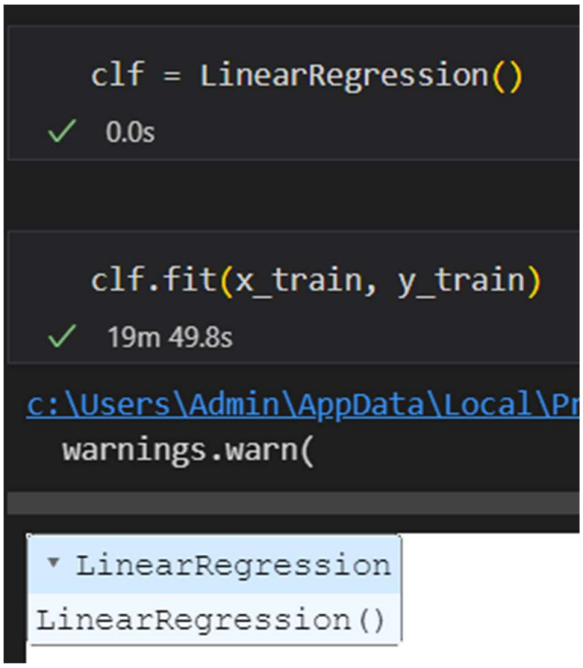
The R^2 score was only 0.15 for the train and 0.0015 for the test and we thought it was because of the model that we used so we trained some more model.

We started off with the same MLP Regression neural network as before.

But we got even worse results with it as the R^2 score was even lower at 0.042 for training and 0.044 for the test. Although here the score was higher for the test but it was still extremely low.



At this point we understood that the complexity of the model was way too much for the data we were training it for so, we tried using a simpler model like Linear Regression.



And the R^2 score for the train set came out as 0.796 which was an extreme delight to us as we thought that the model finally learned something but at the same time the score for the test set came out as -6.58 which meant that the model became severely overfit.

```
print(f'MAE train: {metrics.mean_absolute_error(y_train, y_train_pred)}, test: {metrics.mean_absolute_error(y_test, y_test_pred)}')
print(f'RMSE train: {np.sqrt(metrics.mean_squared_error(y_train, y_train_pred))}, test: {np.sqrt(metrics.mean_squared_error(y_test, y_test_pred))}')
print(f'R^2 train: {(r2_score(y_train, y_train_pred))}, test: {(r2_score(y_test, y_test_pred))}')
print(f'MSE train: {(mean_squared_error(y_train, y_train_pred))}, test: {(mean_squared_error(y_test, y_test_pred))}')
```

✓ 0.0s

```
MAE train: 24700.946257073476, test: 6.2021337090668744e+16
RMSE train: 54221.64044023593, test: 2.7640308401127754e+17
R^2 train: 0.7961011640755955, test: -6.580922873859397e+24
MSE train: 2939986292.0302286, test: 7.639866485094535e+34
```

Adding something of only slight relevance, we even tried using XGBoost but it came as a failure as well, as it was running for more than 3 hours and did not finish training.

```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0).fit(x_train, y_train)
clf.score(x_test, y_test)
```

⊗ 205m 51.7s

Conclusion

In conclusion, we were successful in scoring which tweets are more likely to perform better with respect to other tweets but were unsuccessful when it came to predicting their likes with concrete accuracy.

Some of the reasons for the low R^2 score can be presence of outliers in the data, too less features considered while performing vectorization and lack of Polynomial Features while training the model.

But overall this has been a good learning experience for us as to where to look for when dealing with such underfit models and understanding that sometimes lack of hardware might also be an issue.

References

Yener, Y. (2020, November 7). *Step by step: Twitter sentiment analysis in Python*. Medium. Retrieved April 20, 2023, from https://towardsdatascience.com/step-by-step-twitter-sentiment-analysis-in-python-d6f650ade58d

Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

Koehrsen, W. (2018, January 28). Overfitting vs. underfitting: A complete example. Medium. Retrieved April 20, 2023, from https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765