PRESENTATION

SENTIMENT ANALYSIS

Task 1

Make use of Logistic Regression model from scikit-learn or some other packages in Python, run the Sentiment Analysis solution again and make a very thorough comparison with what we implemented from scratch.

```
train_x_vec = np.zeros((len(train_x), 3))
for i in range(len(train_x)):
    train_x_vec[i, :] = extract_features(train_x[i], freqs)

test_x_vec = np.zeros((len(test_x), 3))
for i in range(len(test_x)):
    test_x_vec[i, :] = extract_features(test_x[i], freqs)
```

```
model = LogisticRegression()
model.fit(train_x_vec, train_y.ravel())
```

TRAINING: The logistic regression model is trained using the fit() method on the vectorized training data (train_x_vec).

Task 1 **PREDICTION**

Sklearn:

```
y_pred = model.predict(test_x_vec)
```

Predictions are made using the predict() method

Custom-built:

```
def predict_tweet(tweet, freqs, theta):
    '''
    Input:
        tweet: a string
        freqs: a dictionary corresponding to the frequencies of each tuple (word, label)
        theta: (3,1) vector of weights
    Output:
        y_pred: the probability of a tweet being positive or negative
    '''

    # extract the features of the tweet and store it into x
    x = extract_features(tweet, freqs)
    # make the prediction using x and theta
    y_pred = sigmoid(np.dot(x, theta))

    return y_pred
```

The predict_tweet function (not fully shown here) calculates the prediction for each tweet using the logistic function (sigmoid).

Task 1

Evaluation: The accuracy and a classification report (precision, recall, f1-score) are calculated. Both methods achieve roughly same accuracy of 0.9945

```
Accuracy: 0.994
              precision    recall  f1-score   support

    Negative       0.99      0.99      0.99      1000
    Positive       0.99      0.99      0.99      1000

    accuracy                           0.99      2000
   macro avg       0.99      0.99      0.99      2000
weighted avg       0.99      0.99      0.99      2000
```

Sklearn

```
Logistic regression model's accuracy = 0.9945
```

Custom-built

Custom-built:

* Use your 'predict_tweet' function to make predictions on each tweet in the test set.
* If the prediction is > 0.5, set the model's classification 'y_hat' to 1, otherwise set the model's classification 'y_hat' to 0. 0.5 plays a role of the decision threshold here.
* A prediction is accurate when the y_hat equals the test_y. Sum up all the instances when they are equal and divide by n.

Task 1

ERROR ANALYSIS:

Sklearn

```
Label   Predicted Probability   Processed Tweet
THE TWEET IS: @phenomyoutube u probs had more fun with david than me : (
THE PROCESSED TWEET IS: ['u', 'prob', 'fun', 'david']
0       0.56764160      b'u prob fun david'
THE TWEET IS: pats jay : (
THE PROCESSED TWEET IS: ['pat', 'jay']
0       0.59072992      b'pat jay'
THE TWEET IS: the internet is being a total bitch : (
THE PROCESSED TWEET IS: ['internet', 'total', 'bitch']
0       0.59092234      b'internet total bitch'
THE TWEET IS: my beloved grandmother : ( https://t.co/wt4oXq5xCf
THE PROCESSED TWEET IS: ['belov', 'grandmoth']
0       0.58913102      b'belov grandmoth'
THE TWEET IS: @CHEDA_KHAN Thats life. I get calls from people I havent seen in 20 years and its always favours : (
THE PROCESSED TWEET IS: ['that', 'life', 'get', 'call', 'peopl', 'havent', 'seen', '20', 'year', 'alway', 'favour']
0       0.57994632      b'that life get call peopl havent seen 20 year alway favour'
THE TWEET IS: Sr. Financial Analyst - Expedia, Inc.: (#Bellevue, WA) http://t.co/ktknMhvwCI #Finance #ExpediaJobs #Job #Jobs #Hiring
THE PROCESSED TWEET IS: ['sr', 'financi', 'analyst', 'expedia', 'inc', 'bellevu', 'wa', 'financ', 'expediajob', 'job', 'job', 'hire']
0       0.62904904      b'sr financi analyst expedia inc bellevu wa financ expediajob job job hire'
Tweet sau khi xử lý: hope first nlp session bore
Xác suất dự đoán cho nhãn 1: 0.6647773099381856
Tâm trạng: Tích cực
```

Custom-built

```
THE PROCESSED TWEET IS: ["i'm", 'play', 'brain', 'dot', 'braindot']
1       0.47906383      b"i'm play brain dot braindot"
THE TWEET IS: I'm playing Brain Dots : ) #BrainDots http://t.co/aOKldo3GMj http://t.co/xWCM9qyRG5
THE PROCESSED TWEET IS: ["i'm", 'play', 'brain', 'dot', 'braindot']
1       0.47906383      b"i'm play brain dot braindot"
THE TWEET IS: I'm playing Brain Dots : ) #BrainDots http://t.co/R2JBO8iNww http://t.co/oW5BBwdEMY
THE PROCESSED TWEET IS: ["i'm", 'play', 'brain', 'dot', 'braindot']
1       0.47906383      b"i'm play brain dot braindot"
THE TWEET IS: @msarosh Uff Itna Miss karhy thy ap :p
THE PROCESSED TWEET IS: ['uff', 'itna', 'miss', 'karhi', 'thi', 'ap', ':p']
1       0.47156052      b'uff itna miss karhi thi ap :p'
THE TWEET IS: @phenomyoutube u probs had more fun with david than me : (
THE PROCESSED TWEET IS: ['u', 'prob', 'fun', 'david']
0       0.53271796      b'u prob fun david'
THE TWEET IS: pats jay : (
THE PROCESSED TWEET IS: ['pat', 'jay']
0       0.50095618      b'pat jay'
THE TWEET IS: @bae_ts WHATEVER STIL L YOUNG &gt;:-(
THE PROCESSED TWEET IS: ['whatev', 'stil', 'l', 'young', '>:-(']
0       0.50033235      b'whatev stil l young >:-('
THE TWEET IS: my beloved grandmother : ( https://t.co/wt4oXq5xCf
THE PROCESSED TWEET IS: ['belov', 'grandmoth']
0       0.50000008      b'belov grandmoth'
THE TWEET IS: @CHEDA_KHAN Thats life. I get calls from people I havent seen in 20 years and its always favours : (
THE PROCESSED TWEET IS: ['that', 'life', 'get', 'call', 'peopl', 'havent', 'seen', '20', 'year', 'alway', 'favour']
0       0.50604028      b'that life get call peopl havent seen 20 year alway favour'
THE TWEET IS: Sr. Financial Analyst - Expedia, Inc.: (#Bellevue, WA) http://t.co/ktknMhvwCI #Finance #ExpediaJobs #Job #Jobs #Hiring
THE PROCESSED TWEET IS: ['sr', 'financi', 'analyst', 'expedia', 'inc', 'bellevu', 'wa', 'financ', 'expediajob', 'job', 'job', 'hire']
0       0.51648276      b'sr financi analyst expedia inc bellevu wa financ expediajob job job hire'
THE TWEET IS: @ITVCentral #Midlands Yes thanks for the depressing weather forecast, where the word 'rain' was mentioned several times :-(
THE PROCESSED TWEET IS: ['midland', 'ye', 'thank', 'depress', 'weather', 'forecast', 'word', 'rain', 'mention', 'sever', 'time', ':-(']
0       0.53599423      b'midland ye thank depress weather forecast word rain mention sever time :-('
```

Task 1

ERROR ANALYSIS:

- Processing Consistency: Both the custom-built and scikit-learn logistic regression models preprocess tweets the same way.
- Prediction Differences: The models produce slightly different probability scores, especially around the decision boundary of 0.5, leading to some tweets being classified differently.
- Model Implementation: The custom-built model and scikit-learn differ in how they handle logistic regression, which explains the small variations in predictions.
- Practical Impact: While the models generally align, discrepancies near the decision threshold could affect the final sentiment classification of some tweets.

Task 1

Advantages:

- Ease of Use: The implementation is straightforward with minimal code, leveraging optimized libraries.
- Performance: Likely to be faster due to the highly optimized nature of sklearn.
- Comprehensive Metrics: Easily generates a detailed classification report.

Disadvantages:

- Abstraction: Provides less insight into the underlying mechanics of the algorithm.

Task 2

In this line of code, if we change the number of iterations to, say, 100K, you might get some "divided by zero" error. Explain why and find a correction.

```
gradientDescent(X, Y, np.zeros((3, 1)), 1e-9, 10000)
```

```
for i in range(0, num_iters):

    # get z, the dot product of x and theta
    z = np.dot(x, theta)

    # get the sigmoid of z
    h = sigmoid(z)

    # calculate the cost function
    J = - (np.dot(y.T, np.log(h)) + np.dot((1-y).T, np.log(1-h))) / float(m)
    losses.append(float(J))
    # update the weights theta
    theta = theta - (alpha * np.dot(x.T, (h-y))) / float(m)
```

$$w = w - \frac{\alpha}{m} \times \left(X^T \cdot \left(y_{\text{pred}} - y\right)\right)$$

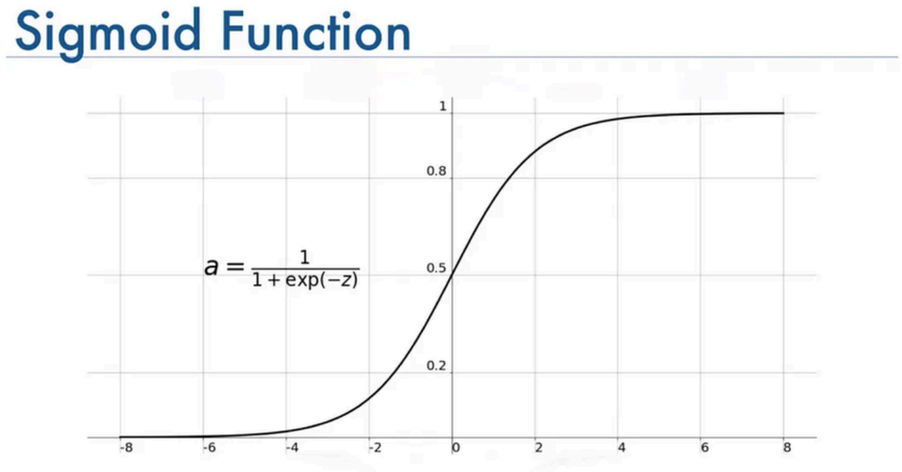
When the number of iterations (num_iters) is large, weight becomes very large or very small

Task 2

...weights very large or very small make the logit z
also be very large or very small...

$$z = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

...and make the h = sigmoid(z)
converges to 0 or 1...




```
# calculate the cost function
J = - (np.dot(y.T, np.log(h)) + np.dot((1-y).T, np.log(1-h))) / float(m)
```

```
np.log(0)
```

```
C:\Users\Latitude E7490\AppData\Local\Temp\ipykernel_13516\2933082444.py:1: RuntimeWarning: divide by zero encountered in log
  np.log(0)
```

```
-inf
```

...and make log(h) or log(1-h)
becomes log(0), which is
mathematically impossible to compute.

Task 2

```
# Adjust the cost function to prevent log(0) errors
J = - (np.dot(y.T, np.log(h + epsilon)) + np.dot((1-y).T, np.log(1-h + epsilon))) / float(m)
```

Correction:

- Add a Small Epsilon to the Logarithm: To prevent division by zero or taking the log of 0, a small Epsilon value can be added inside the log function to keep the value away from the extreme ends (0 or 1).

- Increase Learning Rate: Another approach is to increase the learning rate slightly to avoid very slow convergence, reducing the risk of accumulating numerical errors.

- Explicitly converting the cost (J) to a scalar using .item() before storing it in the losses list. This ensures that J is treated as a single numerical value, which might be crucial for downstream processes or comparisons.

Task 3

In Feature Engineering part, given a sentence s, we build two features: the positive frequency of s and the negative frequency of s. Given a sentence s, normalize these two features with respect to N = train_set_length * the length of s. Compare your result with the original one in the course. Is that normalization a good thing to do?

original:

```
great -> 0.538789
great great -> 0.577115
great great great -> 0.614533
great great great great -> 0.650646
```

after normalization:

```
great -> 0.742688
great great -> 0.742688
great great great -> 0.742688
great great great great -> 0.742688
```

Frequency of a word in a tweet does not affect the prediction probability after normalization

Task 3

```
# N = train_set_length * the length of s
N = train_set_length * len(word_l)
```

```
for word in word_l:

    # increment the word count for the positive label 1
    if (word, 1) in freqs.keys():
        x[0,1] += freqs[(word, 1)]

    # increment the word count for the negative label 0
    if (word, 0) in freqs.keys():
        x[0,2] += freqs[(word, 0)]

x[0,1] = x[0,1] / N
x[0,2] = x[0,2] / N
```

Assume ("great", 1) have value i in freqs dictionary, that means "great" appears i times in all positive tweets.

for "great":
x[0,1] = i/(train_set_length * 1)

for "great great great great":
x[0,1] = 4i/(train_set_length * 4)

So x[0,1] are the same in these 2 cases!
Similarly, x[0,2] in 2 cases are also the same since it refers to the negative frequency.

So, normalization seems not to be a good thing to do in sentimental analysis, since the repetition of a positive or negative word in a tweet matter in this context.

Task 4 **Standard Scaler**

Apply standard scaling techniques such as min-max scaler, standard scaler, etc to the problem and comment.

```
I am happy -> 0.546506
I am bad -> 0.489496
this movie should have been great. -> 0.539491
great -> 0.538789
great great -> 0.577115
great great great -> 0.614533
great great great great -> 0.650646
```

```
I am happy -> 0.500010
I am bad -> 0.499976
this movie should have been great. -> 0.500006
great -> 0.500005
great great -> 0.500029
great great great -> 0.500054
great great great great -> 0.500078
```

The Probability of a tweet being positive or negative before and after using Standard Scaling

If the original features “x_i” had a large range of values, their contribution to “z” could be significant. After scaling, however, **these contributions are much smaller because “x_i’” values are centered around 0 and have a standard deviation of 1**. Consequently, **the product “w_ix_i’” for each feature “x_i” might become very small**, especially if the corresponding weight “w_i” is not large.

Thus, the overall sum “z” (which is the linear combination of the weighted features) is likely to be close to 0, especially if the bias term “w_0” is also small. **The output y_pred=σ(z) will be close to 0.5 for most inputs**.

Task 4 MinMax Scaler

```
I am happy -> 0.546506
I am bad -> 0.489496
this movie should have been great. -> 0.539491
great -> 0.538789
great great -> 0.577115
great great great -> 0.614533
great great great great -> 0.650646
```

```
I am happy -> 0.500000
I am bad -> 0.500000
this movie should have been great. -> 0.500000
great -> 0.500000
great great -> 0.500000
great great great -> 0.500000
great great great great -> 0.500000
```

The Probability of a tweet being positive or negative before and after using MinMax Scaling

This scaling technique transforms features to a specific range (by default, [0, 1]). If features were already close in magnitude, **applying MinMax scaling could compress them further, leading to very small or even identical values** for the dot product $z=w0+w1x1+w2x2+…+wnxn$. When passed through the sigmoid function, **these small values result in σ(z) being exactly 0.5**.

Standard Scaling transforms the features to have a mean of 0 and a standard deviation of 1, preserving **more of the variation in the data**. This allows the model to maintain more of the discriminative power of the features, leading to more varied “z” values and, therefore, more meaningful predictions.

Task 4

Scenario	Initial Loss	Final Loss	Weight Magnitude	Test Accuracy	Comments
Baseline (No Scaling)	0.69	0.67	Small (e.g., 10^-7)	99.44%	High accuracy, but slow convergence. Weights remain small, suggesting conservative learning.
Standard Scaling	0.69	0.10	Moderate (e.g., 10^-3)	99.40%	Improved convergence and larger weights. Slightly lower accuracy but better-converged model.
Min-Max Scaling	8.0	6.0	Very Small (e.g., 10^-7)	99.05%	Poor convergence, with very small weights. Lowest accuracy, indicating ineffective learning.

- **Standard Scaling is the best overall performer**, providing a balance between effective learning and accuracy. Although the accuracy was slightly lower than the baseline, the model's robustness (as indicated by well-converged weights) makes it preferable.
- **The baseline model** (No Scaling) suggests that in some cases, particularly where we need to adjust the threshold or using prediction probability.

Task 5

5. During the session, I gave a natural way to end up with Logistic Regression as the decision function for our Sentiment Analysis problem. Let's come up with another decision function. Let

$$g(s) = \begin{cases} 1 \; if \; positive \; frequency > negative \; frequency \\ 0 \; otherwise \end{cases}$$

Note that g(s) =1 means the sentence s has positive sentiment.

With such mathematical formulation, recompute the precision on the test set. Compare and Explain.

Task 5

Logistic Regression Precision:

$$\text{decision}(x) = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Relies on a trained logistic regression model to predict tweet sentiment.

Frequency-Based Precision:

$$g(s) = \begin{cases} 1 \; if \; positive \; frequency > negative \; frequency \\ 0 \; otherwise \end{cases}$$

Uses the relative frequency of words in positive and negative tweets to classify sentiment.

Task 5

Compute precision on test set of 2 mathematical formulation

```python
def compute_precision_logistic_regression(test_x, test_y, freqs, theta):
    true_positive = 0
    false_positive = 0
    for tweet, actual in zip(test_x, test_y):
        predicted = predict_tweet(tweet, freqs, theta)
        if predicted > 0.5:  # classified as positive
            if actual == 1:
                true_positive += 1
            else:
                false_positive += 1
    precision = true_positive / (true_positive + false_positive)
    return precision
```

```python
def compute_precision(test_x, test_y, freqs):
    """
    Compute the precision using the new decision function.
    """
    true_positive = 0
    false_positive = 0
    for tweet, actual in zip(test_x, test_y):
        predicted = classify_tweet(tweet, freqs)
        if predicted == 1:
            if actual == 1:
                true_positive += 1
            else:
                false_positive += 1
    precision = true_positive / (true_positive + false_positive)
    return precision
```

Task 5

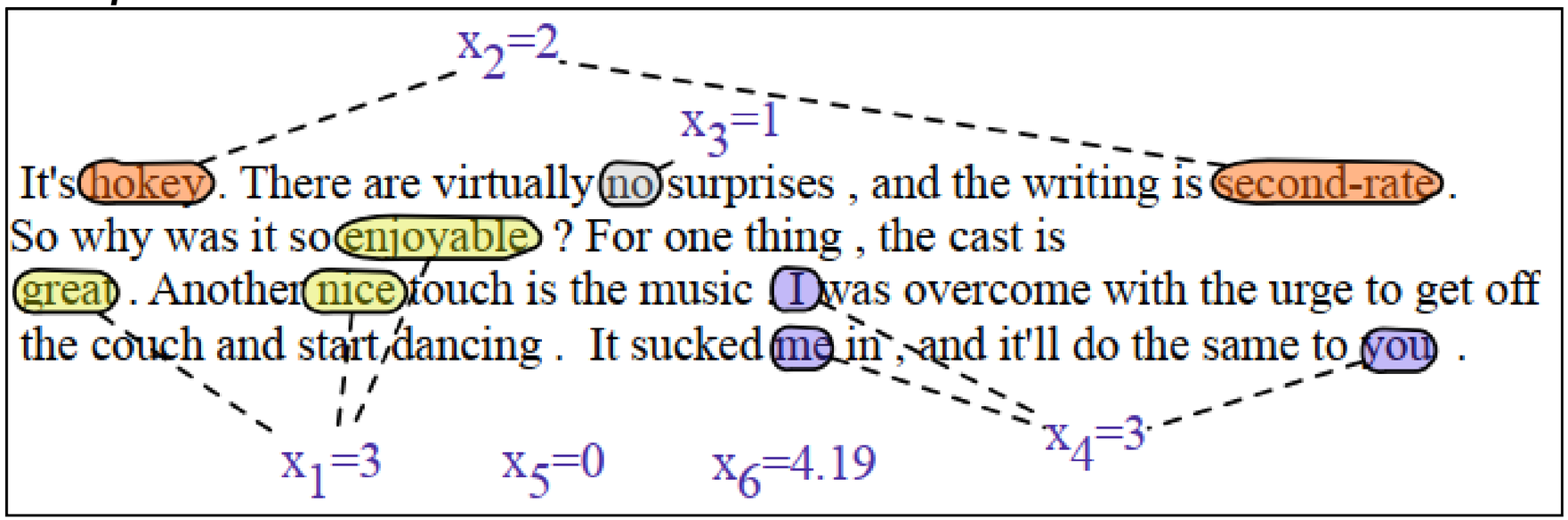
CRITERIA	LOGISTIC REGRESSION	FREQUENCY-BASED DECISION FUNCTION
Precision	**0.9930**	**0.9960**
Sensitivity	Sensitive to feature interactions and correlations	Less sensitive to nuances, purely based on the count of positive vs. negative words.
Model Complexity	Uses a weighted linear combination of features	Simpler, based on word frequency comparison.
Overfitting Risk	Moderate; requires regularization to avoid overfitting	Low; less risk due to its simple rule-based approach
Interpretability	Moderate; weights provide some insight, but model behavior is complex.	High; easy to interpret through direct word frequency comparison.
Generalization to New Data	Better generalization to diverse data sets due to beyond learning patterns	May struggle with complex data due to lack of nuance.

Task 6

Rerun the solution with 4 more features (6 in total) as follows:

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	ln(word count of doc)	$\ln(66) = 4.19$

Example



Task 6

MODEL	2 FEATURES	6 FEATURES
Logistic Regression	0.994	0.9935
Naive Bayes	0.969	0.9765
SVM	0.9925	0.993
Decision Tree	0.99	0.99
Random Forest	0.994	0.994
LightGBM	0.9925	0.994
XGBoost	0.993	0.993

All in all, adding features does improve the results, though does not make a new peak.

Task 7

In your opinion, can we do better? I propose that your team try all ML models that you know and give us the model with the best possible precision.

RANK	MODEL	PRECISION
1	Logistic Regression	0.993
1	Random Forest	0.993
3	XGBoost	0.993
4	LightGBM	0.992
5	SVM	0.99
6	Decision Tree	0.989
7	Naive Bayes	0.969

Overall, Logistic Regression and Random Forest showcase the best precision

Task 7

CAN WE DO BETTER?

When performing Error Analysis, it is observed that each model has its own limitations. Besides, there are also errors in 8 similar sentences ~0.4%. So ensembling results should improve score to 99.6%.

Some icons are not well-extracted

```
THE TWEET IS:            I'm playing Brain Dots : ) #BrainDots http://t.co/R2JBO8iNww h
THE PROCESSED TWEET IS: ["i'm", 'play', 'brain', 'dot', 'braindot']
Class: 1        Prob: 0.3342
_____________________________________________________________________________________
THE TWEET IS:            @msarosh Uff Itna Miss karhy thy ap :p
THE PROCESSED TWEET IS: ['uff', 'itna', 'miss', 'karhi', 'thi', 'ap', ':p']
Class: 1        Prob: 0.3228
_____________________________________________________________________________________
THE TWEET IS:            @phenomyoutube u probs had more fun with david than me : (
THE PROCESSED TWEET IS: ['u', 'prob', 'fun', 'david']
Class: 0        Prob: 0.2700
_____________________________________________________________________________________
THE TWEET IS:            @bumkeyyfel b-butt : ( isn't black cat a bad luck ene
THE PROCESSED TWEET IS: ['b-butt', 'black', 'cat', 'bad', 'luck', 'ene']
Class: 0        Prob: 0.4594
_____________________________________________________________________________________
THE TWEET IS:            pats jay : (
THE PROCESSED TWEET IS: ['pat', 'jay']
Class: 0        Prob: 0.3658
_____________________________________________________________________________________
THE TWEET IS:            @bae_ts WHATEVER STIL L YOUNG &gt;:-(
THE PROCESSED TWEET IS: ['whatev', 'stil', 'l', 'young', '>:-(']
Class: 0        Prob: 0.3736
_____________________________________________________________________________________
THE TWEET IS:            the internet is being a total bitch : (
THE PROCESSED TWEET IS: ['internet', 'total', 'bitch']
Class: 0        Prob: 0.4033
_____________________________________________________________________________________
THE TWEET IS:            my beloved grandmother : ( https://t.co/wt4oXq5xCf
THE PROCESSED TWEET IS: ['belov', 'grandmoth']
Class: 0        Prob: 0.3716
```

‘Neutral’ tweet

```
_____________________________________________________________________________________
THE TWEET IS:            Sr. Financial Analyst - Expedia, Inc.: (#Bellevue, WA) http://t.co/ktknMhvwCI #Finance #ExpediaJobs #Job #Jobs #Hiring
THE PROCESSED TWEET IS: ['sr', 'financi', 'analyst', 'expedia', 'inc', 'bellevu', 'wa', 'financ', 'expediajob', 'job', 'job', 'hire']
Class: 0        Prob: 0.2876
_____________________________________________________________________________________
```

Task 7

After applying new processing method, there were only 3 incorrect sentences with 99.85% accuracy. Of which, 1 was very long and 2 used unclear language.

```
[ ]  ys_pred = sa_with_ea.lr_pred_proba
     for x, y_true, y_pred in zip(sa_with_ea.tweets_test, sa_with_ea.y_test, ys_pred):
         if y_true != y_pred.argmax(axis =0):
             print('THE TWEET IS:\t\t', x)
             print('THE PROCESSED TWEET IS:', sa_with_ea.process_tweet(x))
             print('Class: %d\tProb: %0.4f' % (y_true, y_pred[int(y_true)]))
             print('_____________________________________________'*3)
```

```
THE TWEET IS:           @MarkBreech Not sure it would be good thing 4 my bottom daring 2 say 2 Miss B but Im gonna
THE PROCESSED TWEET IS: ['sure', 'would', 'good', 'thing', '4', 'bottom', 'dare', '2', 'say', '2', 'miss', 'b', 'im'
Class: 1        Prob: 0.3010
_____________________________________________________________________________________________________________________
THE TWEET IS:           @msarosh Uff Itna Miss karhy thy ap :p
THE PROCESSED TWEET IS: ['uff', 'itna', 'miss', 'karhi', 'thi', 'ap', ':p']
Class: 1        Prob: 0.3298
_____________________________________________________________________________________________________________________
THE TWEET IS:           @bae_ts WHATEVER STIL L YOUNG &gt;:-(
THE PROCESSED TWEET IS: ['whatev', 'stil', 'l', 'young', '>:-(']
Class: 0        Prob: 0.3471
_____________________________________________________________________________________________________________________
```

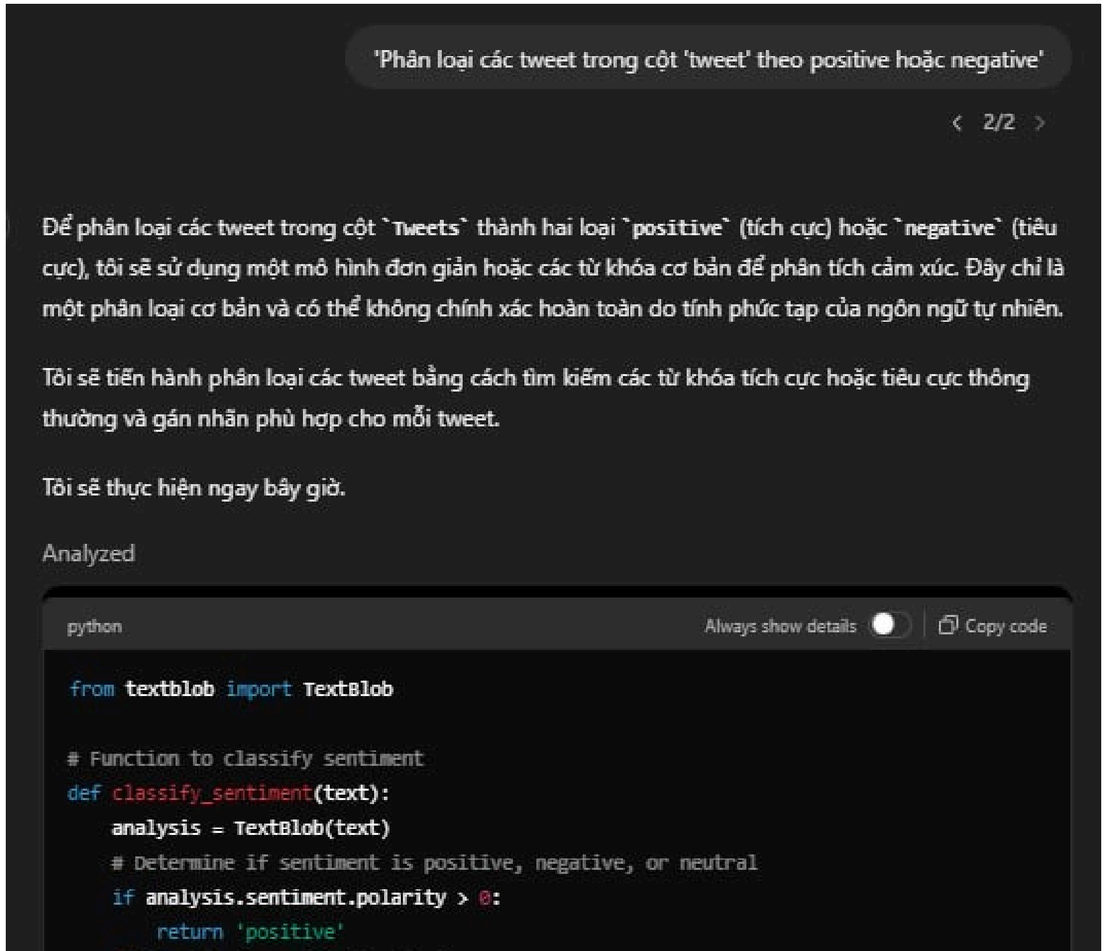
Task 7

HUMAN-LEVEL PERFORMANCE BENCHMARK (HLP)

- In a machine learning project, especially to classification model, we have to set our target, expectation through metrics, like 70%? 90%? or 100% accuracy? How we get this number?
- HLP works with unstructured data and since we are using the ML model, this might not be true.
- The achieved result is quite close to the goal.

Task 8

We are in 2024 right now, so let's use ChatGPT 3.5 Turbo or ChatGPT 4/4o as the benchmark and find a way to run the test set in your course with ChatGPT to determine the sentiment. What is your conclusion?



- Not using prompt techniques
- Inconsistent performance: 96,15%, 96,25%, 99,54% accuracy

Task 8

We are in 2024 right now, so let's use ChatGPT 3.5 Turbo or ChatGPT 4/4o as the benchmark and find a way to run the test set in your course with ChatGPT to determine the sentiment. What is your conclusion?

Using llama 3.1 base got 0.909 accuracy (~ 12'), after fine tuning ~(2hours) got 0.998 (~20') with 1 epoch training. The result may be better over time but the running time is really long.

```
[10] llama_pred = pd.read_csv('y_pred.csv').y_pred.tolist()
     for x, y_true, y_pred in zip(sa.tweets_test, sa.y_test, llama_pred):
         if y_true != y_pred:
             print('THE TWEET IS:\t\t', x)
             print('True: %d\tPred: %d' % (y_true, y_pred))
             print('_____'*3)
```

```
THE TWEET IS:           Crazy girlfriends be like :-)(-: Jesus Christ. http://t.co/RTWjc7e1lM
True: 1 Pred: 0
_____
THE TWEET IS:           me: as long as i feel comfortable im gonna wear what i want
my mother: haha...that sounds nice...but no :-)
True: 1 Pred: 0
_____
THE TWEET IS:           @0PlATE bc I am feeling very creepy today :  )  // pmsl. mianhe, milkeu :(
True: 0 Pred: 1
_____
THE TWEET IS:           Sr. Financial Analyst - Expedia, Inc.: (#Bellevue, WA) http://t.co/ktknMhvwCI #Finance #ExpediaJobs #Job #Jobs #Hiring
True: 0 Pred: 1
_____
```

Task 9

The code presented in a jupyter notebook is for illustrative purpose only. I showed you a better coding version in .py file however it is not good enough to deliver in industrial level. Re-write the solution in OOP (Object-Oriented Programming) kind of way by wrapping everything up in a class named SentimentAnalysis.

Attributes (all the attributes are non-public):

- _train_x
- _train_y
- _freqs
- _X
- _J
- _W

Task 9

METHODS	EXPLANATION
process_tweets	process a string into a list of cleaned words
build_freqs	input _train_x and _train_y to get a dictionary mapping each (word, sentiment) pair to its frequency
extract_features	input a list of cleaned words of a tweet and _freqs, output: x: a feature vector of dimension (1,3)
sigmoid	input the logit z and compute sigmoid of z
gradientDescent	
model_training	calculate values of attribute _freqs, _X, _J, _W
predict_tweet	return the probability of a tweet (str) being positive or negative
test_logistic_regression	compute model accuracy and error analysis
try_a_tweet	users manually input a tweet & model would return a sentiment evaluation

Task 10

(**optional**) Find a set of neutral sentiment tweets or you can find a completely new dataset of your choice with 3 sentiment classes (positive, negative and neutral). Solve the problem this time with Multinomial Logistic Regression (soft-max function instead of sigmoid, see pages 86-89 of the bible:D). You can make use of any Python lib as you want or just build things from scratch as illustrated in the course.

Task 10

label in binary
Logistic Regression

0 → negative
1 → positive
0 → negative
1 → positive

label in multinomial logistic regression
(softmax regression)

[1 0 0 0 0] → class 0
[0 1 0 0 0] → class 1
[0 0 1 0 0] → class 2
[0 0 0 1 0] → class 3
[0 0 0 0 1] → class 4

one hot vector

How about prediction probability?

0.7858

P(y = 1 | x)

> threshold: class 1

[0.40 0.30 0.30] sum = 1

P(y = 0 | x) largest => class 0

P(y = 1 | x)

P(y = 2 | x)

Task 10

In binary Logistic Regression:

$$z = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

$$y_{\text{pred}} = \sigma(z) = \frac{1}{1 + \exp^{-z}}$$ sigmoid function

In Softmax Regression:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right) \quad \text{z}$$

<=>

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)} \quad 1 \le i \le K$$ <=> $$\text{softmax}(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{K} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{K} \exp(z_i)}, \ldots, \frac{\exp(z_K)}{\sum_{i=1}^{K} \exp(z_i)} \right]$$

Task 10

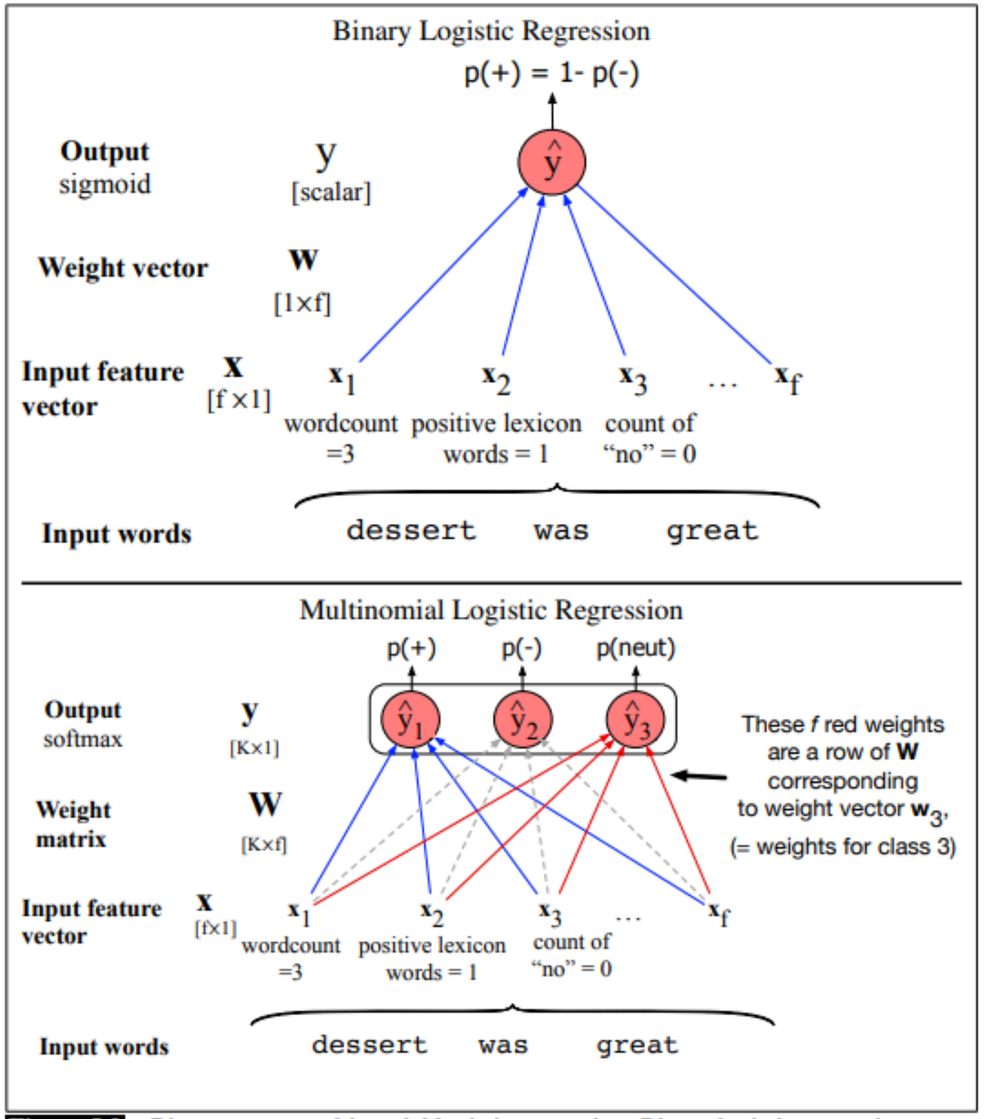


Figure 5.3 Binary versus multinomial logistic regression. Binary logistic regression uses a single weight vector $\mathbf{w}$, and has a scalar output $\hat{y}$. In multinomial logistic regression we have K separate weight vectors corresponding to the K classes, all packed into a single weight matrix $\mathbf{W}$, and a vector output $\hat{\mathbf{y}}$. We omit the biases from both figures for clarity.

Task 10

```python
for i in range(num_iters):

    # Compute the linear combination of inputs and weights (logits)
    z = np.dot(x, W) + b

    # Compute the softmax probabilities using a numerically stable softmax function
    h = softmax(z)

    # Avoid taking log of zero by adding a small constant (epsilon) to h
    epsilon = 1e-10
    h = np.clip(h, epsilon, 1 - epsilon)

    # Calculate the cost (cross-entropy loss)
    J = np.mean(-np.sum(y * np.log(h), axis=1))
    losses.append(float(J))

    # Compute the gradient of the loss with respect to W and b
    dW = np.dot(x.T, (h - y)) / m
    db = np.mean(h - y, axis=0)

    # Update weights and bias using gradient descent
    W -= alpha * dW
    b -= alpha * db
```

Loss function:

$$J = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ji} \log\left(\frac{\exp(\mathbf{w}_j^T \mathbf{x}_i)}{\sum_{k=1}^{C} \exp(\mathbf{w}_k^T \mathbf{x}_i)}\right)$$

y_pred

Update weight and bias:

$$W = W - \frac{\alpha}{m} \times \left(X^T \cdot \left(y_{\text{pred}} - y\right)\right)$$

$$b = b - \frac{\alpha}{m} \times \left(y_{\text{pred}} - y\right)$$

Task 10

Some technical notation:

"Stability improvement" softmax function

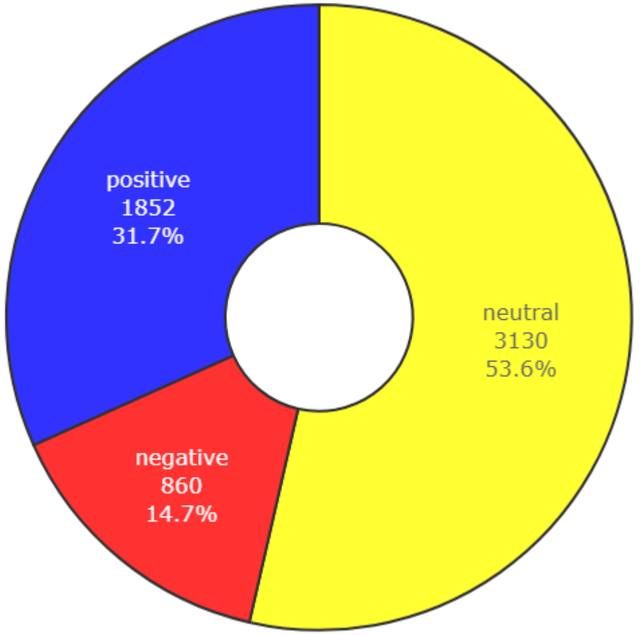
$$\frac{\exp(z_i)}{\sum_{j=1}^{C}\exp(z_j)} = \frac{\exp(-c)\exp(z_i)}{\exp(-c)\sum_{j=1}^{C}\exp(z_j)} = \frac{\exp(z_i - c)}{\sum_{j=1}^{C}\exp(z_j - c)}$$

When one of the z_i too large, the calculation of exp(z_i) can cause overflow, affecting the results of the softmax function.

```
# Avoid taking log of zero by adding a small constant (epsilon) to h
epsilon = 1e-10
h = np.clip(h, epsilon, 1 - epsilon)
```

Same as Task 2!

Task 10



Sentiment distribution of "Financial Sentiment Analysis" dataset

```
_____________________
great
[[0.36383854 0.32059595 0.31556551]]
[0]

_____________________
great great
[[0.39546069 0.3070863  0.297453  ]]
[0]

_____________________
great great great
[[0.42796645 0.29286983 0.27916371]]
[0]

_____________________
great great great great
[[0.46109014 0.27807282 0.26083705]]
[0]

Softmax regression model's accuracy = 0.6533
```