1 Overview

The goal of this assignment is to build a classification machine learning (ML) pipeline in a web application to use as a tool to analyze the models to gain useful insights about model performance. Using trained classification models, build a ML application that predicts whether a product review is positive or negative.

The learning outcomes for this assignment are:

- Preprocessing categorical data using n-gram, Term-Frequency Inverse document, and bag-of-word feature representations
- Build end-to-end classification pipeline with four classifiers 1) Logistic Regression and 2) Stochastic Gradient Descent.
- Evaluate classification methods using standard metrics including precision, recall, and accuracy, ROC Curves, and area under the curve.
- Develop a web application that walks users through steps of the classification pipeline and provide tools to analyze multiple methods across multiple metrics.

Assignment Outline

- Setup
- End-to-End Classification Models
- Testing Code with Github Autograder
- Reflection Assessment

2 Setup

Reading Prerequisites

Review the jupyter notebook in Chapter 9 Unsupervised Learning of "Machine Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow." O'Reilly Media, Inc., 2022. Available on Canvas under 'Library Reserves'.

3 End-to-End Classification Models for Product Review Sentiment

The goal of this assignment is to train multiple classification algorithms using the Amazon Products dataset, evaluate model performance, and deploy the model in a sentiment predict application. You can assume that all User Interface (UI) elements are done for you. Your goal is to fill in the checkpoint functions that demonstrate your understanding of classification models.

We have provided helper functions defined in the help_functions.py file. See Appendix at the end of this document on helper functions.

3.1 Amazon Product Reviews Dataset

This assignment involves training and evaluating ML end-to-end pipeline in a web application using the Amazon Product Reviews dataset. Millions of Amazon customers have contributed over a hundred million reviews to express opinions and describe their experiences regarding products on the Amazon.com website. This makes Amazon Customer Reviews a rich source of information for academic researchers in the fields of natural language processing (NLP), information retrieval (IR), and machine learning (ML), amongst others. Specifically, this dataset was constructed to represent a sample of customer evaluations and opinions, variation in the perception of a product across geographical regions, and promotional intent or bias in reviews.

We have added additional features to the dataset. There are many features, but the important ones include:

- name: name of Amazon product
- Reviews.text: text in review
- Reviews.title: title of reviews

The dataset is available in the Github repository (see Github Classroom link).

3.2 Explore and Preprocess Data (3 points)

The goal of this page is to explore and preprocess the dataset. First, import the dataset from your machine. We have provided code to remove unuseful features using the clean_data() helper function (see helper_function.py). Then, remove punctuation from the reviews. We have provided UI and functions to summarize text statistics from reviews, search reviews with a keyword, and remove reviews. At the end of this page, encode documents with word counts and Term Frequency Inverse Document frequency features. See details about the checkpoint functions below.

Some activities require a try and except block to train classification models (in later sections).

```
try:
        # write some code
Except ValueError as err:
        st.write({str(err)}) # Print the error message
```

Checkpoint 1

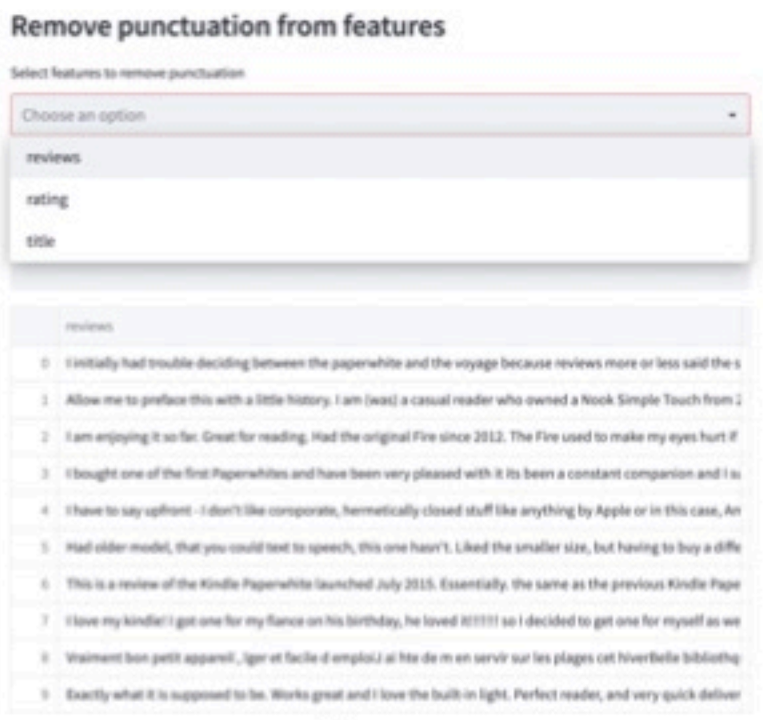
remove_punctuation(df,
            features)

Inputs:
- df (pandas dataframe): dataset
- Features: features to remove punctuation

Outputs: df (updated)

The function returns a dataframe with updated features with removed punctuation.

Perform the following tasks in the remove_punctuation function:


Remove punctuation from features
Select features to remove punctuation
Choose an option
reviews
rating
title

	reviews
0	I initially had trouble deciding between the paperwhite and the voyage because reviews more or less said the s
1	Allow me to preface this with a little history. I am (was) a casual reader who owned a Nook Simple Touch from 2
2	I am enjoying it so far. Great for reading. Had the original Fire since 2012. The Fire used to make my eyes hurt if
3	I bought one of the first Paperwhites and have been very pleased with it its been a constant companion and I su
4	I have to say upfront - I don't like corporate, hermetically closed stuff like anything by Apple or in this case, An
5	Had older model, that you could text to speech, this one hasn't. Liked the smaller size, but having to buy a diffe
6	This is a review of the Kindle Paperwhite launched July 2015. Essentially, the same as the previous Kindle Pape
7	I love my kindle! I got one for my fiance on his birthday, he loved it!!!!!! so I decided to get one for myself as we
8	Vraiment bon petit appareil , lger et facile d emploi.J ai hte de m en servir sur les plages cet hiverBelle bibliothq
9	Exactly what it is supposed to be. Works great and I love the built in light. Perfect reader, and very quick deliver

Figure:

1. Create a translator using the string library that creates a one to one mapping of a character to its translation/replacement.
2. Write a for loop that iterates through the feature names, check that strings are in the feature.
    a. If the features are strings, use the translator to remove punctuation from the strings. It's recommended that you use a lambda function.
3. Store the updated dataframe df in st.session_state['data'].

Example code:

```
translator = str.maketrans('', '', string.punctuation)
for feature_name in features:
if(df[feature_name].dtype ==’object’):
        df[feature_name] = … # add code here
```

Checkpoint 2

This function performs word count encoding on features in the dataframe.

word_count_encoder(df,

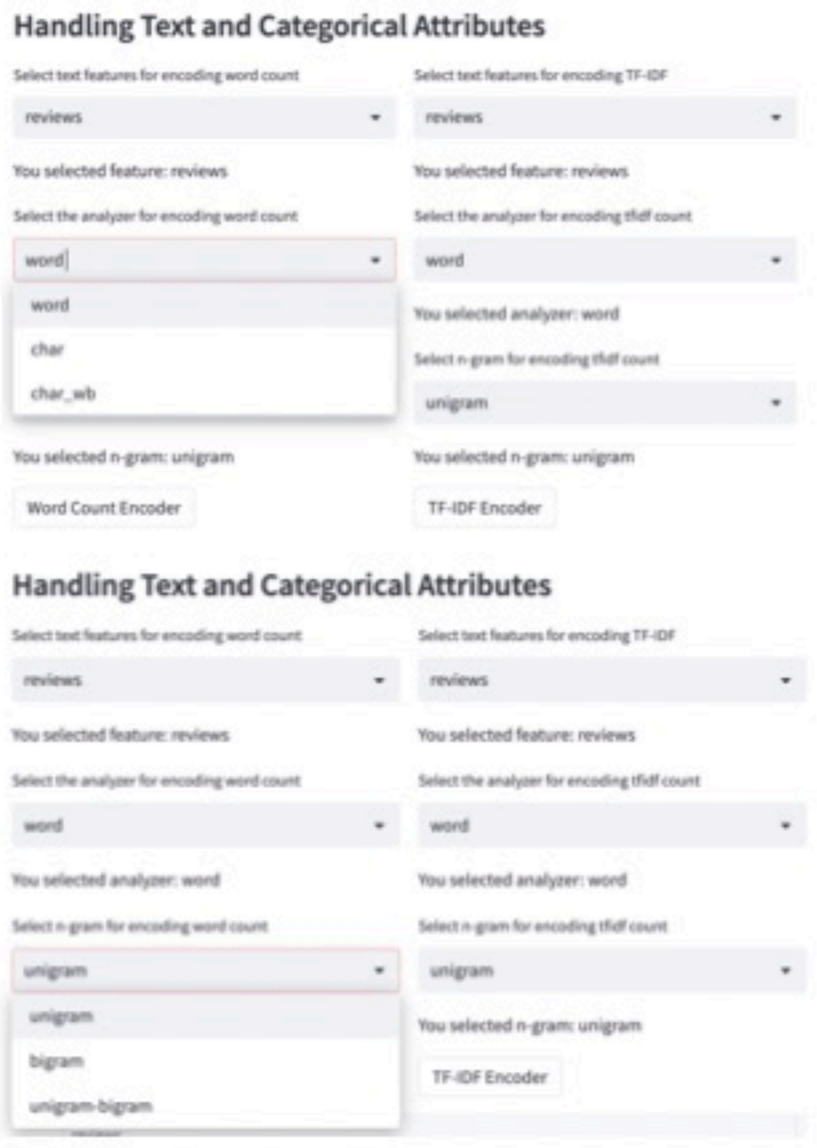
```
        feature,
        analyzer,
        ngram_range,
        stop_words)
```

Input:

- df: the pandas dataframe
- feature: the feature(s) to perform word count encoding
- analyzer: 'word', 'char', 'char_wb'
- ngram_range (tuple): unigram - (1,1), unigram & bigram - (1,2), bigram - (2,2)
- stop_words: stop_words (list) or 'english' (string)

Output:

- df: dataframe with word count feature
- count_vect: CountVectorizer
- word_count_df: pandas dataframe with prefix 'word_count_'


Handling Text and Categorical Attributes

Select text features for encoding word count: reviews
You selected feature: reviews
Select the analyzer for encoding word count: word
Select text features for encoding TF-IDF: reviews
You selected feature: reviews
Select the analyzer for encoding tfidf count: word
You selected analyzer: word
Select n-gram for encoding tfidf count: unigram
You selected n-gram: unigram
You selected n-gram: unigram

Handling Text and Categorical Attributes

Select text features for encoding word count: reviews
You selected feature: reviews
Select the analyzer for encoding word count: word
You selected analyzer: word
Select n-gram for encoding word count: unigram
Select text features for encoding TF-IDF: reviews
You selected feature: reviews
Select the analyzer for encoding tfidf count: word
You selected analyzer: word
Select n-gram for encoding tfidf count: unigram
You selected n-gram: unigram

Figure:

Words need to be encoded as integers or floating point values to input to a machine learning algorithm. The function performs word count encoding on the given features and returns the data frame with word count encoded features.

Perform the following tasks in the word_count_encoder function:

1. Use the CountVectorizer() from sklearn.feature_extraction.text to create a count vectorizer class object.
2. Use the count vectorizer transform() function to transform the features in df to create frequency counts for words.
3. Convert the frequency counts to an array using the toarray() function and convert the array to a pandas dataframe.
4. Add a prefix to the column names in the data frame created in Step 3 using add_prefix() pandas function with 'word_count_' as the prefix.
5. Add the word count dataframe to df using the pd.concat() function.
6. Update the confirmation statement to show the length of the word_count dataframe.

Checkpoint 3

This function performs tf-idf encoding on the given features.

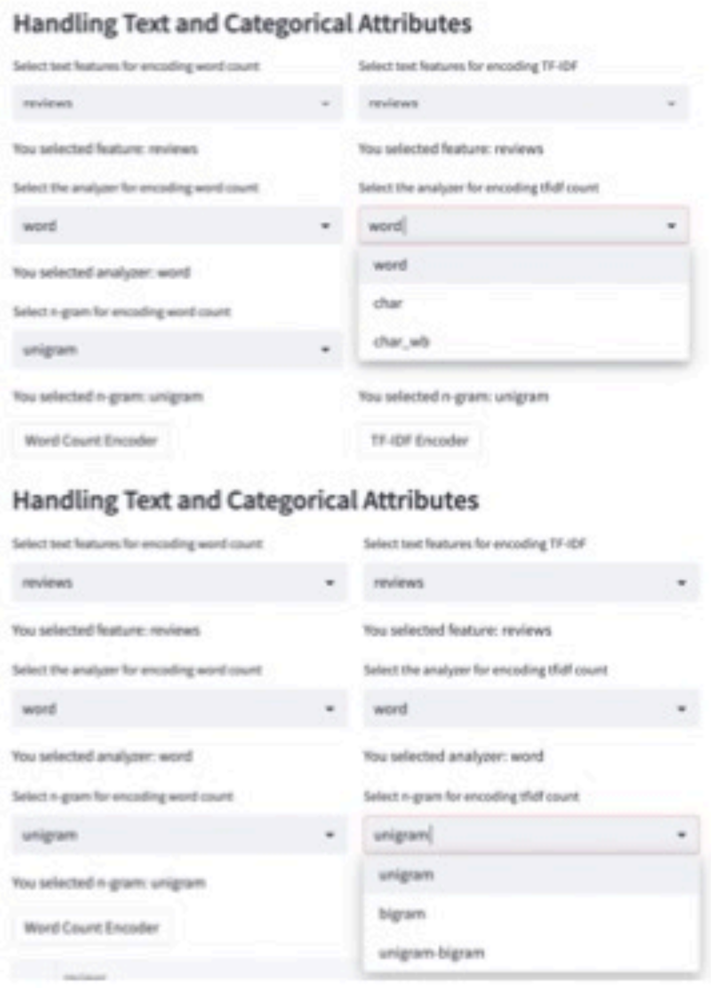
```
tf_idf_encoder(df,
              feature,
              word_encoder)
```

Input:

- df: the pandas dataframe
- feature: the feature(s) to perform tf-idf encoding
- analyzer: 'word', 'char', 'char_wb'
- ngram_range (tuple): unigram - (1,1), unigram & bigram - (1,2), bigram - (2,2)
- stop_words: stop_words (list) or 'english' (string)
- norm: 'l2' - Sum of squares, 'l1'- Sum of absolute values, 'n' for no normalization

Output:

- df: dataframe with tf-idf encoded feature
- count_vect: CountVectorizer
- tfidf_transformer: TfidfTransformer with normalization in norm
- tfidf_df: pandas dataframe with prefix 'tf_idf_word_count_'



Handling Text and Categorical Attributes

Select text features for encoding word count: reviews
You selected feature: reviews
Select the analyzer for encoding word count: word
You selected analyzer: word
Select n-gram for encoding word count: unigram
You selected n-gram: unigram

Select text features for encoding TF-IDF: reviews
You selected feature: reviews
Select the analyzer for encoding tfidf count: word
You selected n-gram: unigram

Handling Text and Categorical Attributes

Select text features for encoding word count: reviews
You selected feature: reviews
Select the analyzer for encoding word count: word
You selected analyzer: word
Select n-gram for encoding word count: unigram
You selected n-gram: unigram

Select text features for encoding TF-IDF: reviews
You selected feature: reviews
Select the analyzer for encoding tfidf count: word
You selected analyzer: word
Select n-gram for encoding tfidf count: unigram

Figure:

Perform TF-IDF encoding on input features, which quantifies the importance or relevance of words or phrases. This function returns the dataframe with TF-IDF encoded features. Perform the following tasks in the tf_idf_word_count_encoder function:

1. Use the CountVectorizer() from sklearn.feature_extraction.text to create a count vectorizer class object.
2. Use the count vectorizer transform() function to the feature in df to create frequency counts for words.
3. Use the TfidfTransformer() from sklearn.feature_extraction.text to create a TF-IDF transformer class object.
4. Transform the frequency counts (from Step 2) into TF-IDF features using the TfidfTransformer object.
5. Create a pandas dataframe for the TF-IDF features which takes the TF-IDF features array as input so convert the TF-IDF features to an array using the toarray() function.
6. Add a prefix to the column names in the data frame created in Step 3 using add_prefix() pandas function with 'tf_idf_word_count_' as the prefix.
7. Add the TF-IDF dataframe to df using the pd.concat() function.

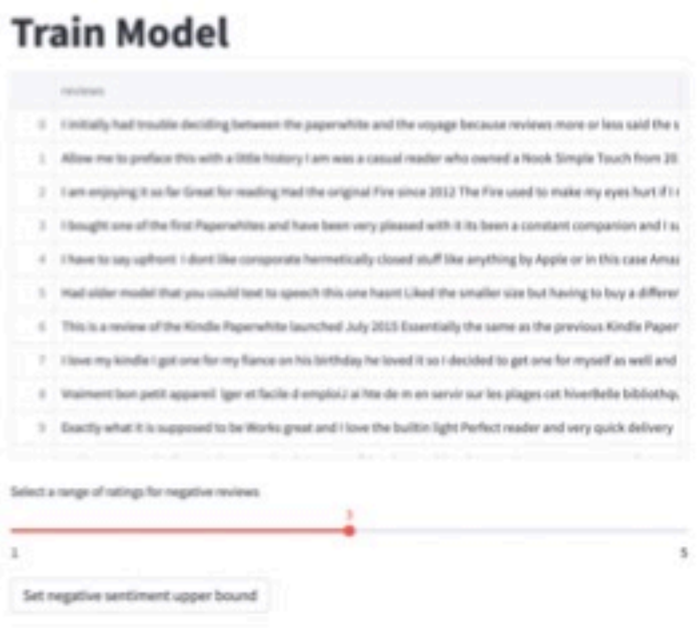
3.3 Train Regression Models (8 points)

The goal of this page is to train multiple models and inspect model coefficients and cross validation results for relevant models. First, we have provided code to assign the negative values to the product ratings using the negative ratings selected from the user (assume rating=3 is neural) in the set_pos_neg_reviews() function. Then, write the split_dataset() function which splits the dataset into training and validation input and output using the appropriate word encoding (as specified by the user). Next, write four functions to training functions to train the following models: 1) Logistic Regression and 2) Stochastic Gradient Descent. Lastly, write a function to inspect the coefficients of each model.

Checkpoint 4

The function splits the dataset into four parts, in the following order: X_train, X_val, y_train, y_val.

```
split_dataset(df,
              number,
              target,
              feature_encoding,
              random_state=42)
```



Train Model

	reviews
0	I initially had trouble deciding between the paperwhite and the voyage because reviews more or less said the s
1	Allow me to preface this with a little history I am was a casual reader who owned a Nook Simple Touch from 20
2	I am enjoying it so far Great for reading Had the original Fire since 2012 The Fire used to make my eyes hurt if I
3	I bought one of the first Paperwhites and have been very pleased with it its been a constant companion and I s
4	I have to say upfront I dont like corporate hermetically closed stuff like anything by Apple or in this case Ama
5	Had older model that you could text to speech this one hasnt Liked the smaller size but having to buy a differe
6	This is a review of the Kindle Paperwhite launched July 2015 Essentially the same as the previous Kindle Paper
7	I love my kindle I got one for my fiance on his birthday he loved it so I decided to get one for myself as well and
8	Vraiment bon petit appareil lger et facile d emploiJ ai hte de m en servir sur les plages cet hiverBelle bibliothq
9	Exactly what it is supposed to be Works great and I love the builtin light Perfect reader and very quick delivery

Select a range of ratings for negative reviews

Figure:

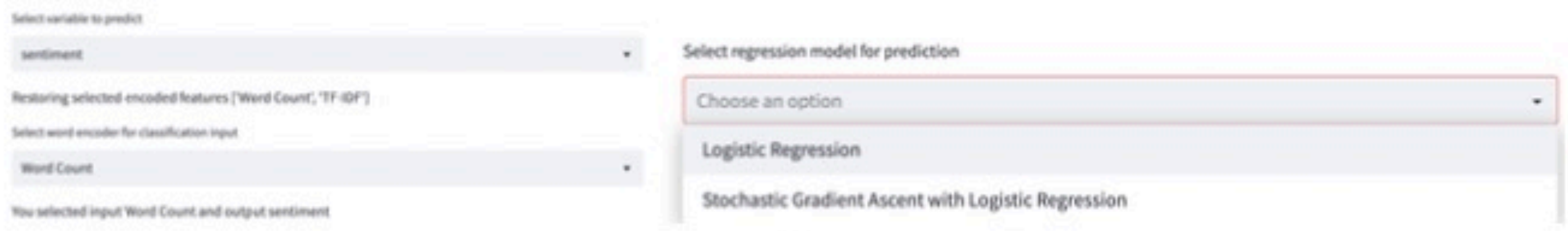
Input:

- X: training features
- y: training targets - number: the ratio of test samples - target: article feature name 'rating'
- feature_encoding: (string) 'Word Count' or 'TF-IDF' encoding
- random_state: determines random number generation for centroid initialization Output:
- X_train_sentiment: training features (word encoded)
- X_val_sentiment: test/validation features (word encoded)
- y_train: training targets
- y_val: test/validation targets

Perform the following tasks in the split_dataset function:

1. Use the train_test_split() function to split the dataset into four parts including X_train, X_val, y_train, y_val sets using the input X, y, number/100 (set test percentage), and random state.
2. Check the feature_encoding list of strings that contain either 'TF-IDF' or 'Word Count' ing the feature_encoding list and set the input to feature names that start with 'tf_idf_word_count_' for TF-IDF and 'word_count_' for word count (see example below). Also, the dataset can contain both feature encodings.

```
if('Word Count' in feature_encoding):
          X_train_sentiment = X_train.loc[:, X_train.columns.str.startswith('word_count_')]
          X_val_sentiment = X_val.loc[:, X_val.columns.str.startswith('word_count_')]
```


Select variable to predict

sentiment

Restoring selected encoded features ['Word Count', 'TF-IDF']

Select word encoder for classification input

Word Count

You selected input Word Count and output sentiment

Select regression model for prediction

Choose an option

Figure:

The LogisticRegression Class is an object with parameters and functions to train and evaluate logistic regression algorithms. Your task is to write the Logistic class functions and return the correct outputs to complete your web application to predict product sentiment.

The LogisticRegression class contains the following variables:

- learning_rate η: controls the pace at which the gradient updates weights and ranges from 0 to 1. Setting η too high causes the gradient to overestimate the cost and cause weight to approach inf or Nan. Setting η too low causes the gradient to take longer to converge to minimum cost.
- num_iterations: controls the number of iterations to update the weights for gradient ascent.
- likelihood_history: cost of the residual sum of squares after taking the gradient in Gradient Ascent.

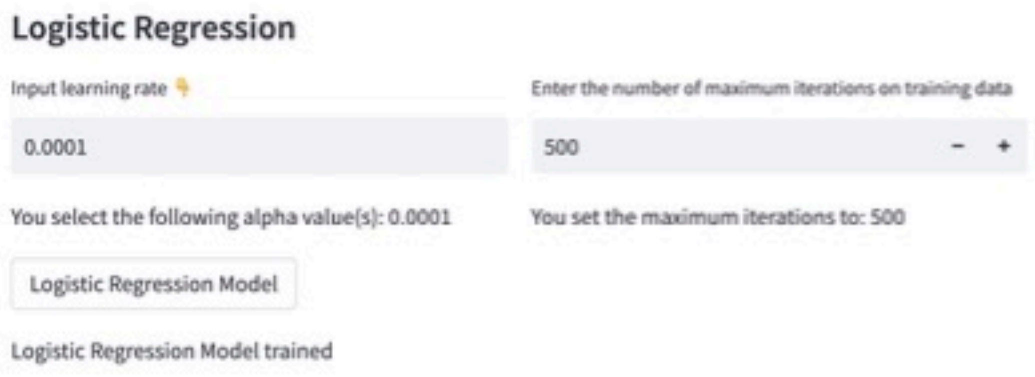
Checkpoint 5

Produces probabilistic estimate for P(y_i = +1 | x_i, w). Estimate ranges between 0 and 1.

predict_probability(self, X)

Input
- X: Input features

Output
- y_pred: probability of positive product review


Logistic Regression

Input learning rate

0.0001

Enter the number of maximum iterations on training data

500

You select the following alpha value(s): 0.0001

You set the maximum iterations to: 500

Logistic Regression Model trained

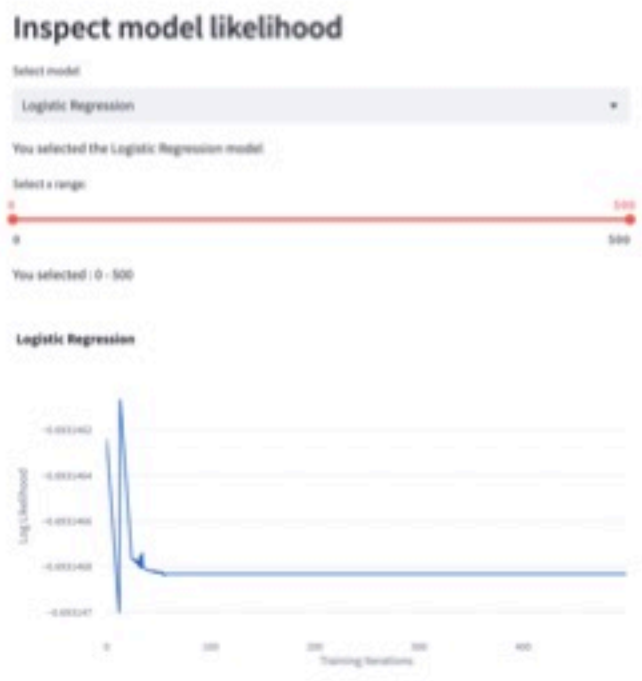
Figure:

Checkpoint 6

Compute the average log-likelihood of logistic regression coefficients.

```
compute_avg_log_likelihood(self,
                           X,
                           Y,
                           W)
```

Input
- X: subset of features in dataset
- Y: true sentiment of inputs
- W: logistic regression weights


Inspect model likelihood

Select model

Logistic Regression

You selected the Logistic Regression model

Select a range

You selected : 0 - 500

Output

- lp: log likelihood estimation

Figure:

Checkpoint 7

Compute the logistic regression derivative using gradient ascent and update weights self.W.

update_weights(self)

Inputs: None
Output: None

Checkpoint 8

Compute hypothetical function h(x) = X*W.

predict(self, X)

Input:

- X: Input features
- W: weights/coefficients of logistic regression model
- b: bias or y-intercept of logistic regression classifier

Output:

- Y: list of predicted classes

Checkpoint 9

Run gradient ascent to fit features to data using logistic regression.

fit(self, X, Y)

Input

- X: Input features
- Y: list of actual product sentiment classes

Output: None

StochasticLogisticRegression Class is an object with parameters and functions to train and evaluate logistic regression algorithms. This class inherits the learning_rate and num_iterations variables and all functions except fit from the LogisticRegression class. Your task is to write the StochasticLogisticRegression class functions and return the correct outputs to complete your web application for predicting product sentiment.

StochasticLogisticRegression defines a new fit() function.

Checkpoint 10

This function prints the coefficients of the trained models.

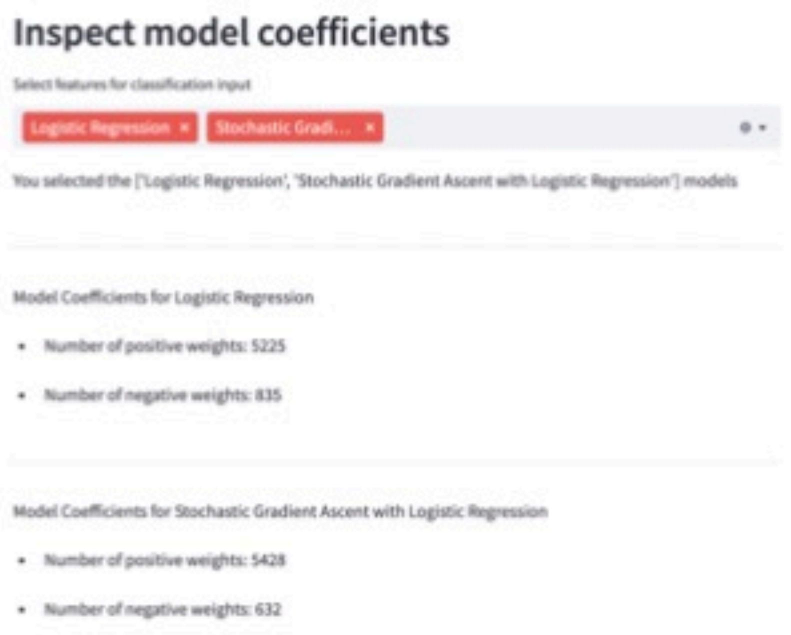
get_weights(self, model_name)

Input:

- model_name (list of strings): list of model names including: 'Logistic Regression', 'Stochastic Gradient Ascent with Logistic Regression'

Output:

- - out_dict: a dictionary contains the coefficients of the selected models, with the following keys: - 'Logistic Regression'
- - 'Stochastic Gradient Ascent with Logistic Regression'



Inspect model coefficients

Select features for classification input

You selected the ['Logistic Regression', 'Stochastic Gradient Ascent with Logistic Regression'] models

Model Coefficients for Logistic Regression

- Number of positive weights: 5225
- Number of negative weights: 835

Model Coefficients for Stochastic Gradient Ascent with Logistic Regression

- Number of positive weights: 5428
- Number of negative weights: 632

Figure:

Checkpoint 11

Run mini-batch stochastic gradient ascent to fit features to data using logistic regression

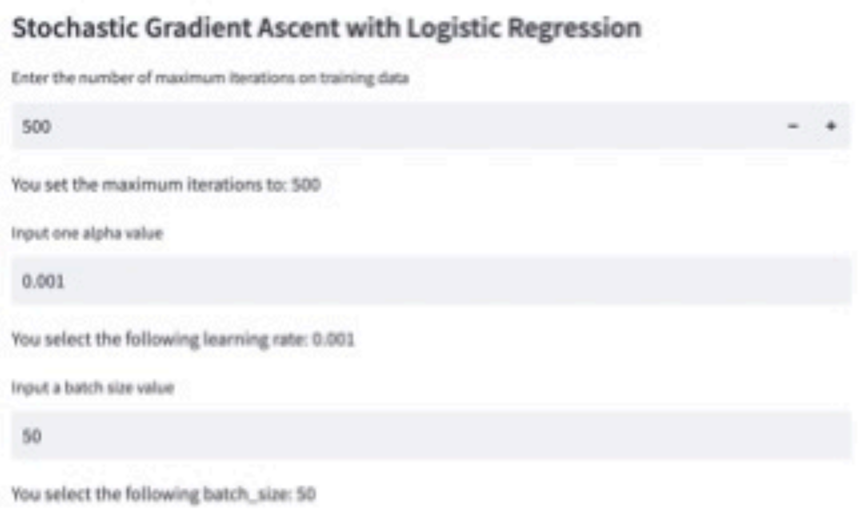
fit(self, X, Y)

Input

- X: input features
- Y: target variable (product sentiment)

Output: None

Use a learning rate schedule that updates the learning rate as follows: lr=lr/1.02 in each training iteration.



Stochastic Gradient Ascent with Logistic Regression

Enter the number of maximum iterations on training data

500

You set the maximum iterations to: 500

Input one alpha value

0.001

You select the following learning rate: 0.001

Input a batch size value

50

You select the following batch_size: 50

Figure:

3.4 Test Regression Models (2 points)

The goal of this page is to evaluate the classification models using precision, recall, accuracy, and ROC Curves. First, the user selects the performance metrics for evaluation. Then, they select the classification models to evaluate. Using the aforementioned inputs, two functions, 1) that computes the evaluation metrics using a trained model and 2) displays a ROC Curve using precision and recall. At the end of this page, the user can select a model to deploy on the 'Deploy App' page.

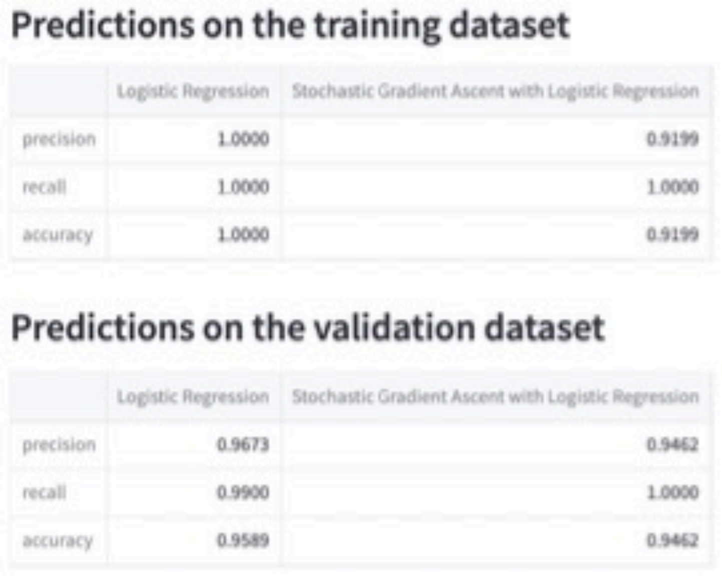
Checkpoint 12

Compute and return the classification accuracy.

compute_accuracy(prediction_labels,
                              true_labels)

Input
- prediction_labels (numpy): predicted product sentiment
- true_labels (numpy): true product sentiment

Output
- accuracy (float): accuracy percentage (0-100%)



Predictions on the training dataset

	Logistic Regression	Stochastic Gradient Ascent with Logistic Regression
precision	1.0000	0.9199
recall	1.0000	1.0000
accuracy	1.0000	0.9199

Predictions on the validation dataset

	Logistic Regression	Stochastic Gradient Ascent with Logistic Regression
precision	0.9673	0.9462
recall	0.9900	1.0000
accuracy	0.9589	0.9462

Figure:

Checkpoint 13

Compute precision and recall of model on X_test data using word_encoder

compute_precison_recall(prediction_labels, true_labels)
                                   true_labels)

Input
- sentiment_predictions (numpy): validation dataset
- y_test (numpy): validation labels/target

Output
- precision (float): precision score = TP/TP+FP
- recall (float): recall score = TP/TP+FN