coursera Q 搜索目录 无限制访问 2000 多门课程。随时取消。 马上开始 ① < 返回到第2周 Ⅲ 课程 Implementing logistic regression from scratch The goal of this assignment is to implement your own logistic regression classifier. You will: Extract features from Amazon product reviews. Convert an SFrame into a NumPy array. Implement the link function for logistic regression. Write a function to compute the derivative of the log likelihood function with respect to a single coefficient. Implement gradient ascent. Given a set of coefficients, predict sentiments. Compute classification accuracy for the logistic regression model. Let's get started! If you are doing the assignment with IPython Notebook

An IPython Notebook has been provided below to you for this assignment. This notebook contains the instructions, quiz questions and partially-completed code for you to use as well as some cells to test your code. Make sure that you are using the latest version of GraphLab Create. What you need to download If you are using GraphLab Create: Download the Amazon product review dataset (subset) in SFrame format. Notice the subset suffix: amazon_baby_subset.gl.zip Download the companion IPython notebook: module-3-linear-classifier-learning-assign... Download the list of 193 significant words: important_words.json.zip · If you are using Amazon EC2, download the binary files for NumPy arrays from the link below. See the companion notebook for the instructions. module-3-assignment-numpy-arrays.npz... Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data If you are not using GraphLab Create: If you are using SFrame, download the Amazon product review dataset (subset) in SFrame format. Notice the subset suffix: amazon_baby_subset.gl.zip If you are using a different package, download the Amazon product review dataset (subset) in CSV format: amazon_baby_subset.csv.zip Download the list of 193 significant words: important_words.json.zip If you are using GraphLab Create and the companion IPython Notebook Open the companion IPython notebook and follow the instructions in the notebook. If you are using other tools This section is designed for people using tools other than GraphLab Create. You will not need any machine learning packages since we will be implementing logistic regression from scratch. We highly suggest you use SFrame since it is open source. In this part of the assignment, we describe general instructions, however we will tailor the instructions for SFrame. If you choose to use SFrame, you should be able to follow the instructions in the next section and complete the assessment. All code samples given here will be applicable to SFrame. You are free to experiment with any tool of your choice, but some many not produce correct numbers for the quiz questions. If you are using SFrame Make sure to download the companion IPython notebook: module-3-linear-classifier-learning-assign... You will be able to follow along exactly if you replace the first two lines of code with these two lines: 1 import sframe 2 products = sframe.SFrame('amazon_baby_subset.gl/') After running this, you can follow the rest of the IPython notebook and disregard the rest of this reading. Note: To install SFrame (without installing GraphLab Create), run 1 pip install sframe If you are NOT using SFrame Load review dataset 1. For this assignment, we will use a subset of the Amazon product review dataset. The subset was chosen to contain similar numbers of positive and negative reviews, as the original dataset consisted primarily of positive reviews. Load the dataset into a data frame named **products**. One column of this dataset is **sentiment**, corresponding to the class label with +1 indicating a review with positive sentiment and -1 for negative sentiment. 2. Let us quickly explore more of this dataset. The name column indicates the name of the product. Try listing the name of the first 10 products in the dataset. After that, try counting the number of positive and negative reviews. Note: For this assignment, we eliminated class imbalance by choosing a subset of the data with a similar number of positive and negative reviews. Apply text cleaning on the review data 3. In this section, we will perform some simple feature cleaning using data frames. The last assignment used all words in building bag-of-words features, but here we limit ourselves to 193 words (for simplicity). We compiled a list of 193 most frequent words into the JSON file named important_words.json. Load the words into a list important_words. Let us perform 2 simple data transformations: Remove punctuation Compute word counts (only for important_words)

> After #4 and #5, the data frame products should contain one column for each of the 193 important_words. As an example, the column perfect contains a count of the number of times the word perfect occurs in each of the reviews. Now, write some code to compute the number of product reviews that contain the word perfect. Hint: First create a column called contains_perfect which is set to 1 if the count of the word perfect (stored in column perfect is >= 1. Sum the number of 1s in the column contains_perfect. Quiz Question. How many reviews contain the word perfect? Convert data frame to multi-dimensional array 8. It is now time to convert our data frame to a multi-dimensional array. Look for a package that provides a highly optimized matrix operations. In the case of Python, NumPy is a good choice. Write a function that extracts columns from a data frame and converts them into a multi-dimensional array. We plan to use them throughout the course, so make sure to get this function right.

features: a list of string, containing the names of the columns that are used as features.

label: a string, containing the name of the single column that is used as class labels.

 If your tool supports it, fill n/a values in the review column with empty strings. The n/a values indicate empty reviews. For instance, Pandas's the fillna() method lets you replace all N/A's in the review columns as follows:

Write a function remove_punctuation that takes a line of text and removes all punctuation from that text. The

 Apply the remove_punctuation function on every element of the review column and assign the result to the new column review_clean. Note. Many data frame packages support apply operation for this type of task.

Now we proceed with the second item. For each word in important_words, we compute a count for the number

of times the word occurs in the review. We will store this count in a separate column (one for each word). The result of this feature processing is a single column for each word in important_words which keeps a count of the

Note: There are several ways of doing this. One way is to create an anonymous function that counts the

products[word] = products['review_clean'].apply(lambda s : s.split().count

occurrence of a particular word and apply it to every element in the review_clean column. Repeat this step for

1 products = products.fillna({'review':''}) # fill in N/A's in the review column

function should be analogous to the following Python code:

number of times the respective word occurs in the review text.

every word in important_words. Your code should be analogous to the following:

return text.translate(None, string.punctuation)

We start with the first item as follows:

1 → def remove_punctuation(text): import string

Consult appropriate manuals.

1 - for word in important_words:

The function should accept three parameters:

· dataframe: a data frame to be converted

The function should return two values:

Convert the column into a 1D array.

Return the 2D array and the 1D array.

dataframe['constant'] = 1

6 label_sarray = dataframe[label]

3

Users of SFrame or Pandas would execute these steps as follows:

1 ▼ def get_numpy_data(dataframe, features, label):

features_frame = dataframe[features] feature_matrix = features_frame.as_matrix()

label_array = label_sarray.as_matrix() return(feature matrix, label array)

sentiment would contain the content of the column sentiment.

named predict_probability that implements the link function.

First compute the dot product of feature_matrix and coefficients.

Your code should be analogous to the following Python function:

2 produces probablistic estimate for P(y_i = +1 | x_i, w).

5 - def predict_probability(feature_matrix, coefficients):

Take dot product of feature_matrix and coefficients

Compute P(y_i = +1 | x_i, w) using the link function

Take two parameters: feature_matrix and coefficients.

Then compute the link function P(y = +1|x, w).

Return the predictions given by the link function.

3 estimate ranges between 0 and 1.

YOUR CODE HERE

YOUR CODE HERE

predictions = ...

return predictions

feature_matrix and the coefficient vector w:

 $\frac{\partial \ell}{\partial w_j} = \sum_{i=1}^N h_j(\mathbf{x}_i) (\mathbf{1}[y_i = +1] - P(y_i = +1|\mathbf{x}_i, \mathbf{w}))$

Aside. How the link function works with matrix algebra

return predictions

score = ...

8

9 10

11 12

13 14

15

feature vector $h(\mathbf{x}_i)$:

 $[feature_matrix] =$

11. Recall from lecture:

The function should do the following:

derivative = ...

return derivative

about the derivation of this equation):

indicator = (sentiment==+1)

assess the algorithm.

following Python function:

return lp

feature_matrix: 2D array of features

The function carries out the following steps:

Repeat steps 2-6 for max_iter times.

step_size, max_iter):

for itr in xrange(max_iter):

indicator = (sentiment==+1)

errors = indicator - predictions

with coefficients[j]

called derivative # YOUR CODE HERE derivative = ...

YOUR CODE HERE

feature_matrix = feature_matrix extracted in #9

Save the returned coefficients to variable coefficients.

sentiment = sentiment extracted in #9

step_size = 1e-7

max_iter = 301

Predicting sentiments

 $\hat{y}_i = \begin{cases} +1 & \text{if } \mathbf{x}_i^{\mathsf{T}} \mathbf{w} > 0 \\ -1 & \text{if } \mathbf{x}_i^{\mathsf{T}} \mathbf{w} \leq 0 \end{cases}$

Measuring accuracy

accuracy can be computed as follows:

following formula:

YOUR CODE HERE predictions = ...

1 from math import sqrt

3

4 +

5 6

7 8

9 10

11 12

13

14 15 -

16

17

21

22 23 24

predictions.

sentiment: 1D array of class labels

Taking gradient steps

3

4

5

Take two parameters errors and feature.

Compute the dot product of errors and feature.

1 → def feature_derivative(errors, feature):

Return the derivative

Your code should be analogous to the following Python function:

Compute the dot product of errors and feature

 $\ell\ell(\mathbf{w}) = \sum_{i=1}^{N} \left((\mathbf{1}[y_i = +1] - 1)\mathbf{w}^\intercal h(\mathbf{w}_i) - \ln\left(1 + \exp\left(-\mathbf{w}^\intercal h(\mathbf{x}_i)\right)\right) \right)$

1 - def compute_log_likelihood(feature_matrix, sentiment, coefficients):

lp = np.sum((indicator-1)*scores - np.log(1, + np.exp(-scores)))

scores = np.dot(feature_matrix, coefficients)

initial_coefficients: 1D array containing initial values of coefficients

The function returns the last set of coefficients after performing gradient ascent.

step_size: a parameter controlling the size of the gradient steps

max_iter: number of iterations to run gradient ascent

Initialize vector coefficients to initial_coefficients.

Once in a while, insert code to print out the log likelihood.

features = ['constant'] + features

3

 one 2D array for features one 1D array for class labels The function should do the following: Prepend a new column constant to dataframe and fill it with 1's. This column takes account of the intercept term. Make sure that the constant column appears first in the data frame. Prepend a string 'constant' to the list features. Make sure the string 'constant' appears first in the list. Extract columns in dataframe whose names appear in the list features. Convert the extracted columns into a 2D array using a function in the data frame library. If you are using Pandas, you would use as_matrix() function. Extract the single column in dataframe whose name corresponds to the string label.

Quiz Question: How many features are there in the feature_matrix? Quiz Question: Assuming that the intercept is present, how does the number of features in feature_matrix relate to the number of features in the logistic regression model? Estimating conditional probability with link function 10. Recall from lecture that the link function is given by $P(y_i = +1 | \mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + \exp\left(-\mathbf{w}^\intercal h(\mathbf{x}_i)\right)},$

where the feature vector $h(\mathbf{x}_i)$ represents the word counts of **important_words** in the review \mathbf{x}_i . Write a function

Since the word counts are stored as columns in **feature_matrix**, each i-th row of the matrix corresponds to the

By the rules of matrix multiplication, the score vector containing elements $\mathbf{w}^\intercal h(\mathbf{x}_i)$ is obtained by multiplying

We will now write a function feature_derivative that computes the derivative of log likelihood with respect to a

In the main lecture, our focus was on the likelihood. In the advanced optional video, however, we introduced a transformation of this likelihood---called the log-likelihood---that simplifies the derivation of the gradient and is more numerically stable. Due to its numerical stability, we will use the log-likelihood instead of the likelihood to

The log-likelihood is computed using the following formula (see the advanced optional video if you are curious

Write a function compute_log_likelihood that implements the equation. The function would be analogous to the

 $egin{bmatrix} h(\mathbf{x}_1)^{\intercal} \ h(\mathbf{x}_2)^{\intercal} \ dots \ h(\mathbf{x}_N)^{\intercal} \end{bmatrix} = egin{bmatrix} h_0(\mathbf{x}_1) & h_1(\mathbf{x}_1) & \cdots & h_D(\mathbf{x}_1) \ h_0(\mathbf{x}_2) & h_1(\mathbf{x}_2) & \cdots & h_D(\mathbf{x}_2) \ dots & dots & \ddots & dots \ h_0(\mathbf{x}_N) & h_1(\mathbf{x}_N) & \cdots & h_D(\mathbf{x}_N) \end{bmatrix}$

 $[score] = [feature_matrix] \mathbf{w} = \begin{bmatrix} h(\mathbf{x}_1)^{\mathsf{T}} \\ h(\mathbf{x}_2)^{\mathsf{T}} \\ \vdots \\ h(\mathbf{x}_N)^{\mathsf{T}} \end{bmatrix} \mathbf{w} = \begin{bmatrix} h(\mathbf{x}_1)^{\mathsf{T}} \mathbf{w} \\ h(\mathbf{x}_2)^{\mathsf{T}} \mathbf{w} \\ \vdots \\ h(\mathbf{x}_N)^{\mathsf{T}} \mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{w}^{\mathsf{T}} h(\mathbf{x}_1) \\ \mathbf{w}^{\mathsf{T}} h(\mathbf{x}_2) \\ \vdots \\ \mathbf{w}^{\mathsf{T}} h(\mathbf{x}_N) \end{bmatrix}$

Compute derivative of log likelihood with respect to a single coefficient

Using the function written in #8, extract two arrays feature_matrix and sentiment. The 2D array

feature_matrix would contain the content of the columns given by the list important_words. The 1D array

single coefficient w_j. The function accepts two arguments: errors: vector whose i-th value contains $1[y_i = +1] - P(y_i = +1|\mathbf{x}_i, \mathbf{w})$ feature: vector whose i-th value contains $h_i(\mathbf{x}_i)$ This corresponds to the j-th column of feature_matrix.

Return the dot product. This is the derivative with respect to a single coefficient w_j.

function that takes gradient steps towards the optimum. Write a function logistic_regression to fit a logistic regression model using gradient ascent. The function accepts the following parameters:

13. Now we are ready to implement our own logistic regression. All we have to do is to write a gradient ascent

2. Predict the class probability $P(y_i=+1|\mathbf{x}_i,\mathbf{w})$ using your $\mathsf{predict_probability}$ function and save it to variable

3. Compute indicator value for ($y_i=+1$) by comparing **sentiment** against +1. Save it to variable **indicator**.

Compute the errors as difference between indicator and predictions. Save the errors to variable errors.

For each j-th coefficient, compute the per-coefficient derivative by calling feature_derivative with the j-th

At the end of day, your code should be analogous to the following Python function (with blanks filled in):

coefficients = np.array(initial_coefficients) # make sure it's a numpy array

Predict $P(y_i = +1|x_1,w)$ using your predict_probability() function

for j in xrange(len(coefficients)): # loop over each coefficient

Recall that feature_matrix[:,j] is the feature column associated

compute the derivative for coefficients[j]. Save it in a variable

add the step size times the derivative to the current coefficient

column of feature_matrix. Then increment the j-th coefficient by (step_size*derivative).

2 - def logistic_regression(feature_matrix, sentiment, initial_coefficients,

Compute indicator value for (y_i = +1)

Compute the errors as indicator - predictions

14. Now, let us run the logistic regression solver with the parameters below:

Now, we write some code to compute class predictions. We do this in two steps:

Quiz question: How many reviews were predicted to have positive sentiment?

Which words contribute most to positive & negative sentiments

1 coefficients = list(coefficients[1:]) # exclude intercept

First compute the scores using feature_matrix and coefficients using a dot product.

Then apply threshold 0 on the scores to compute the class predictions. Refer to the formula above.

16. We will now measure the classification accuracy of the model. Recall from the lecture that the classification

Quiz question: What is the accuracy of the model on predictions made above? (round to 2 digits of accuracy)

Sort all the (word, coefficient_value) tuples by coefficient_value in descending order. Save the sorted list of

18. Compute the 10 words that have the most positive coefficient values. These words are associated with positive

Next, we repeat this exercises on the 10 most negative words. That is, we compute the 10 words that have the

initial_coefficients = a 194-dimensional vector filled with zeros

```
# Checking whether log likelihood is increasing if itr <= 15 or (itr <= 100 and itr % 10 == 0) or (itr <= 1000 and itr %
 25
 26
              100 == 0) \
 27 -
              or (itr <= 10000 and itr % 1000 == 0) or itr % 10000 == 0:
                  lp = compute_log_likelihood(feature_matrix, sentiment, coefficients)
 28
 29
                  print 'iteration %*d: log likelihood of observed labels = %.8f' % \
 30
                       (int(np.ceil(np.log10(max_iter))), itr, lp)
31
          return coefficients
```

Quiz question: As each iteration of gradient ascent passes, does the log likelihood increase or decrease?

15. Recall from lecture that class predictions for a data point x can be computed from the coefficients w using the

```
    Recall that in the earlier assignment, we were able to compute the "most positive words". These are words

that correspond most strongly with positive reviews. In order to do this, we will first do the following:

    Treat each coefficient as a tuple, i.e. (word, coefficient_value). The intercept has no corresponding word, so

  throw it out.
```

tuples to word_coefficient_tuples.

sentiment.

Ten "most negative" words

Your code should be analogous to the following:

(important_words, coefficients)]

 $\label{eq:accuracy} \operatorname{accuracy} = \frac{\# \ \operatorname{correctly} \ \operatorname{classified} \ \operatorname{data} \ \operatorname{points}}{\# \ \operatorname{total} \ \operatorname{data} \ \operatorname{points}}$

Now, word_coefficient_tuples contains a sorted list of (word, coefficient_value) tuples. The first 10 elements in this list correspond to the words that are most positive. Ten "most positive" words

2 word_coefficient_tuples = [(word, coefficient) for word, coefficient in zip

3 word_coefficient_tuples = sorted(word_coefficient_tuples, key=lambda x:x[1],

Quiz question: Which word is not present in the top 10 "most positive" words?

most negative coefficient values. These words are associated with negative sentiment.

Quiz question: Which word is not present in the top 10 "most negative" words? 标记为完成