Analyzing product sentiment

In the last class, we focused on classifiers. Now, we will apply our knowledge to analyze product sentiment, and understanding the types of errors a classifier makes. We also built an exciting IPython notebook for analyzing the sentiment of real product reviews.

In this assignment, we are going to explore this application further, training a sentiment analysis model using a set of key polarizing words, verify the weights learned to each of these words, and compare the results of this simpler classifier with those of the one using all of the words.

Follow the rest of the instructions on this document to complete your task. When you are done, submit a document with each of your answers.

Learning outcomes

- Execute sentiment analysis code with the IPython notebook
- Load and transform real, text data
- Using the *.apply()* function to create new columns (features) for our model
- Compare results of two models, one using all words and the other using a subset of the words
- Compare learned models with majority class prediction
- Examine the predictions of a sentiment model
- Build a sentiment analysis model using a classifier

Download the data a starter code

Before getting started, you will need to download the dataset and the starter IPython notebook that we used in the module

- Download the product review dataset here in SFrame format: amazon_baby.gl.zip
- Download the sentiment analysis notebook from the module here: Analyzing product sentiment.ipynb
- Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data file.

Now you are ready to get started!

What you will do

Now you are ready! We are going to do four tasks in this assignment. There are several results you need to gather along the way to enter into the quiz after this reading.

In the IPython notebook above, we used the word counts for all words in the reviews to train the sentiment classifier model. Now, we are going to follow a similar path, but only use this subset of the words:

```
selected_words = ['awesome', 'great', 'fantastic', 'amazing', 'love', 'horrible'
  , 'bad', 'terrible', 'awful', 'wow', 'hate']
```

Often, ML practicioners will throw out words they consider "unimportant" before training their model. This procedure can often be helpful in terms of accuracy. Here, we are going to throw out all words except for the very few above. Using so few words in our model will hurt our accuracy, but help us interpret what our classifier is doing.

1. **Use .apply() to build a new feature with the counts for each of the selected words**: In the given notebook, we created a column "word_count" with the word counts for each review. Our first task is to create a new column in the products SFrame with the counts for each selected_word above, and, in the process, we will see how the method .apply() can be used to create new columns in our data (our features) and how to use a Python function, which is an extremely useful concept to grasp!

Our first goal is to create a column *products['awesome']* where each row contains the number of times the word *'awesome'* showed up in the review for the corresponding product, and 0 if the review didn't show up. One way to do this is to look at each row '*word_count*' column and follows this logic:

- If 'awesome' shows up in the word counts for a particular product (row of the products SFrame), then we know how often 'awesome' appeared in the review,
- if 'awesome' doesn't appear in the word counts, then it didn't appear in the review, and we should set the count for 'awesome' to 0 in this review.

We could use a for loop to iterate this logic for each row of the products SFrame, but this approach would be really slow, because the SFrame is not optimized for this being accessed with a for loop. Instead, we will use the .apply() method to iterate the the logic above for each row of theproducts['word_count'] column (which, since it's a single column, has type SArray). Read about using the .apply() method on an SArray here.

We are now ready to create our new columns:

- First, you will use a Python function to define the logic above. You will write a function called *awesome_count* which takes in the word counts and returns the number of times '*awesome'* appears in the reviews.

A few tips:

1. Each entry of the 'word_count' column is of Python type dictionary.

2. If you have a dictionary called *dict*, you can access a field in the dictionary using:

```
dict['awesome']
```

but only if '*awesome*' is one of the fields in the dictionary, otherwise you will get a nasty error.

3. In Python, to test if a dictionary has a particular field, you can simply write:

```
if 'awesome' in dict
```

In our case, if this condition doesn't hold, the count of '*awesome*' should be 0.

Using these tips, you can now write the awesome_count function.

- Next, you will use *.apply()* to iterate *awesome_count* for each row of *products['word_count']* and create a new column called '*awesome*' with the resulting counts. Here is what that looks like:

```
products['awesome'] = products['word_count'].apply(awesome_count)
```

And you are done! Check the *products* SFrame and you should see the new column you just create.

- Repeat this process for the other 11 words in *selected_words*. (Here, we described a simple procedure to obtain the counts for each *selected_word*. There are other more efficient ways of doing this, and we encourage you to explore this further.)
- Using the *.sum()* method on each of the new columns you created, answer the following questions: **Question 1: Out of the *selected_words*, which one is most used in the dataset? Which one is least used?**

2. **Create a new sentiment analysis model using only the selected_words as features.** In the given IPython Notebook, we used word counts for all words as features for our sentiment classifier. Now, you are just going to use the *selected_words*:

- Use the same train/test split as in the IPython Notebook from lecture:

```
train_data,test_data = products.random_split(.8, seed=0)
```

- Train a logistic regression classifier(use *graphlab.logistic_classifier.create*) using just the *selected_words*. Hint: you can use this parameter in the *.create()* call to specify the features used to be exactly the new columns you just created:

```
features=selected_words
```

Call your new model: *selected_words_model.*

- You will now examine the weights the learned classifier assigned to each of the 11 words in *selected_words* and gain intuition as to what the ML algorithm did for your data using these features. In GraphLab Create, a learned model, such as the *selected_words_model*, has a field 'coefficients', which lets you look at the learned coefficients. You can access it by using:

```
selected_words_model['coefficients']
```

The result has a column called 'value', which contains the weight learned for each feature.

Using this approach, sort the learned coefficients according to the 'value' column using *.sort()*. **Question 2: Out of the 11 words in *selected_words*, which one got the most positive weight? Which one got the most negative weight? Do these values make sense for you?**

3. **Comparing the accuracy of different sentiment analysis models:**

```
.evaluate(test_data)
```

Question 3: What is the accuracy of the *selected_words_model* on the *test_data*? What was the accuracy of the *sentiment_model* that we learned using all the word counts in the given IPython Notebook? What is the accuracy majority class classifier on this task? How do you compare the different learned models with the baseline approach where we are just predicting the majority class?

Hint: we discussed the majority class classifier in lecture, which simply predicts that every data point is from the most common class. This is baseline is something we definitely want to beat with models we learn from data.

4. **Interpreting the difference in performance between the models:** To understand why the model with all word counts performs better than the one with only the *selected_words*, we will now examine the reviews for a particular product.

   - We will investigate a product named '*Baby Trend Diaper Champ'*. (This is a trash can for soiled baby diapers, which keeps the smell contained.)

   - Just like we did for the reviews for the giraffe toy in the given IPython Notebook, before we start our analysis you should select all reviews where the product name is *'Baby Trend Diaper Champ'*. Let's call this table *diaper_champ_reviews*.

   - Again, use the *sentiment_model* to predict the sentiment of each review in *diaper_champ_reviews* and sort the results according to their '*predicted_sentiment*'.

   - **Question 4: What is the *'predicted_sentiment'* for the most positive review for *'Baby Trend Diaper Champ'* according to the *sentiment_model* from the given IPython Notebook?**

   - Now use the *selected_words_model* you learned using just the *selected_words* to predict the sentiment most positive review you found above. *Hint: if you sorted the diaper_champ_reviews in descending order (from most positive to most negative), this command will be helpful to make the prediction you need:*

```
selected_words_model.predict(diaper_champ_reviews[0:1], output_type
  ='probability')
```

   - **Question 4: Why is the *predicted_sentiment* for the most positive review found using the model with all word counts (*sentiment_model*) much more positive than the one using only the selected_words (selected_words_model)?** *Hint: examine the text of this review, the extracted word counts for all words, and the word counts for each of the selected_words, and you will see what each model used to make its prediction.*