0086_bag_of_words

June 2, 2018

1 Using free text for classification - 'Bag of Words'

There may be times in healthcare where we would like to classify patients based on free text data we have for them. Maybe, for example, we would like to predict likely outcome based on free text clinical notes.

Using free text requires methods known as 'Natural Language Processing'.

Here we start with one of the simplest techniques - 'bag of words'.

In a 'bag of words' free text is reduced to a vector (a series of numbers) that represent the number of times a word is used in the text we are given. It is also posible to look at series of two, three or more words in case use of two or more words together helps to classify a patient.

A classic 'toy problem' used to help teach or develop methos is to try to judge whether people rates a film as 'like' or 'did not like' based on the free text they entered into a widely used internet film review database (www.imdb.com).

Here will will use 50,000 records from IMDb to convert each review into a 'bag of words', which we will then use in a simple logistic regression machine learning model.

We can use raw word counts, but in this case we'll add an extra transformation called tf-idf (frequency–inverse document frequency) which adjusts values according to the number fo reviews that use the word. Words that occur across many reviews may be less discriminatory than words that occur more rarely, so tf-idf reduces the value of those words used frequently across reviews.

This code will take us through the following steps:

1) Load data from internet, and split into training and test sets.

2) Clean data - remove non-text, convert to lower case, reduce words to their 'stems' (see below for details), and reduce common 'stop-words' (such as 'as', 'the', 'of').

3) Convert cleaned reviews in word vectors ('bag of words'), and apply the tf-idf transform.

4) Train a logistic regression model on the tr-idf transformed word vectors.

5) Apply the logistic regression model to our previously unseen test cases, and calculate accuracy of our model.

1.1 Load data

This will load the IMDb data from the web. It is loaded into a Pandas DataFrame.

```
In [67]: import pandas as pd
         import numpy as np

         file_location = 'https://raw.githubusercontent.com/MichaelAllen1966/1805_nlp_basics/ma
         data = pd.read_csv(file_location)
```

Show the size of the data set (rows, columns).

```
In [12]: data.shape

Out[12]: (50000, 2)
```

Show the data fields.

```
In [5]: list(data)

Out[5]: ['review', 'sentiment']
```

Show the first record review and recorded sentiment (which will be 0 for not liked, or 1 for liked)

```
In [11]: print ('Review:')
         print (data['review'].iloc[0])
         print ('\nSentiment (label):')
         print (data['sentiment'].iloc[0])

Review:
I have no read the novel on which "The Kite Runner" is based. My wife and daughter, who did, tl

Sentiment (label):
1
```

1.2 Splitting the data into training and test sets

Split the data into 70% training data and 30% test data. The model will be trained using the training data, and accuracy will be tested using the independent test data.

```
In [13]: from sklearn.model_selection import train_test_split
         X = list(data['review'])
         y = list(data['sentiment'])
         X_train, X_test, y_train, y_test = train_test_split(
             X,y, test_size = 0.3, random_state = 0)
```

1.3 Defining a function to clean the text

This function will:

1) Remove ant HTML commands in the text

2) Remove non-letters (e.g. punctuation)

3) Convert all words to lower case

4) Remove stop words (stop words are commonly used works like 'and' and 'the' which have little value in nag of words. If stop words are not already installed then open a python terminal and type the two following lines of code (these instructions will also be given when running this code if the stopwords have not already been downloaded onto the computer running this code).

import nltk
nltk.download("stopwords")

5) Reduce words to stem of words (e.g. 'runner', 'running', and 'runs' will all be converted to 'run')

6) Join words back up into a single string

In [21]:
```
def clean_text(raw_review):
    # Function to convert a raw review to a string of words

    # Import modules
    from bs4 import BeautifulSoup
    import re

    # Remove HTML
    review_text = BeautifulSoup(raw_review, 'lxml').get_text()

    # Remove non-letters
    letters_only = re.sub("[^a-zA-Z]", " ", review_text)

    # Convert to lower case, split into individual words
    words = letters_only.lower().split()

    # Remove stop words (use of sets makes this faster)
    from nltk.corpus import stopwords
    stops = set(stopwords.words("english"))
    meaningful_words = [w for w in words if not w in stops]

    # Reduce word to stem of word
    from nltk.stem.porter import PorterStemmer
    porter = PorterStemmer()
    stemmed_words = [porter.stem(w) for w in meaningful_words]

    # Join the words back into one string separated by space
    joined_words = ( " ".join( stemmed_words ))
    return joined_words
```

Now will will define a function that will apply the cleaning function to a series of records (the clean text function works on one string of text at a time).

```
In [25]: def apply_cleaning_function_to_series(X):
             print('Cleaning data')
             cleaned_X = []
             for element in X:
                 cleaned_X.append(clean_text(element))
             print ('Finished')
             return cleaned_X
```

We will call the cleaning functions to clean the text of both the training and the test data. This may take a little time.

```
In [26]: X_train_clean = apply_cleaning_function_to_series(X_train)
         X_test_clean = apply_cleaning_function_to_series(X_test)
```

```
Cleaning data
Finished
Cleaning data
Finished
```

1.4 Create 'bag of words'

The 'bag of words' is the word vector for each review. This may be a simple word count for each review where each position of the vector represnts a word (returned in the 'vocab' list) and the value of that position represents the number fo times that word is used in the review.

The function below also returns a tf-idf (frequency–inverse document frequency) which adjusts values according to the number fo reviews that use the word. Words that occur across many reviews may be less discriminatory than words that occur more rarely. The tf-idf transorm reduces the value of a given word in proportion to the number of documents that it appears in.

The function returns the following:

1) vectorizer - this may be applied to any new reviews to convert the revier into the same word vector as the training set.

2) vocab - the list of words that the word vectors refer to.

3) train_data_features - raw word count vectors for each review

4) tfidf_features - tf-idf transformed word vectors

5) tfidf - the tf-idf transformation that may be applied to new reviews to convert the raw word counts into the transformed word counts in the same way as the training data.

Our vectorizer has an argument called 'ngram_range'. A simple bag of words divides reviews into single words. If we have an ngram_range of (1,2) it means that the review is divided into single words and also pairs of consecutiev words. This may be useful if pairs of words are useful, such as 'very good'. The max_features argument limits the size of the word vector, in this case to a maximum of 10,000 words (or 10,000 ngrams of words if an ngram may be mor than one word).

```
In [28]: def create_bag_of_words(X):
             from sklearn.feature_extraction.text import CountVectorizer

             print ('Creating bag of words...')
             # Initialize the "CountVectorizer" object, which is scikit-learn's
             # bag of words tool.

             # In this example features may be single words or two consecutive words
             vectorizer = CountVectorizer(analyzer = "word",   \
                                          tokenizer = None,    \
                                          preprocessor = None, \
                                          stop_words = None,   \
                                          ngram_range = (1,2), \
                                          max_features = 10000)

             # fit_transform() does two functions: First, it fits the model
             # and learns the vocabulary; second, it transforms our training data
             # into feature vectors. The input to fit_transform should be a list of
             # strings. The output is a sparse array
             train_data_features = vectorizer.fit_transform(X)

             # Convert to a NumPy array for easy of handling
             train_data_features = train_data_features.toarray()

             # tfidf transform
             from sklearn.feature_extraction.text import TfidfTransformer
             tfidf = TfidfTransformer()
             tfidf_features = tfidf.fit_transform(train_data_features).toarray()

             # Take a look at the words in the vocabulary
             vocab = vectorizer.get_feature_names()

             return vectorizer, vocab, train_data_features, tfidf_features, tfidf
```

We will apply our bag_of_words function to our training set. Again this might take a little time.

```
In [29]: vectorizer, vocab, train_data_features, tfidf_features, tfidf  = (
                 create_bag_of_words(X_train_clean))
```

```
Creating bag of words...
```

Let's look at the some items from the vocab list (positions 40-44). Some of the words may seem odd. That is because of the stemming.

```
In [40]: vocab[40:45]
```

```
Out[40]: ['accomplish', 'accord', 'account', 'accur', 'accuraci']
```

And we can see the raw word count represented in train_data_features.

```
In [43]: train_data_features[0][40:45]
```

```
Out[43]: array([0, 0, 1, 0, 0], dtype=int64)
```

If we look at the tf-idf transform we can see the value reduced (words occuring in many documents will have their value reduced the most)

```
In [44]: tfidf_features[0][40:45]
```

```
Out[44]: array([0.        , 0.        , 0.06988648, 0.        , 0.        ])
```

1.5 Training a machine learning model on the bag of words

Now we have transformed our free text reviews in vectors of numebrs (representing words) we can apply many different machine learning techniques. Here will will use a relatively simple one, logistic regression.

We'll set up a function to train a logistic regression model.

```
In [49]: def train_logistic_regression(features, label):
             print ("Training the logistic regression model...")
             from sklearn.linear_model import LogisticRegression
             ml_model = LogisticRegression(C = 100,random_state = 0)
             ml_model.fit(features, label)
             print ('Finsihed')
             return ml_model
```

Now we will use the tf-idf tranformed word vectors to train the model (we could use the plain word counts contained in 'train_data_features' (rather than using'tfidf_features'). We pass both the features and the known label corresponding to the review (the sentiment, either 0 or 1 depending on whether a person likes the film or not).

```
In [50]: ml_model = train_logistic_regression(tfidf_features, y_train)
```

```
Training the logistic regression model...
Finsihed
```

1.6 Applying the bag of words model to test reviews

We will now apply the bag of words model to test reviews, and assess the accuracy.

We'll first apply our vectorizer to create a word vector for review in the test data set.

```
In [51]: test_data_features = vectorizer.transform(X_test_clean)
         # Convert to numpy array
         test_data_features = test_data_features.toarray()
```

As we are using the tf-idf transform, we'll apply the tfid transformer so that word vectors are transformed in the same way as the training data set.

```
In [53]: test_data_tfidf_features = tfidf.fit_transform(test_data_features)
         # Convert to numpy array
         test_data_tfidf_features = test_data_tfidf_features.toarray()
```

Now the bit that we really want to do - we'll predict the sentiment of the all test reviews (and it's just a single line of code!). Did they like the film or not?

```
In [55]: predicted_y = ml_model.predict(test_data_tfidf_features)
```

Now we'll compare the predicted sentiment to the actual sentiment, and show the overall accuracy of this model.

```
In [66]: correctly_identified_y = predicted_y == y_test
         accuracy = np.mean(correctly_identified_y) * 100
         print ('Accuracy = %.0f%%' %accuracy)

Accuracy = 87%
```

87% accuracy. That's not bad for a simple Natural Language Processing model, using logistic regression.