Data Mining
Assignment 3

Ali Gholami

Department of Computer Engineering & Information Technology
Amirkabir University of Technology

https://aligholamee.github.io
aligholami7596@gmail.com

Abstract

Keywords.

1 Text Preprocessing

1.1 Separating Train and Test Data

In the first step, we need to properly separate the train samples and their labels from each other. To achieve this, we have implemented the function *extract labels* to does that for us. Columns are based on the current dataset which is called *spam collection dataset*.

```
def extract_labels(data):
        return (dataset['v2'], data['v1'])
```

1.2 Extracting Features of Texts

In this section, we'll be using *Bag of Words* model from *Scikit-learn* to to turn the text content into numerical feature vectors. This model does the following steps to turn texts into feature maps.

1. Assign a fixed ID to every word occurring in any document (for instance by building a dictionary from words to integer indices).

2. Each unique word in our dictionary will correspond to a feature (descriptive feature).

Scikit-learn has a high level component which will create feature vectors for us *CountVectorizer*.

```
count_vect = CountVectorizer()
X_train = count_vect.fit_transform(train_data)
```

It worths mentioning that the parameters and return value of the *fit_transform* function is as follows:

```
        Parameters
        ----------
        raw_documents : iterable
        An iterable which yields either str, unicode or file objects.

        Returns
        -------
        X : array, [n_samples, n_features]
        Document-term matrix.
```

1.3 Issue With Occurrences

Occurrence count is a good start but there is an issue: longer documents will have higher average count values than shorter documents, even though they might talk about the same topics.

To avoid these potential discrepancies it suffices to **divide** the number of occurrences of each word in a document by the total **number of words in the document**. We'll delve into the formal representation of the *TF-IDF*. It is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

1.3.1 Term Frequency

We denote the raw count of a word in a document as $f_{t,d}$. It show the number of times that term t has occurred in the document d. There are also other approaches to denote the number of occurrences of a term in a document and they are trying to illustrate the weighted frequency of each term in a document. Some of them are:

- Boolean Frequencies
- Logarithmically Scaled Frequency
- Augmented Frequencies

1.3.2 Inverse Document Frequency

The inverse document frequency is a measure of how much information the word provides, that is, whether the term is common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient.

$$idf(t, D) = \log \frac{N}{|\{d \epsilon D : t \epsilon d\}}$$

Where N is the number of documents in the database D. The denominator illustrates the number of documents in which they include the term t.

1.3.3 TF-IDF

A high weight in tf–idf is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents.

$$tfidf(t, d, D) = tf(t, d) * idf(t, D)$$

So far we have defined metrics that can be conducted to improve the features extracted from the emails in the first step. We'll change the transformer to use the *tf-idf* metric to extract features. Using this approach will result in a more precise text classification. This will **normalize the frequency of redundant words**.

```
count_vect = CountVectorizer()
X_train = count_vect.fit_transform(train_data)

tf_idf = TfidfTransformer()
X_train = tf_idf.fit_transform(X_train)
```

2 Text Classification

Now that we have our features, we can train a classifier to try to predict the category of a post as *Spam* or *Ham* (Not Spam). There are various algorithms which can be used for text classification. We will start with the most simplest one called Naive Bayes. For this task, we only need the two class Naive Bayes classifier. We have also included the testing section of the code. The result will be available in the *predict* variable.

```
clf = MultinomialNB().fit(train_data, train_labels)
test_data = count_vect.transform(test_data)
test_data = tf_idf.transform(test_data)
print("Accuracy: ", accuracy_score(test_labels, predicted))
```

In order to use *Recall* and *Precision* metrics of Scikit-learn, we have to convert the labels into binary format. We'll use *Label Binarizer* to achieve this.

```
lb = preprocessing.LabelBinarizer()
predicted_binarized = lb.fit_transform(predicted)
test_labels_binarized = lb.fit_transform(test_labels)
print("Recall: ", recall_score(test_labels_binarized, predicted_binarized))
print("Precision: ", precision_score(test_labels_binarized, predicted_binarized))
```

References

[1] Prashant Gupta, *Cross-Validation in Machine Learning*. Towards Data Science, Jun 5, 2017.

[2] Scikit-Learn, *sklearn.tree.DecisionTreeClassifier*. http://scikit-learn.org.

[3] Scikit-Learn, *Tuning the hyper-parameters of an estimator*. http://scikit-learn.org.