Untitled

April 27, 2019

0.0.1 Melika Adabinezhad 810195536

0.1 assignment 3 report

1 naive bayes spam classification

1.0.1 feature explanation

The two main features i've chosen are the presence of the word "call" and a phone number in the email. The reason is that someone you know usually doesn't ask you to call them providing their phone number and if you don't know the person very well, you'll probably reply their email and won't call them most of the time. As I observed, most spam emails ask you to call them or text them or reply to them.

1.0.2 algorithm

The method of this classification is naive bayes inference. In this family of algorithms, every pair of features being classified is independent of each other. The learning model is the same as bayes rule except the fact that denominator is same for both conditional probabilities we calculate for different classes so we won't count it in calculations. We must also normalize the text.

```
In [1]: #NORMALIZATION
        import numpy as np
        import string
        from nltk.corpus import stopwords
        from nltk.tokenize import word_tokenize
        from nltk.stem import PorterStemmer
        import nltk
        # nltk.download('punkt')
        # nltk.download('stopwords')


        def convert_lowercase(s):
            return s.lower()


        def remove_punctuation(s):
            return s.translate(str.maketrans('', '', string.punctuation))
```

```
def remove_whitespace(s):
    return s.strip()


def remove_stopwords(tokens):
    stop_words = set(stopwords.words('english'))
    return [i for i in tokens if not i in stop_words]


def apply_stemming(tokens):
    stemmer = PorterStemmer()
    return [stemmer.stem(i) for i in tokens]


def normalize(sen_dataframe):
    for index, row in sen_dataframe.iterrows():
        row['text'] = convert_lowercase(row['text'])
        row['text'] = remove_punctuation(row['text'])
        row['text'] = remove_whitespace(row['text'])
        row['text'] = word_tokenize(row['text'])
        row['text'] = remove_stopwords(row['text'])
        row['text'] = apply_stemming(row['text'])

    return sen_dataframe
```

```
In [5]: import pandas
        import numpy as np
        import matplotlib.pyplot as plt



        def is_phone_number(s):
            try:
                int(s)
                if(len(s) == 11):
                    return True
                return False
            except ValueError:
                return False


        def calculate_prob(nfrac, ntotal):
            return nfrac/ntotal


        def calculate_num_of_distinct_words(data_frame):
            words = set()
```

```
    count = 0
    for row in data_frame:
        for word in row:
            if word not in words:
                count += 1
                words.add(word)

    return count


def count_word(data_frame, w):
    count = 0
    for row in data_frame:
        for word in row:
            if word == w:
                count += 1

    return count


def count_all_words(data_frame):
    count = 0
    for row in data_frame:
        count += len(row)
    return count


def count_phonenumbers(data_frame):
    count = 0
    for row in data_frame:
        for word in row:
            if is_phone_number(word):
                count += 1

    return count


features = ['text', 'win', 'click', 'call', 'claim', 'prize', 'select', 'urgent', 'gua
            'send', 'cash', 'now', 'reply', 'date', 'area', 'award', 'gift', 'answer',


features = apply_stemming(features)

data_frame = pandas.read_csv("data/train_test.csv")
length = len(data_frame)

test_dataframe = data_frame.iloc[0: 1014]
normalize(test_dataframe)
```

```python
data_frame = data_frame.iloc[1014: length]

normalized_vectors = normalize(data_frame)



total_num_of_docs = len(data_frame)

num_of_distinct_words = calculate_num_of_distinct_words(
    normalized_vectors.text.values)

hams = data_frame.groupby('type').get_group('ham')
num_of_hams = len(hams)
ham_prob = calculate_prob(num_of_hams, total_num_of_docs)


spams = data_frame.groupby('type').get_group('spam')
num_of_spams = len(spams)
spam_prob = calculate_prob(num_of_spams, total_num_of_docs)


hams = normalized_vectors.groupby('type').get_group('ham')
num_of_ham_words = count_all_words(hams.text.values)

phone_num_ham = count_phonenumbers(hams.text.values)
phone_prob_ham = calculate_prob(
    phone_num_ham+1, num_of_ham_words+num_of_distinct_words)

ham_probs = {}
for f in features:
    num = count_word(hams.text.values, f)
    prob = calculate_prob(
        num+1, num_of_ham_words+num_of_distinct_words)
    ham_probs[f] = prob


spams = normalized_vectors.groupby('type').get_group('spam')
num_of_spam_words = count_all_words(spams.text.values)

phone_num_spam = count_phonenumbers(spams.text.values)
phone_prob_spam = calculate_prob(
    phone_num_spam+1, num_of_spam_words+num_of_distinct_words)

spam_probs = {}
for f in features:
    num = count_word(spams.text.values, f)
    prob = calculate_prob(
```

```python
        num+1, num_of_spam_words+num_of_distinct_words)
    spam_probs[f] = prob



count_correct_train = 0
for index, row in data_frame.iterrows():
    s = row['text']
    spam_p = [spam_prob]
    ham_p = [ham_prob]

    seen = set()
    phone_seen = False
    for word in s:
        if word in features and word not in seen:
            seen.add(word)
            ham_p.append(ham_probs[word])
            spam_p.append(spam_probs[word])
        elif is_phone_number(word) and not(phone_seen):
            spam_p.append(phone_prob_spam)
            ham_p.append(phone_prob_ham)
            phone_seen = True

    if(len(spam_p) == 1):
        spam_p.append(calculate_prob(1, num_of_distinct_words))

    spam_seen_evidence = 1
    for p in spam_p:
        spam_seen_evidence = spam_seen_evidence * p

    ham_seen_evidence = 1
    for p in ham_p:
        ham_seen_evidence = ham_seen_evidence * p

    if(ham_seen_evidence > spam_seen_evidence):
        predict = 'ham'
    else:
        predict = 'spam'

    if(predict == row['type']):
        count_correct_train += 1



count_correct_test = 0
correct_detected_spams = 0
num_spams = len(test_dataframe.groupby('type').get_group('spam'))
```

```
detected_spams = 0

for index, row in test_dataframe.iterrows():
    s = row['text']
    spam_p = [spam_prob]
    ham_p = [ham_prob]

    seen = set()
    phone_seen = False
    for word in s:
        if word in features and word not in seen:
            seen.add(word)
            ham_p.append(ham_probs[word])
            spam_p.append(spam_probs[word])
        elif is_phone_number(word) and not(phone_seen):
            spam_p.append(phone_prob_spam)
            ham_p.append(phone_prob_ham)
            phone_seen = True

    if(len(spam_p) == 1):
        spam_p.append(calculate_prob(1, num_of_distinct_words))

    spam_seen_evidence = 1
    for p in spam_p:
        spam_seen_evidence = spam_seen_evidence * p

    ham_seen_evidence = 1
    for p in ham_p:
        ham_seen_evidence = ham_seen_evidence * p

    if(ham_seen_evidence > spam_seen_evidence):
        predict = 'ham'
    else:
        predict = 'spam'

    if(predict=='spam'):
        detected_spams += 1

    if(predict == row['type']):
        count_correct_test += 1
        if(row['type']=='spam'):
            correct_detected_spams += 1
```

1.0.3 overfit

When our model doesn't generalize well from our training data to unseen data, we say overfitting has happened. In other words, A model that has learned the noise instead of the signal is considered "overfit" because it fits the training dataset and has high variance but has poor fit with new datasets. We can't know how well our model will perform on new data until we actually test it. We can split our initial dataset into separate training and test subsets. This method can approximate of how well our model will perform on new data. If our model does much better on the training set than on the test set, then we're likely overfitting. I take 1/5 of train data as test data.

1.0.4 does overfit exist with relation to main features

Fortunately, no.

```
In [6]: print("correctness of train data: ", str(count_correct_train*100/total_num_of_docs), "
        print("correctness of test data: ", str(count_correct_test*100/len(test_dataframe)), "
```

```
correctness of train data:  95.2439625431247 %
correctness of test data:  94.0828402366864 %
```

1.0.5 analytics

There are false positives on test and false negatives on evaluation.

```
In [7]: recall = correct_detected_spams / num_spams
        print("Recall = CorrectDetectedSpams / Spams: ", recall)

        precision = correct_detected_spams / detected_spams
        print("Precision = CorrectDetectedSpams / DetectedSpams: ", precision)

        accuracy = count_correct_test/len(test_dataframe)
        print("Accuracy = CorrectDetected / Total: ", accuracy)
```

```
Recall = CorrectDetectedSpams / Spams:  0.7687074829931972
Precision = CorrectDetectedSpams / DetectedSpams:  0.8129496402877698
Accuracy = CorrectDetected / Total:  0.9408284023668639
```