Assignment-3 report

Shaik Imthiyash cs21b074

April 2024

Classification of Emails Spam vs Non-Spam

1 Modelling as a binary classification problem

In the context of Email classification we consider it to be binary classification with the labels being either Spam or Non-Spam and datapoints as vectors.

let x be the email and y being it's corresponding label,

Then the data can be represented as following,

$$\{X|y\} = \{x_1, x_2, x_3, ..., x_d|y\}$$

Here,

- X the vector form of the data point (Email).
- y the label for the email $y \in \{+1, 0\}$ $+1$ for Spam and 0 for Non-Spam.
- x_i is the feature that represents whether the i^{th} word of our dictionary is present in the email X or not, $x_i \in \{1, 0\}$.
- d the dimensions of the vector or also defined as the total no of words present in our dictionary.

2 Data Conversion

2.1 Dataset Selection

The dataset I chose for this binary classification problem for training is the Deysi/spam-detection-dataset dataset available at the following url: https://huggingface.co/datasets/Deysi/spam-detection-dataset.

It provides an ≈10.9k Emails with their corresponding texts including the subject and label in which 8.18k are used for training purposes and the other for testing.

2.2 Training Data Extraction

The train dataset contains 2 columns of the form "TEXT" and "LABEL", where each row of first column contains a text form of the email and each row of the second column is word i.e either "spam" or "not-spam".

First we need to convert the labels from text (string) to integer (0 for non-spam and 1 for spam).

The following code achieves the desired translation of labels.

```
# loading the dataset

dataset = load_dataset("Deysi/spam-detection-dataset")

# Extracting training data from the datset

train_dataset_text = dataset['train']['text']
train_dataset_target = dataset['train']['label']

# Coverting the dataset targets to 1(spam) or 0(non-spam)

for i in range(0,len(train_dataset_target)):
    if train_dataset_target[i] == 'spam':
        train_dataset_target[i] = 1
    else:
        train_dataset_target[i] = 0
```

2.3 Dictionary Construction

Here we are required to traverse through all the emails present in the training data and obtain all the words neglecting the numerics and punctuation symbols.

This is achieved by using the built-in word_tokenizer function from nltk module.

The dictionary I used has the items of the following format : $\{'\text{sample}' : [x, y]\}$

Here,

- 'sample' is a word in our dictionary.
- x represents the probability of the word 'sample' occuring in all the emails with label 0.
- y represents the probability of the word 'sample' occuring in all the emails with label 1.

The following code shown in *Figure 1* handles the construction of dictionary,

Here we also consider laplace smoothing by adding 1 to the numerator and denominator while calculating the probability in the following step $\forall x$ and y,

$$my\_dict[x][y] = (my\_dict[x][y] + 1)/(c[y] + 1)$$

assuring that the probability $p_x^y > 0 \ \forall x$ and y

```
for i in range(0,len(train_dataset_text)):
    words = nltk.word_tokenize(train_dataset_text[i])
    l = set()
    for j in words:
        if j not in l:
            l.add(j)
            y = train_dataset_target[i]
            if j in my_dict:
                my_dict[j][y] += 1
            else:
                my_dict[j] = [int(y==0),int(y==1)]

for i in my_dict:
    my_dict[i][0] = (my_dict[i][0]+1)/(c[0]+1)
    my_dict[i][1] = (my_dict[i][1]+1)/(c[1]+1)
```

Figure 1: Dictionary construction and laplace smoothing

2.4 Converting Email to Datapoint

Since we got the dictionary now we need to convert each email to a vector or datapoint.

We consider it to be a row vector of d values, where each value is either 0 or 1 representing whether the i^{th} word is present in the email or not.

The following code represents converting the email to datapoint,

```
def vec(words):
    x = []
    for i in my_dict:
        x.append(int(i in words))
    return np.array(x)

train = []
for i in range(0,len(train_dataset_target)):
    train.append(vec(nltk.word_tokenize(train_dataset_text[i])))
```

Figure 2: Email to Datapoint conversion

3 Training Algorithms

3.1 Naive-Bayes Algorithm

Using Naive-Bayes algorithm we can find the probability using the given formula,

Let $x_{test} = \{f_1, f_2, f_3, ..., f_d\} \in \{0,1\}^d$, where $f_i \in \{0,1\} \forall i$

$$P(y \mid x) = \frac{P(x)P(x \mid y) \cdot P(y)}{P(x)}$$

$$P((y_{test} = y) \mid x) \propto \Pi_{k=1}^{d} (p_k^y)^{f_k} . (1 - p_k^y)^{(1-f_k)} . p(y_{test} = y)$$

Here,

- $P((y_{test} = y)/x)$ is the posterior probability of the label y_{test} being equal to y given the input features x.

- d is the no of words present in the dictionary or dimensions of the data vector.
- p_k^y is the probability of k^{th} word in the dictionary occuring in all the emails with label y.
- f_k is the k^{th} value in the data vector of given test email i.e either 1 or 0.

Using the above formula we calculate both $P(y_{test} = 1/x)$ and $P(y_{test} = 0/x)$.

If the $P(y_{test} = 1/x) > P(y_{test} = 0/x)$ then we predict the value of y_{test} to be 1 else the value of $y_{test} = 0$.

The algorithmic implementation is as given below,

```
# Naive-Bayes algorithm implementation

cnt = 0;j = 0
# directory = 'test'
# for filename in os.listdir(directory):
#     filepath = os.path.join(directory, filename)
#     if os.path.isfile(filepath):
#         with open(filepath, 'r') as file:
#             content = file.read()
for content in test_dataset_text:
    words = nltk.word_tokenize(content)
    p0 = c[0]/(c[0]+c[1]);p1 = c[1]/(c[0]+c[1])
    for i in my_dict:
        if i in words:
            p0 *= (my_dict[i][0])
            p1 *= (my_dict[i][1])
        else:
            p0 *= (1-my_dict[i][0])
            p1 *= (1-my_dict[i][1])
    cnt += (test_dataset_target[j] != (p0 <= p1))
    j += 1

print('Accuracy is ',100*(1-(cnt/len(test_dataset_text))))
```

Figure 3: Naive-Bayes Algorithm Implementation

- **spam-detection-dataset(test)**
  Accuracy : 99.413%.
- **spam_ham_dataset**
  Accuracy : 74.648%.
- **enron_spam(test)**
  Accuracy : 73.450%.

3.2 Perceptron-Algorithm

Here we first go through the training data to obtain an optimum w using the following steps,

- step_i : Iterate through the test datapoints and their corresponding labels until convergence.
- step_ii : We predict label of the given train email as follows,
  - Calculate $y = w^T x$.

  - if $y >= 0$ then predict label as 0(non-spam) else as 1(spam).

- step_iii : Check whether the label matches the given train label.
  - if yes proceed with the next email.
  - else update our w using the below formula,

$$w^{t+1} = w^t + xy'$$

    * $y' = +1$ when the predicted label $y = 0$
    * $y' = -1$ when the predicted label $y = 1$

The algorithmic implementation of Perceptron is as follows,

```
# Perceptron Algorithm

def pred(a):
    return int(a < 0)

def vec(words):
    x = []
    for i in my_dict:
        x.append(int(i in words))
    return np.array(x)

train = []
for i in range(0,len(train_dataset_target)):
    train.append(vec(nltk.word_tokenize(train_dataset_text[i])))

w = np.zeros(len(my_dict))
while True:
    change = np.array(w)
    for j in range(0,len(train_dataset_text)):
        if not(pred(np.dot(w,train[j])) == train_dataset_target[j]):
            y = 1 - 2*train_dataset_target[j]
            w += train[j]*y
    if list(w) == list(change):
        break

cnt = 0
j = 0
# directory = 'test'
# for filename in os.listdir(directory):
#     filepath = os.path.join(directory, filename)
#     if os.path.isfile(filepath):
#         with open(filepath, 'r') as file:
#             content = file.read()
for content in test_dataset_text:
    words = nltk.word_tokenize(content)
    x = []
    for i in my_dict:
        x.append(int(i in words))
    cnt += (pred(np.dot(w,x)) != test_dataset_target[j])
    j += 1

print('Accuracy is',100*(1-(cnt/len(test_dataset_text))),'%')
```

Figure 4: Perceptron Algorithm Implementation

- **spam-detection-dataset(test)**
  Accuracy : 99.670%.
- **spam_ham_dataset**
  Accuracy : 77.528%.
- **enron_spam(test)**
  Accuracy : 54.70%.

3.3 Support Vector Machine Algorithm(SVM)

The goal of Support Vector Machine (SVM) is to find the optimal hyperplane that best separates the data points of different classes in the feature space.

SVM aims to find the hyperplane that maximizes the margin, which is the distance between the hyperplane and the nearest data points of each class, known as support vectors. Maximizing the margin helps to improve the generalization ability of the classifier and enhances its robustness to new data points.

Here we use the in-built module SVC from sklearn.SVM of python.

The following is the implementation of the SVM Algorithm,

```
# SVM Algorithm Implementation

from sklearn.svm import SVC

X = np.array(train)
y = np.array(train_dataset_target)
print(X.shape,y.shape)

svm_model = SVC(kernel='linear')
svm_model.fit(X, y)

j = 0;cnt = 0
for content in test_dataset_text:
    words = nltk.word_tokenize(content)
    x = []
    for i in my_dict:
        x.append(int(i in words))
    predicted_label = svm_model.predict(np.array(x).reshape(1,len(x)))
    cnt += (predicted_label[0] != test_dataset_target[j])
    j += 1
print('Accuracy is',100*(1-(cnt/len(test_dataset_text))),'%')
```

Figure 5: SVM Algorithm Implementation

- **spam-detection-dataset(test)**
  Accuracy : 99.780%.
- **spam_ham_dataset**
  Accuracy : 79.907%.
- **enron_spam(test)**
  Accuracy : 66.80%.

Here we can see that SVM performs better than Perceptron and Naive-Bayes Algorithm.

3.4 Logistic Regression

Here we implement the Logistic regression in the form of Gradient Descent algorithm where the Update step is as follows,

$$W^{t+1} = W^t + \eta \left(\sum_{i=1}^{n} x_i (y_i - g((W^t)^T x_i)) \right), g(\theta) = \frac{1}{1 + e^{-\theta}}$$

We take step-size $\eta = 0.01$.

```
# Logistic Regression Implementation
import math

def g(w,x):
    a = np.dot(w.T,np.array(x))
    return 1/(1+math.exp(-a))

w_log_reg = np.zeros(len(x))

c = 0;nt = 0.001
while c<=20:
    sum = np.zeros(len(train[0]))
    for i in range(0,len(train_dataset_text)):
        sum += np.array(train[i])*(train_dataset_target[i]-g(w_log_reg,train[i]))
    w_log_reg = w_log_reg + nt*(sum)
    c += 1
    print(c)

print(w_log_reg)

j = 0;cnt = 0
for content in test_dataset_text:
    words = nltk.word_tokenize(content)
    x = []
    for i in my_dict:
        x.append(int(i in words))
    y_pred = 0
    if g(w_log_reg,np.array(x)) > 0.5:
        y_pred = 1
    cnt += (y_pred != test_dataset_target[j])
    j += 1
    print(j,cnt)

print('Accuracy is',100*(1-(cnt/len(test_dataset_text))),'%')
```

Figure 6: Logistic Regression Implementation

- **spam-detection-dataset(test)**
  Accuracy : 99.552%.
- **spam_ham_dataset**
  Accuracy : 62.385%.
- **enron_spam(test)**
  Accuracy : 49.651%.

4 Algorithm Selection

Based on the performance of above algorithms on different datasets the combined graph for accuracy over different datasets is as follows,

From the above graph we can say that SVM algorithm produces more accuracy compared to the other 3 algorithms.

Hence the final SVM algorithmic implementation that takes input from a directory named test is as follows,

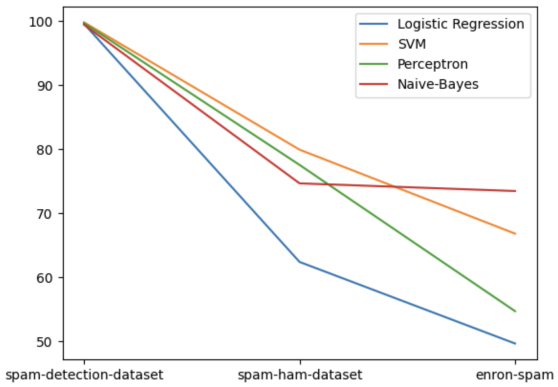

Figure 7: Naive-bayes vs Perceptron vs SVM vs Logistic Regeresssion

```
# SVM Algorithm Implementation

def vec(words):
    x = []
    for i in my_dict:
        x.append(int(i in words))
    return np.array(x)

train = []
for i in range(0,len(train_dataset_target)):
    train.append(vec(nltk.word_tokenize(train_dataset_text[i])))
    print(i)

X = np.array(train)
y = np.array(train_dataset_target)
print(X.shape,y.shape)

svm_model = SVC(kernel='linear', C = 1.0)
svm_model.fit(X, y)

j = 0;test_labels = []
directory = 'test'
for filename in os.listdir(directory):
    filepath = os.path.join(directory, filename)
    if os.path.isfile(filepath):
        with open(filepath, 'r') as file:
            content = file.read()
            words = nltk.word_tokenize(content)
            x = []
            for i in my_dict:
                x.append(int(i in words))
            predicted_label = svm_model.predict(np.array(x).reshape(1,len(x)))
            test_labels.append(predicted_label)
            if predicted_label == 0:
                print("Non-Spam")
            else:
                print("Spam")
            j += 1
            print(j)
```

Figure 8: Final Implementation

The Algorithm works as follows,

It trains on the data obtained from spam-detection-dataset(train) and takes input from a directory named 'test' and outputs either "spam" or "Non-spam" based on predicted label.

The label number i.e {1,0} is appended to the test_labels list inorder to make it easy for comparision with actual labels.