Intro to ML HW1

September 18, 2019

In [1]:
```
"""
Student name: Aniket Bhatnagar
NYU id: N14602070
Net id: ab8700
"""

import os
import numpy as np
```

In [2]:
```
"""
Code and explanation for Q1 contained in this section
"""

def get_data_from_file(file_path):
    """
    Read from file_path and return list_texts and list_labels from the same.
    """
    list_texts = []
    list_labels = []
    with open(file_path) as f:
        x = f.read()
        data_rows = x.split('\n')
        for data_row in data_rows:
            if data_row == '':  # Would be the case with last line of txt files
                continue
            row_elements = data_row.split(' ')
            label = row_elements[0]
            text_data = ' '.join(row_elements[1:])

            list_labels.append(label)
            list_texts.append(text_data)

    return list_texts, list_labels

# Data directory path
data_dir_path = '../hw1_data/'
train_file_path = os.path.join(data_dir_path, 'spam_train.txt')
test_file_path = os.path.join(data_dir_path, 'spam_test.txt')
```

```
        # Read data from train and test file paths
        train_texts, train_labels = get_data_from_file(train_file_path)
        test_texts, test_labels = get_data_from_file(test_file_path)

        # Split train data into train and validation sets
        val_texts = train_texts[4000:]
        train_texts = train_texts[:4000]
        val_labels = train_labels[4000:]
        train_labels = train_labels[:4000]

        # Print length of train and validation texts and labels lists
        print ("No of items in train_texts", len(train_texts))
        print ("No of items in train_labels", len(train_labels))
        print ("No of items in val_texts", len(val_texts))
        print ("No of items in val_labels", len(val_labels))

        # Measuring performance of final classifier would be a problem if
        # validation dataset was not created, because validation set helps to
        # identify best modelling algorithm and its hyper-parameters.
        # And in its absence one would use test set for the same, which is a
        # bad practice as then decision to choose between algorithms or
        # hyper-parameters is based on the test set performance. This can make
        # the model biased towards the data distribution of test set.
```

```
No of items in train_texts 4000
No of items in train_labels 4000
No of items in val_texts 1000
No of items in val_labels 1000
```

In [3]:
```
        """
        Solution for Q2 contained in this section
        """

        def get_vocabulary(list_texts):
            """
            Get the list of words which occur 30 or more times,
            in the list of texts provided as argument to the function.
            """

            # Build dict word_doc_frequency
            # which stores in how many documents is a word occuring
            word_doc_frequency = {}
            for text in list_texts:
                words_list = text.split(' ')
                words_set = set(words_list)  # Since there could be duplicate words in a text
```

```
        for word in words_set:
            try:
                word_doc_frequency[word] += 1
            except KeyError:
                word_doc_frequency[word] = 1

    # Add only those words to vocabulary which have document count >= 30
    in_vocab_words = []
    for word in word_doc_frequency.keys():
        if word_doc_frequency[word] >= 30:
            in_vocab_words.append(word)

    return in_vocab_words


def convert_list_text_to_list_feature_vectors(list_texts, word_index_map):
    """
    Converts a list of texts to a list of feature vectors where
    each feature vector is of length equal to the no of words in vocabulary.
    Let the list of feature vectors be list_feature_vectors, then
    list_feature_vectors[i][j] is 1 if jth word in vocabulary is present in ith text.
    :param list_texts: List of texts passed
    :param word_index_map: Dictionary s.t. keys are words in vocab and
                           values are their index in the vocab
    :return list_feature_vectors: Defined in doc strings
    """

    list_feature_vectors = []
    list_words_in_vocab = word_index_map.keys()
    no_words_in_vocab = len(list_words_in_vocab)
    for text in list_texts:
        words_in_text = text.split()
        words_in_text_n_vocab = set(words_in_text) & set(list_words_in_vocab)

        feature_vector = np.zeros(no_words_in_vocab)

        for word in words_in_text_n_vocab:
            word_ind = word_index_map[word]
            feature_vector[word_ind] = 1

        list_feature_vectors.append(feature_vector)

    return list_feature_vectors


# Put together list of words in vocabulary basis train_texts
words_in_vocab = get_vocabulary(train_texts)
word_index_map = dict(zip(words_in_vocab, range(len(words_in_vocab))))
```

```
        # Convert text_lists into feature_vectors basis the word_index_map
        train_features = convert_list_text_to_list_feature_vectors(
            train_texts, word_index_map
        )
        val_features = convert_list_text_to_list_feature_vectors(
            val_texts, word_index_map
        )
        test_features = convert_list_text_to_list_feature_vectors(
            test_texts, word_index_map
        )

        # Convert list of feature vectors to array of feature vectors
        train_features_array = np.array(train_features)
        val_features_array = np.array(val_features)
        test_features_array = np.array(test_features)

        # Print shape of each features_array
        print ("Shape of training features array", train_features_array.shape)
        print ("Shape of val features array", val_features_array.shape)
        print ("Shape of test features array", test_features_array.shape)
```

```
Shape of training features array (4000, 2376)
Shape of val features array (1000, 2376)
Shape of test features array (1000, 2376)
```

In [4]:
```
        """
        Solution for Q3 contained in this section
        """

        def train_perceptron(features_array, labels_vector):

            # Obtain basic information from data
            no_attributes = features_array.shape[1]

            # Train perceptron on the given data
            weights = np.random.randn(no_attributes)
            no_updates = 0
            no_epochs = 0
            while True:

                no_local_updates = 0

                for feature_vector, actual_label in zip(features_array, labels_vector):
                    wx = np.dot(feature_vector, weights)
                    sign_wx = 1 if wx >= 0 else -1
```

```
                if actual_label * sign_wx <= 0:
                    weights = weights + actual_label * feature_vector
                    no_updates += 1
                    no_local_updates += 1

            no_epochs += 1

            if no_local_updates == 0:
                break

        return weights, no_updates, no_epochs


    def test_perceptron(weights, features_array, labels_array):

        count_correct = 0
        no_samples = features_array.shape[0]
        for feature_vector, actual_label in zip(features_array, labels_array):
            wx = np.dot(feature_vector, weights)
            y_predicted = 1 if wx >= 0 else -1
            if y_predicted == actual_label:
                count_correct += 1

        fraction_misses = (no_samples - count_correct) / no_samples

        return fraction_misses
```

In [5]:
```
"""
Solution for Q4 contained in this section
"""

# Convert labels to integer type and set to -1 when 0 before training the model
train_labels = [1 if label == '1' else -1 for label in train_labels]
val_labels = [1 if label == '1' else -1 for label in val_labels]
test_labels = [1 if label == '1' else -1 for label in test_labels]

# Run train_perceptron on training dataset
weights, no_updates, no_epochs = train_perceptron(train_features_array, train_labels)
print ("No of mistakes the model made during training process", no_updates)
print ("Total no of epochs model took to complete training", no_epochs)

# Run test_perceptron on training dataset
fraction_misses = test_perceptron(weights, train_features_array, train_labels)
print ("Fraction of misses on training dataset", fraction_misses)

# Run test_perceptron on validation dataset
fraction_misses = test_perceptron(weights, val_features_array, val_labels)
print ("Fraction of misses on validation dataset", fraction_misses)
```

```
No of mistakes the model made during training process 432
Total no of epochs model took to complete training 11
Fraction of misses on training dataset 0.0
Fraction of misses on validation dataset 0.02
```

```
In [6]: """
        Solution for Q5 contained in this section
        """

        # Get indices in weights vector which have most positive and most negative values
        max_weight_value_indices = np.argsort(weights)[::-1][:15]  # For most positive weights
        min_weight_value_indices = np.argsort(weights)[:15]  # For most negative weights

        most_pos_wt_words = []
        most_neg_wt_words = []

        for wt_ind in max_weight_value_indices:
            word_value = words_in_vocab[wt_ind]
            most_pos_wt_words.append(word_value)

        for wt_ind in min_weight_value_indices:
            word_value = words_in_vocab[wt_ind]
            most_neg_wt_words.append(word_value)

        print ("\nWords which correspond to the most +ve weight for SPAM classification\n")
        for word in most_pos_wt_words:
            print (word)
        print ("\nWords which correspond to the most -ve weight for SPAM classification\n")
        for word in most_neg_wt_words:
            print (word)
```

```
Words which correspond to the most +ve weight for SPAM classification

sight
market
deathtospamdeathtospamdeathtospam
click
remov
present
basenumb
our
your
yourself
nbsp
will
over
```

```
most
internet

Words which correspond to the most -ve weight for SPAM classification

wrote
i
prefer
recipi
copyright
version
set
reserv
post
still
team
standard
author
url
but
```

```
In [7]: """
        Solution for Q6 contained in this section
        """
        def train_averaged_perceptron(features_array, labels_vector):

            # Obtain basic information from data
            no_attributes = features_array.shape[1]

            # Train perceptron on the given data
            weights = np.random.randn(no_attributes)
            weights_array = []
            no_updates = 0
            no_epochs = 0
            while True:

                no_local_updates = 0

                for feature_vector, actual_label in zip(features_array, labels_vector):

                    weights_array.append(weights)
                    wx = np.dot(feature_vector, weights)
                    sign_wx = 1 if wx >= 0 else -1

                    if actual_label * sign_wx <= 0:
                        weights = weights + actual_label * feature_vector
                        no_updates += 1
```

```
                    no_local_updates += 1

            no_epochs += 1

            if no_local_updates == 0:
                break

        weights_array = np.array(weights_array)
        final_weights = np.mean(weights_array, axis=0)

        return final_weights, no_updates, no_epochs


    # Run train_averaged_perceptron on training dataset
    weights, no_updates, no_epochs = train_averaged_perceptron(
        train_features_array, train_labels
    )
    print ("No of mistakes the model made during training process", no_updates)
    print ("Total no of epochs model took to complete training", no_epochs)

    # Run test_perceptron on training dataset
    fraction_misses = test_perceptron(weights, train_features_array, train_labels)
    print ("Fraction of misses on training dataset", fraction_misses)

    # Run test_perceptron on validation dataset
    fraction_misses = test_perceptron(weights, val_features_array, val_labels)
    print ("Fraction of misses on validation dataset", fraction_misses)
```

```
No of mistakes the model made during training process 418
Total no of epochs model took to complete training 12
Fraction of misses on training dataset 0.001
Fraction of misses on validation dataset 0.014
```

In [8]:
```
    """
    Solution for Q7 contained in this section
    """

    def train_perceptron(features_array, labels_vector, no_epochs):

        # Obtain basic information from data
        no_attributes = features_array.shape[1]

        # Train perceptron on the given data
        weights = np.random.randn(no_attributes)
        no_updates = 0
        for i in range(no_epochs):
```

```
            for feature_vector, actual_label in zip(features_array, labels_vector):
                wx = np.dot(feature_vector, weights)
                sign_wx = 1 if wx >= 0 else -1

                if actual_label * sign_wx <= 0:
                    weights = weights + actual_label * feature_vector
                    no_updates += 1

        return weights, no_updates


    def train_averaged_perceptron(features_array, labels_vector, no_epochs):

        # Obtain basic information from data
        no_attributes = features_array.shape[1]

        # Train perceptron on the given data
        weights = np.random.randn(no_attributes)
        weights_array = []
        no_updates = 0
        for i in range(no_epochs):

            for feature_vector, actual_label in zip(features_array, labels_vector):

                weights_array.append(weights)
                wx = np.dot(feature_vector, weights)
                sign_wx = 1 if wx >= 0 else -1

                if actual_label * sign_wx <= 0:
                    weights = weights + actual_label * feature_vector
                    no_updates += 1

        weights_array = np.array(weights_array)
        final_weights = np.mean(weights_array, axis=0)

        return final_weights, no_updates
```

In [9]:
```
    """
    Solution for Q8-part a-contained in this section
    """
    # Run train_perceptron on training dataset for different no_epochs
    # And run test_perceptron on validation dataset for different no_epochs
    no_epochs_list = [5, 7, 9, 11, 13, 15]
    for no_epochs in no_epochs_list:
        print ("\nTraining the perceptron for %d epochs" % no_epochs)

        weights, no_updates = train_perceptron(
            train_features_array, train_labels, no_epochs
```

```
        )
        print ("No of mistakes the model made during training process", no_updates)

        # Run test_perceptron on training dataset
        fraction_misses = test_perceptron(weights, train_features_array, train_labels)
        print ("Fraction of misses on training dataset", fraction_misses)

        # Run test_perceptron on validation dataset
        fraction_misses = test_perceptron(weights, val_features_array, val_labels)
        print ("Fraction of misses on validation dataset", fraction_misses)
```

```
Training the perceptron for 5 epochs
No of mistakes the model made during training process 412
Fraction of misses on training dataset 0.00275
Fraction of misses on validation dataset 0.027

Training the perceptron for 7 epochs
No of mistakes the model made during training process 416
Fraction of misses on training dataset 0.0025
Fraction of misses on validation dataset 0.023

Training the perceptron for 9 epochs
No of mistakes the model made during training process 421
Fraction of misses on training dataset 0.0005
Fraction of misses on validation dataset 0.018

Training the perceptron for 11 epochs
No of mistakes the model made during training process 412
Fraction of misses on training dataset 0.0
Fraction of misses on validation dataset 0.02

Training the perceptron for 13 epochs
No of mistakes the model made during training process 443
Fraction of misses on training dataset 0.0
Fraction of misses on validation dataset 0.015

Training the perceptron for 15 epochs
No of mistakes the model made during training process 414
Fraction of misses on training dataset 0.0
Fraction of misses on validation dataset 0.019
```

```
In [11]: """
         Solution for Q8-part b-contained in this section
         """
         # Run train_averaged_perceptron on training dataset for different no_epochs
         # And run test_perceptron on validation dataset for different no_epochs
```

```
no_epochs_list = [5, 7, 9, 11, 13, 15]
for no_epochs in no_epochs_list:
    print ("\nTraining the perceptron for %d epochs" % no_epochs)

    weights, no_updates = train_averaged_perceptron(
        train_features_array, train_labels, no_epochs
    )
    print ("No of mistakes the model made during training process", no_updates)

    # Run test_perceptron on training dataset
    fraction_misses = test_perceptron(weights, train_features_array, train_labels)
    print ("Fraction of misses on training dataset", fraction_misses)

    # Run test_perceptron on validation dataset
    fraction_misses = test_perceptron(weights, val_features_array, val_labels)
    print ("Fraction of misses on validation dataset", fraction_misses)
```

```
Training the perceptron for 5 epochs
No of mistakes the model made during training process 403
Fraction of misses on training dataset 0.003
Fraction of misses on validation dataset 0.017

Training the perceptron for 7 epochs
No of mistakes the model made during training process 401
Fraction of misses on training dataset 0.001
Fraction of misses on validation dataset 0.017

Training the perceptron for 9 epochs
No of mistakes the model made during training process 425
Fraction of misses on training dataset 0.00075
Fraction of misses on validation dataset 0.02

Training the perceptron for 11 epochs
No of mistakes the model made during training process 433
Fraction of misses on training dataset 0.00075
Fraction of misses on validation dataset 0.021

Training the perceptron for 13 epochs
No of mistakes the model made during training process 430
Fraction of misses on training dataset 0.00075
Fraction of misses on validation dataset 0.016

Training the perceptron for 15 epochs
No of mistakes the model made during training process 431
Fraction of misses on training dataset 0.0005
Fraction of misses on validation dataset 0.02
```

In [13]:
```
"""
Solution for Q9 contained in this section
"""
# Put together (train_features_array, val_features_array)
# and (train_labels, val_labels)
combined_train_features = np.concatenate(
    [train_features_array, val_features_array], axis=0
)
combined_train_labels = train_labels + val_labels

# Train the normal perceptron model since best results
# were obtained in last question when normal perceptron
# was used and trained for 13 epochs
weights, no_updates = train_perceptron(
    combined_train_features, combined_train_labels, 13
)
print ("No of mistakes the model made during training process", no_updates)

# Test the trained model on testing data
# Run test_perceptron on test dataset
fraction_misses = test_perceptron(
    weights, test_features_array, test_labels
)
print ("Fraction of misses on test dataset", fraction_misses)
```

```
No of mistakes the model made during training process 508
Fraction of misses on test dataset 0.019
```