CSCU9YE Assignment Report

Student no: 2519302

November 17th 2018

Contents

1 Introduction

This project is tasked with the classification of spam emails, to a respectable degree of accuracy using machine learning models. Specifically, this project will employ a Support Vector Machine, Random Forest Classifier and Multilayer Perceptron Classifier. This report will aim to provide an overview of the datasets used, an understanding of how the models used work, how the models were evaluated and the results obtained from running these models on the datasets mentioned.

2 Specification

This section will walk through the flowchart shown in *Figure 1*.

Load Emails

Firstly, the emails need to be loaded into the program in order to parse through them and begin the pre-processing stage. This involves reading in all the files in the folder and saving them to a variable for manipulation further on in the program.

Pre-Process Data

This stage is where the data is processed to convert case, remove elements such as punctuation and stop words and is where the process of lemmatization happens. This process is also referred to as normalization. These entities are removed due to their inherent lack of relevance to the end classification. Lemmatization on the other hand is done in order to reduce the complexity of the generated dictionary in the next step and thus the overall runtime of the algorithms on the dataset.

Dictionary Creation

This process, mentioned briefly above, entails the tallying of the most commonly occurring words in all the emails in the dataset. This is done such that a correlation can be obtained when reading through similar pieces of text. This allows us to see if the emails share similarities and thus may be classed under a label. The generated dictionary is then compared to the features extracted from each email.

Feature Extraction

The feature extraction phase entails extracting a feature vector for all emails in the dataset. The features contain the number of occurrences of each entity in the email, with respect to the dictionary generated.

Split Dataset

The dataset is split into a test set and a training set. This is done such that the model can be trained on a subset of the dataset as a whole and the model can then be evaluated on the same data. The split is done at approximately 70 / 30, where the training set is 70% of the full dataset and the remaining 30% is used as the test set.

Train Model

The models are trained with a subset of the dataset as mentioned above.

Pre-Trained Model

This process is where the pre-trained model is tested with the test set obtained from when the dataset was split.

Evaluate Trained Model

The outputs of the models used here are evaluated using the performance matrices: Accuracy, Precision, Recall and F1-Score. The unlabelled dataset used here is the same test set, but without any accompanying labels designating the entities classification.

2.1 Pseudocode

Shown below is the pseudocode for the spam classification algorithm:

```
begin spam_classification

    begin load_emails
        load emails into program
        split dataset into test set and training set
    end load_emails

    begin pre_processing
        convert text to lower case
        remove stop_words
        lemmatize text
    end pre_processing

    begin create_dictionary
        generate dictionary
    end create_dictionary

    begin feature_extraction
        extract feature vectors for all emails
    end feature_extraction

    Train models on training set
    Test models on test set
    Generate predictions using models
    Evaluate model predictions

end spam_classification
```

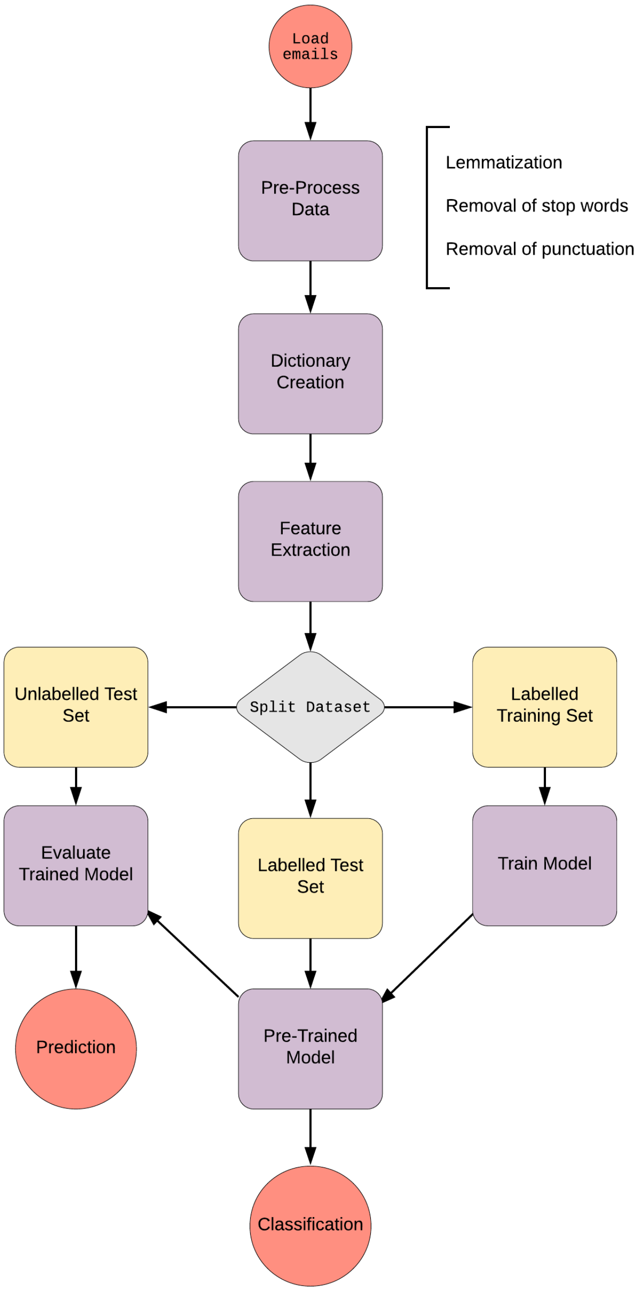

Figure 1: Flowchart for the general Spam Classification process

3 Pre-Processing

This section will aim to cover the different aspects of pre-processing employed in this project. Within the datasets used in this project, we can see that the general steps one would need to take are as follows:

- Lemmatization
- Removal of stop words
- Removal punctuation

Along with the aforementioned steps, the text will be processed to handle case. This is done by converting the text to lower case before any text normalization is done.

3.1 Lemmatization

Lemmatization is a form of text normalization where the aim is to remove inflection and/or derive the base word from a family of words in the dataset [1, 2, 3]. The base of a word, is also known as the **lemma** or the **dictionary form**. This process also leads to a decrease in complexity during the runtime of the algorithm. An example of which is shown in *Table 1* below.

Such mapping also allows us to find all relevant documents using a specific word.

Word		**Lemma**
Cleaning	→	Clean
Cleaner	→	Clean
Cleanliness	→	Clean

Table 1: Showing the mapping of words to its lemma

3.2 Removal of Stop Words

Stop words are words such as "the", "he" or "in". These words do not contribute to the overall meaning of the text and are thus removed during the pre-processing stage [1]. This also tends to be done in order to improve indexing times for larger datasets.

3.3 Removal of Punctuation

Punctuation, like stop words are removed due to their lack of contribution to the overall understanding of the text.

4 Machine Learning Methods

This section will aim to provide some knowledge on the machine learning models employed in this project.

4.1 Support Vector Machine

A Support Vector Machine (SVM) is a discriminative classification method. This method produces a classification by finding an optimal hyperplane that segregates the data points and thus returns classifications [4]. This can be visualized easiest on a 2-Dimensional plot as can be seen below in *Figure 2*:

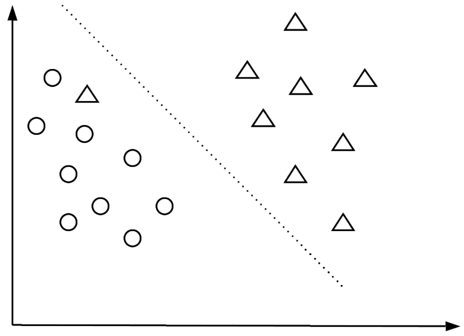

Figure 2: SVM on 2-D plot area

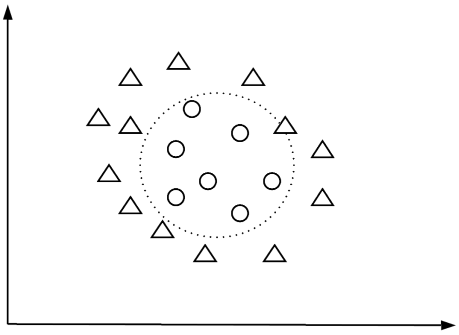

Figure 3: Kernel feature of SVM

Figure 3 above shows an SVM making a classification on data where it would seem like there is no *linear* hyperplane to be found. In order to make a classification, the SVM employs the kernel method [5]. This transformation, introduces a new dimension, by way of suggesting that for every **x** and **x'**, there is a function *k* such that *k* is equivalent to the sum of the squares of **x** and **x'** [6]. This transformation then allows the model to find an optimal hyperplane within this third dimension. Transforming this back into a 2-Dimensional plot, the hyperplane is mapped as a circular boundary around the classified data points.

4.2 Random Forest

Random Forest is a supervised learning algorithm. The way it works is by generating a number of decision trees, all of which generate a classification. This is analogous to each decision tree in the forest 'voting' on a classification. The algorithm obtains a final singular classification by choosing the classification with the most 'votes' [7]. To note, the decision trees generated by this algorithm are not pruned.

4.3 Multilayer Perceptron

A Multilayer Perceptron (MLP), is a form of feedforward artificial neural network. Its general structure consists of an input layer, a number of hidden layers and an output layer. Within each of these layers there are neurons which compute an output based on an activation function. This activation function is different based on what sort of problem the MLP is being applied onto. In the case of this project, the activation function used is the ReLU, the rectified linear unit function.

Each neuron within each layer is connected by a weighted connection. This weight is randomly assigned at the start of the model's training epoch and is refined through the process of backpropagation. This process is how the MLP 'learns'. This process of backpropagation is repeated until the error from the output can no longer be reduced or is zero. At the end of this process, the model returns a classification.

5 Datasets Used

This section will provide a brief description of the datasets used in this project.

5.1 Ling-Spam Dataset

The Ling-Spam dataset used in this project was that provided in the assignment brief. This dataset is a subset of the Ling-Spam corpus, a set of legitimate and spam emails sent via the Linguist list (a mailing list on linguistics). The original corpus consisted of 2893 emails, of which, 481 were spam and the remaining 2412 were legitimate [8]. In the subset provided, the dataset consists of 962 total emails, both spam and legitimate. Of those emails, 702 were used in the training set and the remaining 260 were used as the test set.

This dataset had already been neatly organized into a 50/50 split of spam and non spam emails in each respective subset, this allowed for a fairly simple process of preprocessing the data.

5.2 Enron Dataset

In addition to the dataset referred to above, this project will be using the Enron corpus of spam and legitimate emails. This corpus originally consists of 500,000 emails [9]. The dataset used in this project however is a subset of that corpus, containing 33,699 emails. Of these emails, 17,154 were spam and 16,545 were legitimate. This dataset was split into a training set and test set using a ratio of 70 / 30 respectively.

An important thing to note is that this is not the subset in its entirety, since upon extraction of the files, the anti-virus software on my computer locked certain files from being able to be modified and thus needed to be removed from the dataset in order for the program to run.

6 Measurement Metrics

This section aims to provide an overview of the metrics used to evaluate the models run in this project. For the sake of brevity, the abbreviations used in the equations for each metric are listed below:

- TP = True Positives
- FP = False Positives
- TN = True Negatives
- FN = False Negatives.

6.1 Accuracy

Accuracy is a metric that evaluates the fraction of correct predictions a model has made with respect to the overall predictions the model has made [10]. Formally, Accuracy for binary classification tasks is defined as such:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

6.2 Precision

Precision is a metric that calculates the proportion of correctly classified instances. A high precision value indicates a low amount of False Positives generated [11]. This equation is formally defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

6.3 Recall

Recall is used to find the proportion of actual positive classifications are accurate [11]. A Recall value of 1 indicates that the model provided no False Negatives. This equation is formally defined as such:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

6.4 F1-Score

The F1-Score provides the balance between the Precision and Recall values from the model outputs [12]. In essence, the closer the F1-Score is to 1, the more accurate the model is in its classifications. This equation is defined as such:

$$F1 - Score = 2\left(\frac{Precision * Recall}{Precision + Recall}\right) \tag{4}$$

7 Results and Discussion

This section will provide the results of the models running on the datasets as mentioned above in the report and aim to draw a conclusion from it. To note, for both datasets, the models were run with the same hyperparameters where applicable. This was done such that the produced results were affected solely by the dataset they were run on and not by any additional tuning for optimization within a specific search space.

For the Support Vector Machine, the specific variation used from the *scikit-learn* library was the *LinearSVC*. It was run with a *max_iters* parameter of 5000.

For the Random Forest Classifier, the model was run with a *n_estimators* parameter value of 2000. This means that it will generate 2000 Decision Trees in its Forest.

The Multilayer Perceptron used was *MLPClassifer* from the library and it was run with the following parameters:

- *hidden_layer_sizes* = (6,6,6,6,6,6,6,5,5,5,5,5,2)
- *solver* = lbfgs
- *alpha* = 1e-05
- *random_state* = 2

These parameters were chosen for the *MLPClassifier* through a combination of reading through the *scikit-learn* API for this specific model and looking at how each parameter affects the output of the model by testing different outputs on both datasets with different parameter settings. Additionally, looking through the conventions within this library for parameter settings depending on the size of the dataset greatly assisted in the final parameter settings used.

7.1 Results on Ling-Spam data

The results obtained by running all the aforementioned models on the Ling-Spam dataset is as shown below:

	Accuracy	Precision	Recall	F1-Score
Linear SVM	0.94615	0.93939	0.95385	0.94656
Random Forest Classifier	0.97308	0.95555	0.99230	0.97358
MLP	0.98077	0.98449	0.97692	0.98069

Table 2: Table showing results for machine learning models on Ling-Spam dataset

We can observe from the data as shown in *Table 2* above, the Random Forest Classifier and MLP performed the best in the test set. It is interesting to note however that, although the MLP objectively performed better across the board compared to the Random Forest Classifier, it has a noticeably worse Recall score. This difference is not so significant that it adversely affects the classifications obtained, however it is noteworthy that Random Forest is known to perform well on binary classification problems.

7.2 Results on Enron data

The results as obtained from running the models on the Enron dataset is as follows:

	Accuracy	Precision	Recall	F1-Score
Linear SVM	0.96747	0.97112	0.96573	0.96842
Random Forest Classifier	0.97745	0.97418	0.98238	0.97826
MLP	0.97063	0.97548	0.96745	0.97145

Table 3: Table showing results for machine learning models on Enron dataset

We can immediately see from the data shown in *Table 3* that there are a number of similarities observable from the results on the previous dataset. The Linear SVM is the poorest performing model, the MLP performing the best overall and the Random Forest Classifier performing almost similarly to the MLP, only performing better on Recall score.

7.3 Discussion

From the results as shown in the sections above, we can see that the MLP is objectively the best model to be used for prediction as far as the datasets it has been tested on can attest. However, if accurate classifications are what is desired the tests above show that the Random Forest Classifier is least likely to make incorrect classifications. These results however are not indicative of real world performance. These models were all trained on what are objectively small datasets and tested within a specific search space.

In order to test out how accurate the models were once trained

References

[1] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*, vol. 39. Cambridge University Press, 2008.

[2] H. Jabeen, "Stemming and lemmatization in python." https://www.datacamp.com/community/tutorials/stemming-lemmatization-python, 2018 (accessed November 17, 2018).

[3] K. Fortney, "Pre-processing in natural language machine learning." https://towardsdatascience.com/pre-processing-in-natural-language-machine-learning-898a84b8bd47, 2017 (accessed November 17, 2018).

[4] S. Patel, "Chapter 2 : Svm support vector machine theory." https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72, May.

[5] S. Ray, "Understanding support vector machine algorithm from examples (along with code)." https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/, 2017 (accessed November 17, 2018).

[6] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The annals of statistics*, pp. 1171–1220, 2008.

[7] L. Breiman and A. Cutler, "Random forests," n.d.

[8] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C. D. Spyropoulos, and P. Stamatopoulos, "A memory-based approach to anti-spam filtering for mailing lists," *Information retrieval*, vol. 6, no. 1, pp. 49–73, 2003.

[9] W. W. Cohen, "Enron email dataset." https://www.cs.cmu.edu/ ./enron/, May 2015 (accessed November 19, 2018).

[10] "Classification: Accuracy." https://developers.google.com/machine-learning/crash-course/classification/accuracy, Oct.

[11] "Classification: Precision and recall." https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall, Oct.

[12] J. Brownlee, "Classification accuracy is not enough: More performance measures you can use." https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/, Mar.

A Code

A.1 utility.py

```
import os
import nltk
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from collections import Counter
import sklearn
from sklearn import svm
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
import string


def load_data(init_folder):
        if "enron" in init_folder:
                test_set, training_set, test_set_classification,
                   training_set_classification = load_enron_data(
                   init_folder)
        else:
                test_set, training_set, test_set_classification,
                   training_set_classification = load_lingspam_data
                   (init_folder)

        return test_set, training_set, test_set_classification,
           training_set_classification

def load_lingspam_data(init_folder):
        folders = [folder.path for folder in os.scandir(init_folder
           ) if os.path.isdir(folder)]

        test_set, test_set_classification = lingspam_scraper(
           folders[0])
        training_set, training_set_classification =
           lingspam_scraper(folders[1])

        print("\n...Data has been loaded...\n")

        return test_set, training_set, test_set_classification,
           training_set_classification



def lingspam_scraper(pathname):
        index_skip_subjectline = 2
        list_emails = list()
        files = os.listdir(pathname)
        files.sort()

        for i in range(0, len(files)):
                with open(pathname + "/" + files[i]) as lingspam:
                        lines = lingspam.readlines()
                        list_emails.append(lines[
                           index_skip_subjectline])

        '''
                once the files are sorted, since the datasets are a
                    50/50 split of spam and non-spam
                we can easily label them
        '''
        split_set = int(len(list_emails)/2)
        set_classifications = ([0]*split_set + [1]*split_set)

        return list_emails, set_classifications

def load_enron_data(init_folder):
        folders = [folder.path for folder in os.scandir(init_folder
           ) if os.path.isdir(folder)]

        test_set, test_set_classification = enron_scraper(folders
           [0])
        training_set, training_set_classification = enron_scraper(
           folders[1])

        print("\n...Data has been loaded...\n")

        return test_set, training_set, test_set_classification,
           training_set_classification

def enron_scraper(pathname):
        index_skip_subjectline = 1
        list_emails = list()
        list_ham = list()
        list_spam = list()
        files = os.listdir(pathname)

        for i in range(len(files)):
                if "ham" in files[i]:
                        mail_string = ""
                        with open(pathname + "/" + files[i], "rb")
                           as enron:
                                for line in enron:
                                        mail_string += line.decode(
                                           "latin-1")
                                list_ham.append(mail_string)
                else:
                        mail_string = ""
                        with open(pathname + "/" + files[i], "rb")
                           as enron:
                                for line in enron:
                                        mail_string += line.decode(
                                           "latin-1")
                                list_spam.append(mail_string)
        list_emails = (list_ham + list_spam)
        set_classifications = ([0]*len(list_ham) + [1]*len(
           list_spam))
        return list_emails, set_classifications



def normalize(email):
        stop_words = set(stopwords.words('english'))
        punc = set(string.punctuation)
        lemma = WordNetLemmatizer()

        no_stopwords = " ".join([word for word in email.lower().
           split() if word not in stop_words])
        no_punc = ''.join([char for char in no_stopwords if char
           not in punc])
        normalized_text = " ".join(lemma.lemmatize(word) for word
           in no_punc.split())

        return normalized_text

def pre_process(email_list):
        normalized_emails = list()

        for i in range(len(email_list)):
                normalized_emails.append(normalize(email_list[i]))

        print("Pre-Processing Complete...\n")

        return normalized_emails

def make_dict(email_list):
        all_words = list()

        for i in range(len(email_list)):
                all_words.extend(email_list[i].split())

        dictionary = Counter(all_words)
        dictionary = dictionary.most_common(3000)

        print("Dictionary Creation Complete...\n")

        return dictionary

def extract_feature_vector(email, dictionary):
        feature_vector = list()
        words = email.split()

        for entry in dictionary:
                feature_vector.append(words.count(entry[0]))

        return feature_vector

def feature_extraction(email_list, dictionary):
        features = list()

        for i in range(len(email_list)):
                features.append(extract_feature_vector(email_list[i
            ], dictionary))

print("Feature Extraction Complete...\n")

return features
```

A.2 spam_classifier.py

```
from utility import *

''' Run Machine Learning Models '''

def run_models(dirname):

        ''' Load Emails '''
        testing_set, training_set, test_classification,
           training_classification = load_data(dirname)

        ''' Data Pre-Processing '''
        testing_set = (pre_process(testing_set))
        training_set = pre_process((training_set))

        ''' Dictionary Creation '''
        dictionary = make_dict(training_set)

        ''' Feature Extraction '''
        training_vec = feature_extraction(training_set, dictionary)
        testing_vec = feature_extraction(testing_set, dictionary)

        ''' Model Training '''

        # LinearSVC Classifier
        model_svc = svm.LinearSVC(max_iter=5000)
        model_svc.fit(training_vec, training_classification)

        # Random Forest Classifier
        model_rfc = RandomForestClassifier(n_estimators=2000)
        model_rfc.fit(training_vec, training_classification)

        # MLP Classifier
        model_mlp = MLPClassifier(hidden_layer_sizes
           =(6,6,6,6,6,6,5,5,5,5,5,2), solver="lbfgs", alpha=1e
           -05, random_state=2)
        model_mlp.fit(training_vec, training_classification)

        print("...Training Complete...\n")

        ''' Model Testing and Results '''

        # Linear SVC Results
        model_svc_res = model_svc.predict(testing_vec)

        # Random Forest Classifier Results
        model_rfc_res = model_rfc.predict(testing_vec)

        # Multilayer Perceptron Results
        model_mlp_res = model_mlp.predict(testing_vec)


        print("Results for Machine Learning Models on {}\n".format(
           dirname))
        print("-----------------------------------------------\n")
        print("Linear Support Vector Machine Classifier Results\n")
        print("Accuracy Score: {}".format(metrics.accuracy_score(
           test_classification, model_svc_res)))
        print("Precision Score: {}".format(metrics.precision_score(
           test_classification, model_svc_res)))
        print("Recall: {}".format(metrics.recall_score(
           test_classification, model_svc_res)))
        print("F1 Score: {}".format(metrics.f1_score(
           test_classification, model_svc_res)))
        print("\n-----------------------------------------------\n"
           )
        print("Random Forest Classifier Results\n")
        print("Accuracy Score: {}".format(metrics.accuracy_score(
           test_classification, model_rfc_res)))
        print("Precision Score: {}".format(metrics.precision_score(
           test_classification, model_rfc_res)))
        print("Recall: {}".format(metrics.recall_score(
           test_classification, model_rfc_res)))
        print("F1 Score: {}".format(metrics.f1_score(
           test_classification, model_rfc_res)))
        print("\n-----------------------------------------------\n"
           )
        print("Multilayer Perceptron Classifier (relu) Results\n")
        print("Accuracy Score: {}".format(metrics.accuracy_score(
           test_classification, model_mlp_res)))
        print("Precision Score: {}".format(metrics.precision_score(
           test_classification, model_mlp_res)))
        print("Recall: {}".format(metrics.recall_score(
           test_classification, model_mlp_res)))
        print("F1 Score: {}".format(metrics.f1_score(
           test_classification, model_mlp_res)))
        print("\n-----------------------------------------------\n"
           )


lingspam = "lingspam-dataset"
enron = "enron-dataset"

''' Run Models for Ling-Spam and for Enron '''
run_models(lingspam)
run_models(enron)
```