SPAM DETENTION

STEFANO COSTANZO

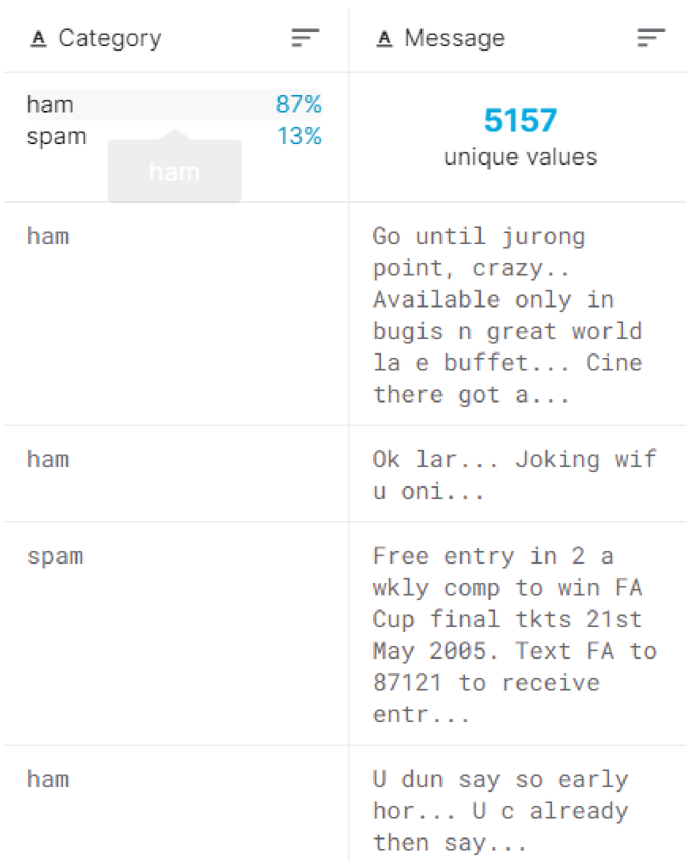
• DATASET

The dataset selected contains of 5157 email with two column:

- Category: Ham or spam
- Boby message of the email

With a Proportion of 87% ham and 13% spam

The target of the project is to realize a spam detention system using machine learning techniques.



Category	Message
ham 87% spam 13%	5157 unique values
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got a...
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entr...
ham	U dun say so early hor... U c already then say...

- **WHAT IS THE ALGORITHM**

The problem needs to apply a **Supervised Learning**: We have examples of **categorized** email (y matrix of yi = (1,0))

Many researchers and academicians have proposed different email spam classification techniques which have been successfully used to classify data into groups. These methods include probabilistic, decision tree, artificial immune system, support vector machine (SVM), artificial neural networks (ANN), and case-based technique:

- Support **Vector Machines** (SVM) has proved over the years to be one of the most powerful and efficient state-of-the-art classification techniques for solving the email spam problem. They are supervised learning models that analyze data and identify patterns used for categorisation and exploring the relationship between variables of interest. SVM algorithms are very potent for the identification of patterns and classifying them into a specific class or group. They can be easily trained and according to some researchers, they outperform many of the popular email spam classification methods

- Several research work have employed **neural network** to classify unwanted emails as spam by applying content-based filtering. These techniques decide the properties by either computing the rate of occurrence of keywords or patterns in the email messages. Literatures show that Neural Network algorithms that are utilised in email filtering attain moderate classification performance. Some of the most popular spam email classification algorithms are Multilayer Perceptron Neural Networks (MLPNNs) and Radial Base Function Neural Networks (RBFNN).

Final Decision: Neauronal network

- Not so common as SVM: It can represent an auxiliary or at least different opportunity to face the spam generation algorithms

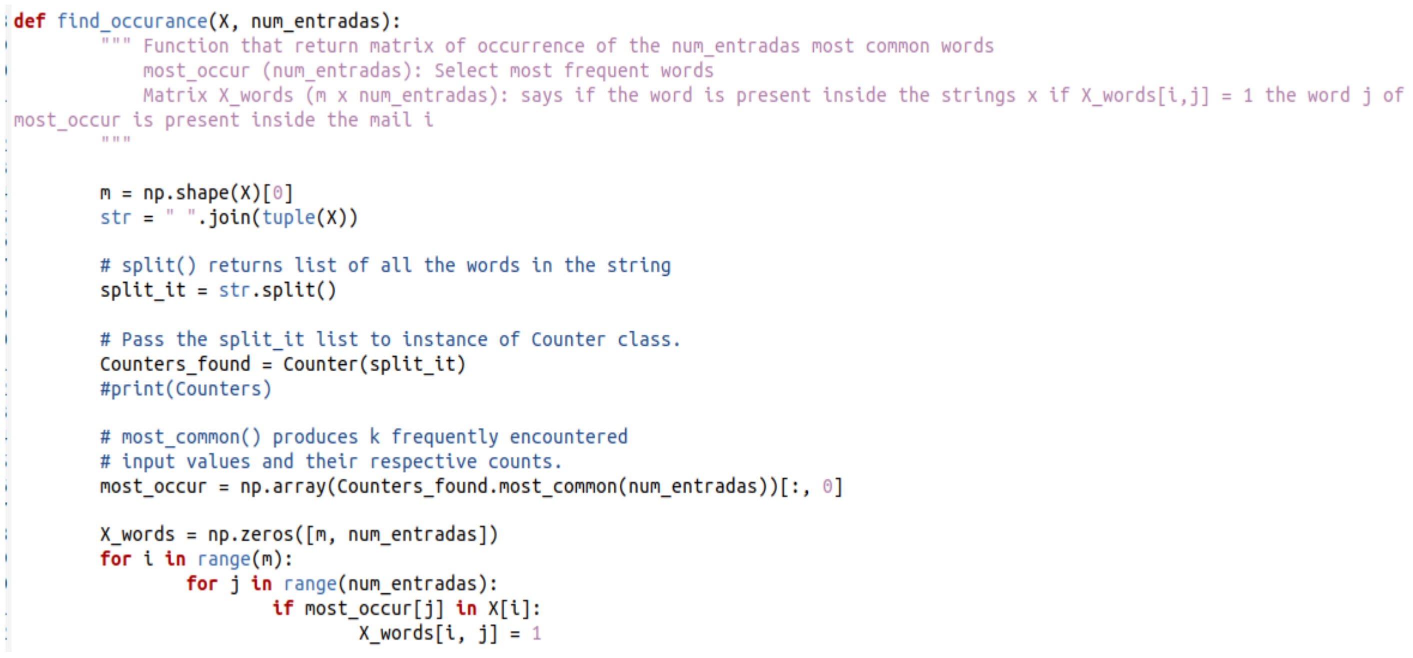
- Quite robust to noise in the training data

- Strong flexibility in configuration: Represent a good example to apply all the analysis and measuments about algorithm of machine learning

- **HOW IT WORKS:**

1) PREPARATION PART

- Divide the X dataset in two parts: **Train** (80%) and **Test** (20%).
- Generate y with values 1 for ham and 0 for spam emails.
- From the input array X of the emails is created an array **most_occur** of the most frequent words. But generally the words that are more indicative of spam or ham.
  Than, most_occur generates the **X_words** matrix (m = num.email, num_entradas), the real input to the Neuronal Network system: Every value of the matrix can be 1 if the word appears in the email or 0 otherwise.
  Here the implementation:

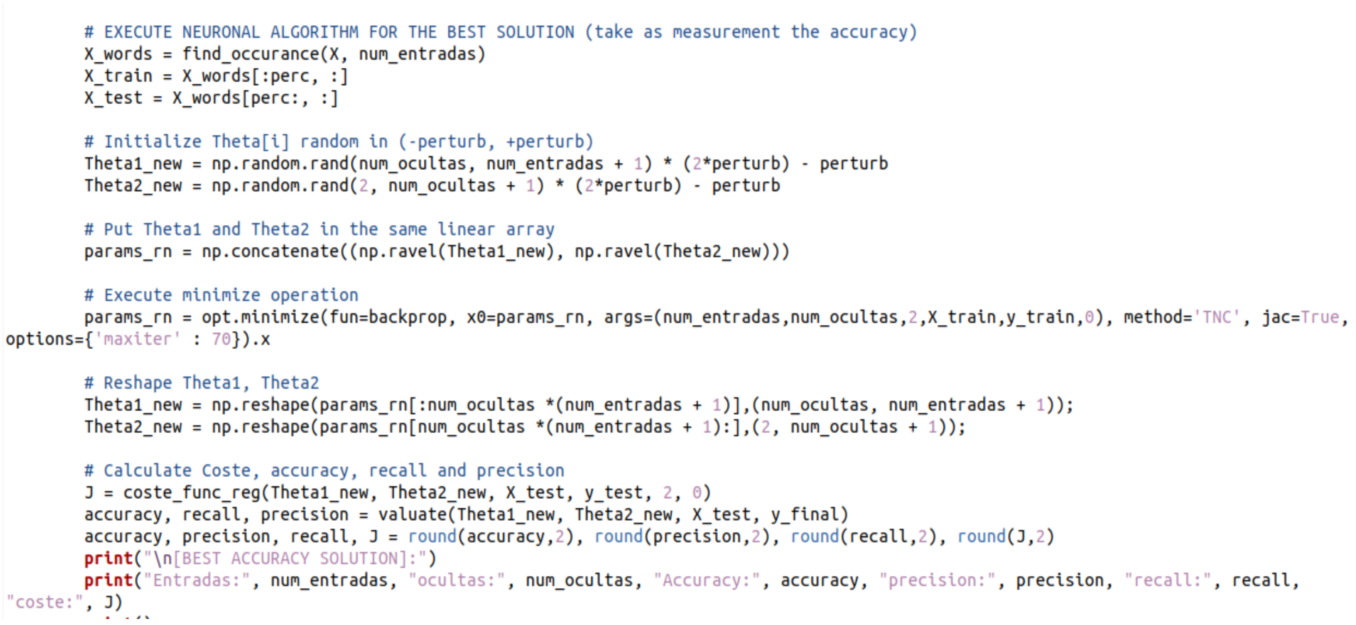


```
def find_occurance(X, num_entradas):
        """ Function that return matrix of occurrence of the num_entradas most common words
            most_occur (num_entradas): Select most frequent words
            Matrix X_words (m x num_entradas): says if the word is present inside the strings x if X_words[i,j] = 1 the word j of
most_occur is present inside the mail i
        """

        m = np.shape(X)[0]
        str = " ".join(tuple(X))

        # split() returns list of all the words in the string
        split_it = str.split()

        # Pass the split_it list to instance of Counter class.
        Counters_found = Counter(split_it)
        #print(Counters)

        # most_common() produces k frequently encountered
        # input values and their respective counts.
        most_occur = np.array(Counters_found.most_common(num_entradas))[:, 0]

        X_words = np.zeros([m, num_entradas])
        for i in range(m):
                for j in range(num_entradas):
                        if most_occur[j] in X[i]:
                                X_words[i, j] = 1
```

2) INITIAL IMPLEMENTATION

The first step is to start with the **simplest implementation**:

NN with less num_entradas (about 500) and num_ocultas (about 25). Only one hidden layer. Small amount of iterations of the opt.minimize function of the TNC algorithm. Applies the NN technique with all its parts: Forward and backword propagations (Take a look to the figure).



```
        # EXECUTE NEURONAL ALGORITHM FOR THE BEST SOLUTION (take as measurement the accuracy)
        X_words = find_occurance(X, num_entradas)
        X_train = X_words[:perc, :]
        X_test = X_words[perc:, :]

        # Initialize Theta[i] random in (-perturb, +perturb)
        Theta1_new = np.random.rand(num_ocultas, num_entradas + 1) * (2*perturb) - perturb
        Theta2_new = np.random.rand(2, num_ocultas + 1) * (2*perturb) - perturb

        # Put Theta1 and Theta2 in the same linear array
        params_rn = np.concatenate((np.ravel(Theta1_new), np.ravel(Theta2_new)))

        # Execute minimize operation
        params_rn = opt.minimize(fun=backprop, x0=params_rn, args=(num_entradas,num_ocultas,2,X_train,y_train,0), method='TNC', jac=True,
options={'maxiter' : 70}).x

        # Reshape Theta1, Theta2
        Theta1_new = np.reshape(params_rn[:num_ocultas *(num_entradas + 1)],(num_ocultas, num_entradas + 1));
        Theta2_new = np.reshape(params_rn[num_ocultas *(num_entradas + 1):],(2, num_ocultas + 1));

        # Calculate Coste, accuracy, recall and precision
        J = coste_func_reg(Theta1_new, Theta2_new, X_test, y_test, 2, 0)
        accuracy, recall, precision = valuate(Theta1_new, Theta2_new, X_test, y_final)
        accuracy, precision, recall, J = round(accuracy,2), round(precision,2), round(recall,2), round(J,2)
        print("\n[BEST ACCURACY SOLUTION]:")
        print("Entradas:", num_entradas, "ocultas:", num_ocultas, "Accuracy:", accuracy, "precision:", precision, "recall:", recall,
"coste:", J)
```

First measurements gives good results**: 95% accuracy** and **98% recall**.

The last value appears really important in our system because the categorized dataset is unbalanced (many more ham that spam email) and because we don’t want many “false negative” in the prediction that categorize as spam email which are not – Maybe important ones.

3) IMPROVE THE MODEL

- *3-D charts and number hidden nodes*

The next step is **to find the best parameters** for the **num_entradas**, that represents the number of features of X_words and the input parameters of the NN scheme, and the **num_ocultas**, the hidden nodes of the NN.

The **3d_chart** function increase the number of this dimensions to reach the overfit limit (when the results begin to get worst) and represents the results in three 3-D charts.

The 3 dimensions are x = num_entradas, y = num_ocultas, and z = (**accuracy, recall, precision**)). Where X = (1000-7000), y = (50-300)

As a result we can see that a good compromise for the num_ocultas is 100-200

Here the figures:

Fig1: **Accuracy** (from 98.5 to 99.1)

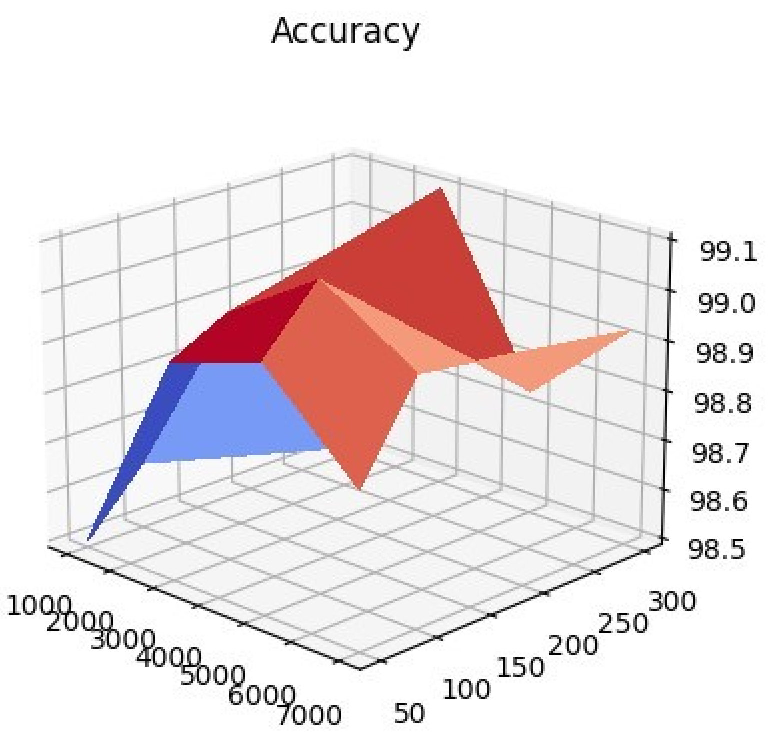


Fig2: **Recall** (from 99.3 to 99.9)

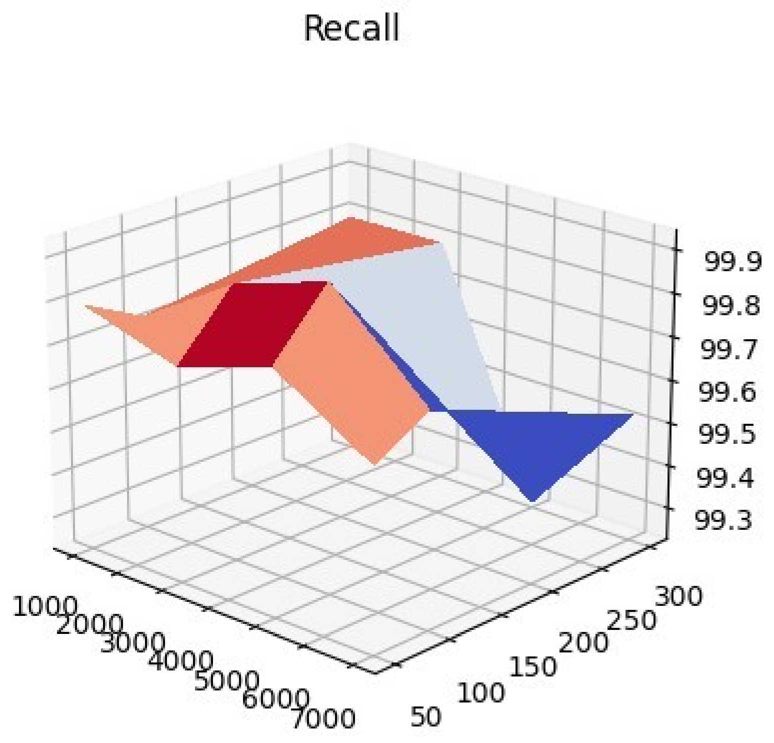
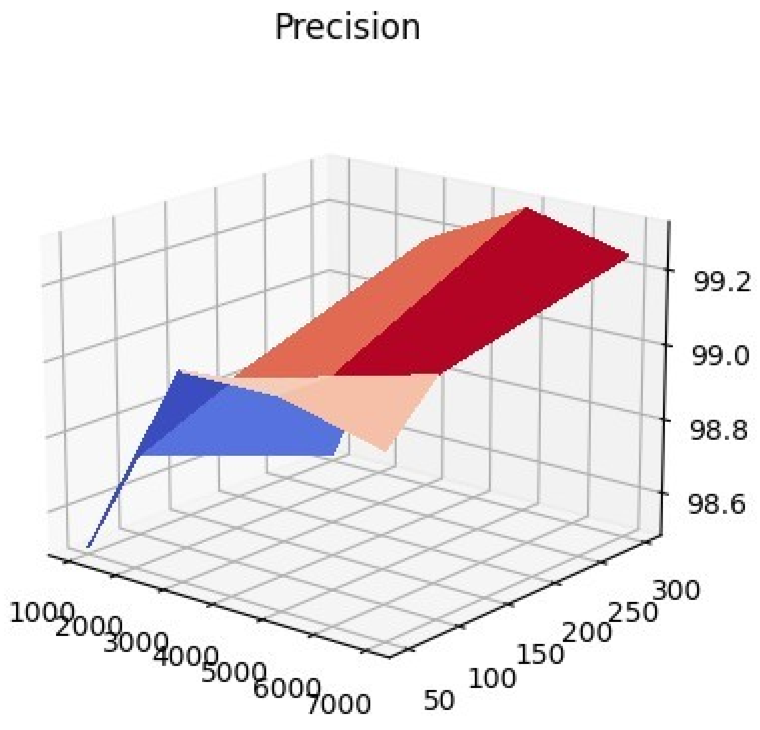


Fig3: **Precision** (from 98.6 to 99.2)



- *K-validation, 2-D chart and number input nodes*

For the small amount of examples is implemented a **k-validation** function with num_ocultas = 100 and K = 5 to find the best number of features of X (num_entradas).

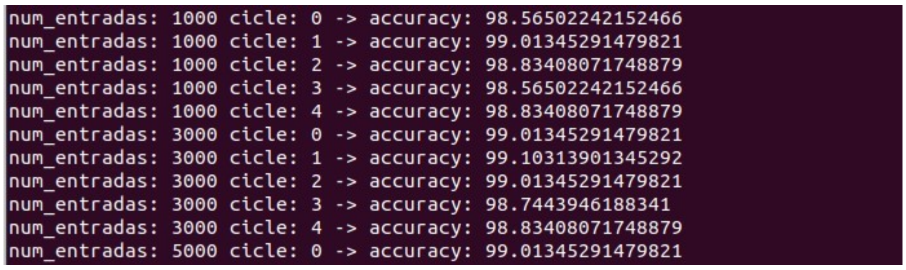
In this case the accuracy is the chosen metric.

As a result 5000-6000 words are a good compromise to not overfit:

Take attention that the NN algorithms takes a lot of computation and ,for this reason, the number of features is not greater than 15000. This limitation does not permit to understand carefully if the system is overfitting after 15000 words.

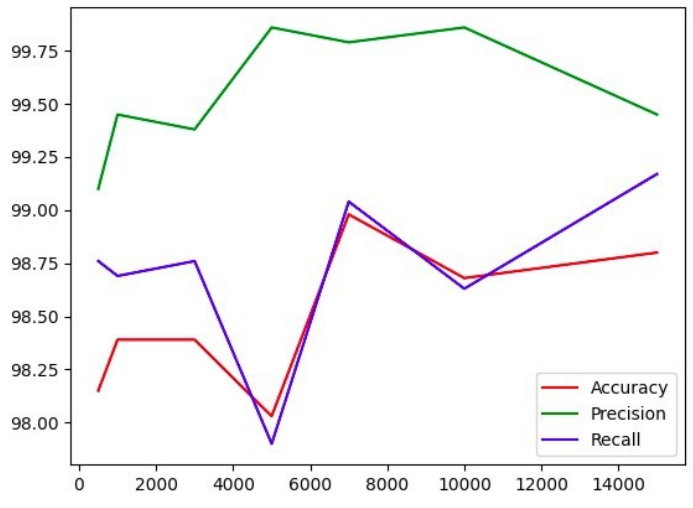
Results are showen in a **2D chart**. Where x = num_entradas, y = (accuracy, recall, precision)

Fig1: The console (only a part of the outputs)



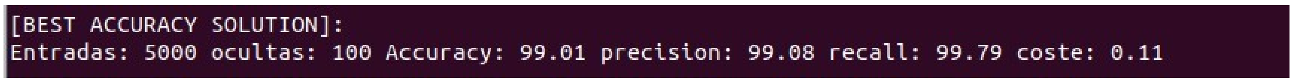
```
num_entradas: 1000 cicle: 0 -> accuracy: 98.56502242152466
num_entradas: 1000 cicle: 1 -> accuracy: 99.01345291479821
num_entradas: 1000 cicle: 2 -> accuracy: 98.83408071748879
num_entradas: 1000 cicle: 3 -> accuracy: 98.56502242152466
num_entradas: 1000 cicle: 4 -> accuracy: 98.83408071748879
num_entradas: 3000 cicle: 0 -> accuracy: 99.01345291479821
num_entradas: 3000 cicle: 1 -> accuracy: 99.10313901345292
num_entradas: 3000 cicle: 2 -> accuracy: 99.01345291479821
num_entradas: 3000 cicle: 3 -> accuracy: 98.7443946188341
num_entradas: 3000 cicle: 4 -> accuracy: 98.83408071748879
num_entradas: 5000 cicle: 0 -> accuracy: 99.01345291479821
```

Fig2: The chart



4) EVALUATE THE RESULTS

Now that we have the suitable input parameters (num_entradas and num_ocultas) we can execute it with the test dataset and measure the results:



```
[BEST ACCURACY SOLUTION]:
Entradas: 5000 ocultas: 100 Accuracy: 99.01 precision: 99.08 recall: 99.79 coste: 0.11
```

5) LIMITATIONS AND RESTRICTIONS

- System based only from one source of example: Can be good for this case but weak for another Spam generator
- Not regularization: Because is a difficult concept to apply at the case: Give more importance to some words to *indicative of spam/ ham*
- *The email detection needs a lot of calculation for a NN so is difficult to form detailed and heavy chart*
- *The dataset doesn't expose the typology of email: It can help to undestand the «white box»*

6) FUTURE WORKS

- Improve all the weakness of the system:
- Add Steamming (example with NLTK) to give robustness to the system: Surpass spam «smart» decision: Capital character, designations, small ortographic error, etc.
- Regularization

7) CONCLUSIONS

The results appears really good: Accuracy and recall are really high (accuracy over 99% and recall over 99,7%). The last value appears really important in our system because the categorized dataset is unbalanced (many more ham that spam email) and because we don't want many "false negative" in the prediction that categorize as spam email which are not – Maybe important ones.

The good results empathizes the optimality of NN models that represents, in addition to SVM, a good option to face spam generation algorithms.

The model gives us configuration flexibility. There are a lots of parameters to change: Input nodes and hidden nodes of the NN; Regularization; Amplification of the indicative words in input to the algorithm; The number of iterations of the optimize.minimize function and others.