Applying Machine Learning Techniques for Categorical Classification of E-Mails

Swapnil Pade, Vanshika Jain, Darius Chua, Xiaoyan Hong ; Team name: Datum

Data Science and Business Analytics FML group, ESSEC Business School | CentraleSupélec

Abstract:

In this report we intend to classify emails giving them specific labels from 0 to 7 into eight classes which are 'Updates', 'Personal', 'Promotions', 'Forums', 'Purchases', 'Travel', 'Spam' and 'social'. This task has received less attention in literature than spam filtering and is complex due to the large number of parameters and lack of balance of class variables. In this work we make an approach of classifying the emails categorically with the K-NN model initially. We then intend to boost our classification outcomes using XGBoost, Neural Networks and CatBoost models to further improve the accuracy of our classification model.

Introduction:

There are various classification techniques which could be implemented for email classification like Naïve Bayes, Support Vector Machine, Decision Trees, Random Forest, KNN, Neural Network, XGBoost and AdaBoost Classifier. But, for our case of multivariate categorical classification we would try classifying emails to different categories using XGBoost and Neural network models, since, after extensive research it seems that these techniques fit best for multiclass classification.

About the Dataset:

For our training data we have a comma separated values file containing 'Sr no.', 'date', 'org',tld','ccs'(quantitative),'bcced'(quantitative), 'mail type'(categorical),'images'(quantitative), 'urls'(quantitative),'salutation'(binary),'designation'(binary),'chars_in_subject'(quantitative), 'chars_in_body'(quantitative) and 'label'(0-7). Hence, for our test data we have all these specified variables (except label) over which we intend to train the classification algorithms to perform with optimal accuracy on our test data.

Data Visualisation:

We first looked at the distribution of numeric data. Except for the categorical data, we can see most numerical data has positive skew distribution.

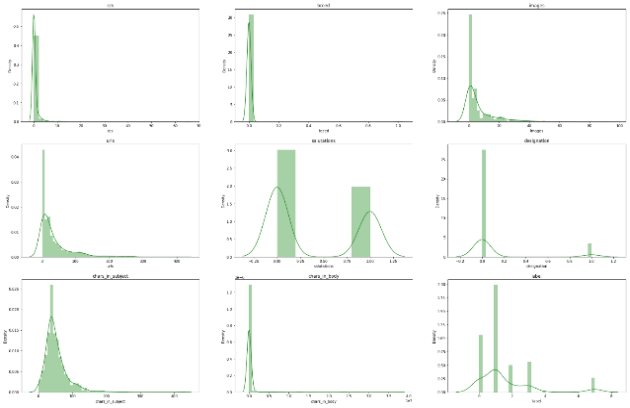

Fig.1 : The distribution of numeric data in the dataset

Besides the distribution we also look at the correlation heap map. We can clearly see the high correlation between images and urls. Also, there is low correlation between bcced and chars_in_body.

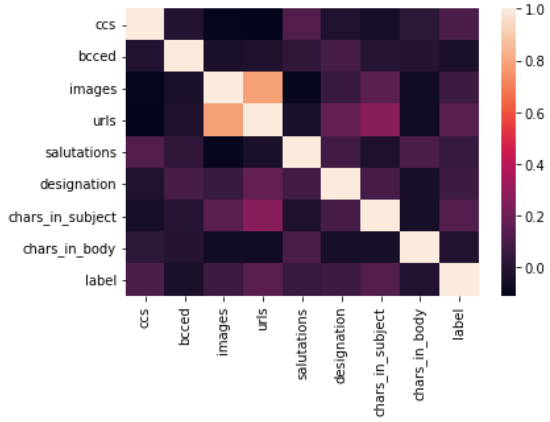

Fig.2: Correlation heatmap for numerical features

Besides the numerical data, for the feature 'date', we need to find the numerical connection between the time and other parameters. For example, the following figure shows the number of spam emails by month. In this case, October is the month for most spam emails being sent.

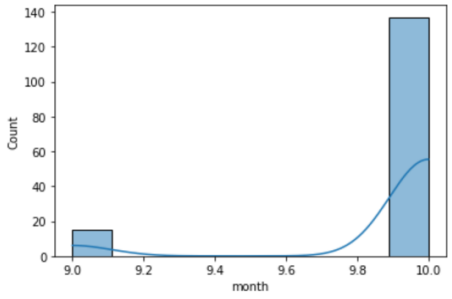

Fig.3: Monthly distribution of spam emails

Feature Engineering and Pre-processing:

- **Handling missing values**
  First off, we had to fill the null values in the entire table. So, numerical columns have been replaced with '0' and characters by 'no entry' or 'na'.

- **Standardization of data**
  Also, converting the strings in the entire dataset to lower case (for uniformity).

- **One hot encoding**
  We have done one hot encoding on the 'mail type' category from our data set.

- **Normalization**
  Normalization makes sure all elements lie within zero and one. It is useful to normalize our data, given that the distribution of data is unknown. Hence, we did min-max normalisation of all the training variables.

- **Derived variables**

1) The 'sender' column has been derived by combining string columns 'org' and 'tld'. derived variable being 'sender' ('org' + 'tld'). Finally, we have cleaned this by replacing the joiners and na values.
2) Log(x+1) transforms of continuous columns deriving 'log ccs', 'log images' , 'log urls', 'log subject chars', ' log body char' from respective columns present in the training data set.
3) New derived interaction feature 'pic url' obtained by ['urls' x 'images'], as we believe that there is a strong correlation between these two variables, i.e, the emails containing the images, might Also contain relevant number of urls and it will be an interesting feature to classify the type of emails into different labels.
4) Furthermore, we derive Date related features from the date column as follows:
   - We take the time stamp to derive these three features:
     a) 'off_hrs': The time between 0:00-8:00
     b) 'work_hrs': The time between 8:00-18:00 on workdays
     c) 'aft_work_hrs': The time after 18:00 on workdays
   - Then, we utilize the day part of the date field to identify whether email was sent on working days or weekends. 'wk_end': weekend
   - Lastly, we use the month from date field to identify different significant months which we see as discussed in the feature engineering section, can be useful for categorizing certain types of emails. These fields as are follows:
     a) 'time3': August, may distinguish forum emails
     b) 'time6': October, may distinguish spam emails
     c) 'time4': November and December, may distinguish promotion emails

- **Target encoding**
  Additionally, in target encoding features are replaced with a blend of posterior probability of the target given categorical value and the prior probability of the target over all the training data. Hence, we have done target encoding on our output variable 'label'.

- **Removing irrelevant fields**
  Finally, we drop off the irrelevant columns such as 'date', 'org' , 'tld' , 'mail type', and 'type' to avoid the problem of overfitting our model.

Implementing XGBoost:

XGBoost is an ensemble Machine Learning algorithm based on decision trees that employs a gradient boosting framework. It allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

This algorithm has different Booster Parameters that can be tuned to prevent overfitting and generate more accurate results. Description of some of these parameters is as given below:

- **n_estimators**
  It is a hyperparameter that determines how many trees/estimators are built within the ensemble model. The more you use, the more accurate it's because of the nature of the Gradient Boosting algorithm. The downside is that the larger n_estimators size it's the longer it takes to train and also can, potentially, overfit to your train data

- **Learning rate**
  Makes the model more robust by shrinking the weights on each step. Generally, the default value of learning rate of 0.1 works but somewhere between 0.05 to 0.3 can work for different problems.

- **Max_depth**
  Used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample but can result in overfitting over train data. The default value for this is 6.

- **Min_child_weight**
  Defines the minimum sum of weights of all observations required in a child. Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the sample selected for a tree. Default value for this is 1.

- **colsample_bytree**
  Denotes the fraction of columns to be randomly samples for each tree. The default value for this is 1.

  Here we have utilised only the tweaks tp the n_estimator parameter. We have taken 3 possibilities with 50, 100 AND 150 number of trees in our gradient XGBoost algorithm.

Implementing Neural Network:

We chose the multi-layer perceptron classifier as it is suitable for multi-class classification. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. MLP utilizes a supervised learning technique called backpropagation for training.

There are three hyperparameters that are important when tuning the model: solver, hidden_layer_sizes and max_iter.

1. **Solver: 'lbfgs', 'sgd', 'adam'**
   The default solver 'adam' works well on relatively large datasets (n>10000) in terms of both training time and validation score. This is exactly our case, and by using Grid Search Cross Validation, we also justified 'adam' as the best choice.

2. **Hidden_layer_sizes**
   The hidden layer size can receive the input of a tuple(i,j), with i being the number of perceptrons in each layer and j being the number of hidden layers. We tried multiple layers first but didn't get satisfied results, the models became slower, and the scores worsened. It is probably because the dataset itself has some noises and the features are largely limited with only the meta data. Therefore, we only focused on single layer tuning afterwards.

3. **max_iter**
   Normally, it could be better to let the model run until it converges if the computing resource allows. While it is not the case here, the dataset, as we mentioned earlier, is not clean and also ambiguous because it is only the metadata of the emails. Thus, it is better to stop at proper iterations to reach a better result avoiding overfitting problems.

To tune the hyperparameters, we use two strategies, as the neural network model has lower execution time in our scenario, one method is to tune these parameters manually.

The other strategy is to use automated searching methods like GridSearchCV and RandomizedSearchCV. Regarding these two methods, GridSearchCV worked better in our case as it traverses all the pairs of hyperparameters we want to try. However, if the mode runs slowly, like XGBoost, RandomizedSearchCV is better as it executes faster than GridSearchCV.

Below we can see the scores for different combinations of number of hidden layers and maximum iterations.

Results:

Below mentioned are the accuracies that we attained on out dataset:

Algorithm Implemented	Test Accuracy
XGBoost	**59.016%**
Neural Network (Multi layer perceptron)	
Hidden layer= 50, iteration=50	**59.404%**
Hidden layer= 40, iteration=90	**58.628%**

Conclusion:

We have attained best accuracy by implementing Neural networks (Multi-layer perceptron) with 50 hidden layers for our test data. This can be entitled to the fact that Neural networks are good to model with nonlinear data with large number of inputs; for example, images. It is reliable in an approach of tasks involving many features.

During the duration of our project we did try a few other techniques which didn't work quite well. Such as, CatBoost, which is similar to XGBoost, except that it executes faster compared to the later one because its decision trees are complete binary trees. Also with CatBoost, we don't have to do the one-hot encoding. There are three hyperparameters we need to tune if using this model: iterations, depth and loss function, as if you want to use the categorical features directly, you also need to indicate the indexes by the hyperparameter 'cat_features'. We also tried to use PCA to reduce the dimensions of the features dataframe and remove the noise perhaps. However, maybe because the dataset is not clean enough or the features are not significant enough, the results weren't improved thus we didn't use PCA in our final version.