Email classification

Shruti Sawant

1 Problem statement

Build an AI model to classify emails into different categories based on the content of the emails. Please download the "A subset of about 1700 labelled email messages". Procedure:

- Pre-process and clean the data
- Build a model to categorize them into the following classes
  - Build a model to categorize them into the following classes
  - Company Business, Strategy, etc. (elaborate in Section 3 [Topics])
  - Purely Personal
  - Personal but in professional context (e.g., it was good working with you)
  - Logistic Arrangements (meeting scheduling, technical support, etc)
  - Employment arrangements (job seeking, hiring, recommendations, etc)
  - Document editing/checking (collaboration)

Following sections explain the approach used to solve given problems step-by-step. Performance analysis is presented using various techniques. Python opensource models, packages or libraries were used for implentation. Please check Jupyter notebooks containing comments and illustrative examples are included along with this report.

2 Data loader

The given dataset contains folders corresponding to various categories of emails. Each folder contains text files corresponding to different emails. For this study, I considered total six folders corresponding to six categories mentioned in the problem statement. First step was to develop data loader to fetch the data for each email from text files. The content of the text file upto line 'Xfilename' are

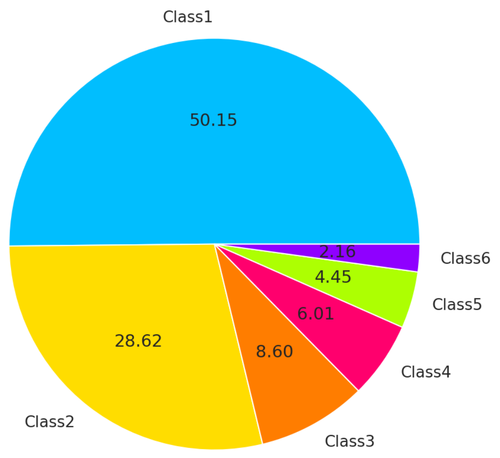

Figure 1: Pie chart showing distribution of samples in six classes

removed while reading the data. Please see file $'DataLoader.pynb'$ for python code with comments and illustrative example for further details. Input to the data loader is the text files from six folders and out of the data loader is six csv files with two columns, namely, 'Content' and 'Label'. Special characters and digits from the extracted content are removed as part of data cleaning in the data loader.

3 Data analysis

This section describes detailed analysis conducted for given problem step-by-step. Codes corresponding to this part can be found in files $'DataAnalysis.ipynb'$ and $'Email_multiclass_sbert_embeddings.ipynb'$ with comments and illustrative examples.

In order to get better understanding of the data, distribution of samples across six categories was observed. Six categories of email form six classes. The given dataset was highly imbalanced as shown in the pie-chart in Figure1. Samples corresponding to class 1 are nearly 50% where as for class 6 is around 2%. This affects the performance of classifier. The challenge of working with imbalanced dataset is that the classifier will ignore the minority class resulting into poor performance on that particular class.

One approach to addressing this issue is to oversample the minority class by duplicating samples in the minority class, although these examples don't add any new information to the model. Another approach is to synthesize new samples from the existing samples. This is known as Synthetic Minority Oversampling Technique (SMOTE) technique. In this study a data augmentation for the

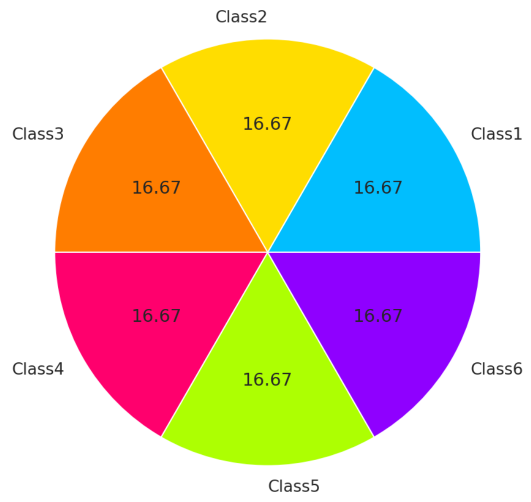

Figure 2: Pie chart showing distribution of samples in six classes after SMOTE sampling

minority class is done using SMOTE.

As part of data loader, text data was cleaned by removing special characters and digits. While working with text data for natural launguage processing (NLP) application, certain pre-processing steps help in improving the performance. Before proceeding to main task of classification, following preprocrossing steps were incorporated :

- Lower case :

  Converting all the content extracted from text file to lowercase helps in the process of preprocessing and in later stages in the NLP application.

- Tokenization:

  Next, breaking a sentence into words is called tokenization, and it produces list of words and punctuation.

- Stemming : This process is the last step in the preprocessing pipeline. Here, we convert our tokens into their base form. Words like ”eating”, ”ate” and ”eaten” get converted to eat.

Word vectorization is a methodology in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers (word embeddings) which used can be used for futher analysis using various machine learning algorithms. For this study, I have explored two vectorizers provided in sklearn library, namely, ’CountVectorizer’ and ’TfidfVectorizer’.

Table 1: Accuracy and F1-scores for various classification models using TF-IDF vectorizer

Classifier model	With imbalanced data		With balanced data	
	Accuracy	F1-score	Accuracy	F1-score
Logistic regressor	0.66	0.35	0.83	0.82
Gaussian naive Bayes	0.51	0.25	0.85	0.84
KNN	0.27	0.16	0.78	0.75
Decision tree	0.49	0.29	0.83	0.83
Random forest	0.62	0.32	0.91	0.91
SVM Linear	0.66	0.44	0.89	0.89
ANN	0.62	0.41	0.92	0.92

CountVectorizer simply counts the number of times a word appears in a document (using a bag-of-words approach), while TF-IDF Vectorizer takes into account not frequency a word that appears in a document but also how important that word is to the whole vocabulary. This is done by penalizing words appearing with higher frquency, reducing the count of these as these words are likely to be less important.

For this study, TF-IDF was observed to be slightly better compared to Countvecorizer. For multi class problem using imbalanced data, TF-IDF proved to provide embeddings that will allow much more accurate machine-learning models to be built.

For classification, using embeddings vector, I evaluated performance of various machine learning algorithms:

- Logistic regression with L-1 regularizer
- Gaussian Naive Bayes
- K-nearest neighbour (KNN)
- Decision tree
- Random forest with 50 estimators
- Support vector machine (SVM) with linear kernel
- Artificial neural network (ANN) with 50 hidden layers

For all the models, hyper-parameters were selected from suitable range of values to get highest performance. As the dataset was smaller in size, 10-fold cross-validation strategy was used. Along with accuracy, F-1 score is also computed as performnace metric. F1 is a great scoring metric for imbalanced data when more attention is needed on minory classes. Performance evaluation is recorded in Table1 using TF-IDF embeddings.

Table 2: Accuracy and F1-scores for various classification models using SBERT embeddings

Classifier model	With imbalanced data		With balanced data	
	Accuracy	F1-score	Accuracy	F1-score
Logistic regressor	0.65	0.31	0.82	0.81
Gaussian naive Bayes	0.58	0.42	0.69	0.69
KNN	0.63	0.37	0.83	0.80
Decision tree	0.50	0.30	0.77	0.77
Random forest	0.64	0.30	0.93	0.93
SVM Linear	0.67	0.44	0.86	0.86
ANN	0.65	0.45	0.92	0.92

For classification using imbalanced data, values of F1-score were much less than values of accuracy for the same models. This is because majority class gets more significance due to more number of samples.

Oversampling of minory class increases total number of samples from 1663 to 5004. The distribution of six class after SMOTE is shown in the pie-chart in Figure2. Now, each class is equally distributed. Comparitive analysis has been included with and without using SMOTE technique to demonstrate effect of class imbalance.

Deep learning models like BERT and RoBERTa has set a new state-of-the-art performance on various tasks in the domain of NLP. However, it requires that both sentences are fed into the network, which causes a massive computational overhead. Recently Sentence-BERT (SBERT), a modification of the pretrained BERT network that reduces the computation while maintaining the accuracy from BERT. SBERT outperforms other state-of-the-art sentence embeddings methods. Using SBERT, word embeddings are computed and the same analysis is conducted for six-class classification problem. Performance evaluation is recorded in Table2.With imbalanced data, values of F1-score and values of accuracy are comparable for classifier models.

4 Inferences and Conclusion

With imbalanced data, SVM gives the best performance among all models. Although performance of ANN is comparable to SVM, it is computationally more expensive to train ANN. KNN suffers dimensionality curse when it comes high dimensional feature vector. Random forest models address overfitting issue occurring in case of decision trees, giving better performance. Naive Bayes has poor performance due to assumption that all features are conditionally independent. Deep learning architectures typically require lot of data, therefore for this problem there is no significant improvement in the performance using SBERT embeddings. However computation cost increased significantly. The highest accuracy of 92% with F-1 score of 92% was observed using ANN.