Automatic YouTube Comment Spam Filtering

Lige Wang (lw2427), Yu-Ping Lin (ypl238), Yu Sun (ys3225), Xiaoxue Ma (xm576)
BetaGo, Center for Data Science, New York University

***Abstract*—In this project we experimented with and evaluated several classification models to deal with social spam problem on YouTube. Among the increasing YouTube users, there are also many malicious users disseminating undesired contents and resulting in many YouTuber disabled comments function. In order to protect users' right of enjoying social Network and communication feature on YouTube, we employed three classification algorithms: Naive Bayes, Logistic Regression from sklearn, and Neural Network from Pytorch. To extract features, we used count vectorizer, TF-IDF, ngram, top k, word2vec, bag-of-word, and GloVe. We tuned the hyper-parameters to improve performance and then applied cross-validation to avoid over-fitting for every model. For future work, we hope to collect larger and more diverse dataset of YouTube comments to enhance its performance and generalizability.**

I. Business Understanding

YouTube, as one of the most influential video platform with social Network features in the world, has attracted thousands of global users, especially after establishing their deployment of monetization system for channel owner uploading original videos. But it also brought negative impacts like uncontrollable malicious users spreading high volume of low-quality information and undesired content, which has been defined as social spam. Some of spam could just have harmless self-promotion purpose, but some of them could threaten Internet users' security, such as malicious links stealing users' payment information.

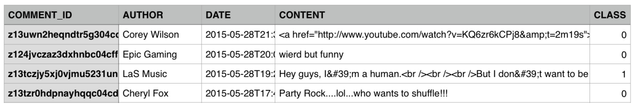
This spam problem starts to get public's attention as more and more influential channel owners disabled comments. YouTube users, therefore, suffer from losing communication and social Network features. Until now, YouTube still lacks of effective and efficient approaches to moderate and control spam comments. Consequently, it is necessary to develop an automatic spam filtering which specializes in dealing with short text.

This project aims to design a classification model to detect and filter out spam comments automatically. It could be integrated into YouTube as a plug-in. Our use scenario is that YouTuber users can choose to install this online tool to assist them in filtering spam comment out, like spam comments could be folded and therefore provide users with a clear Internet environment. Identifying spam in this use scenario constitutes our data mining models. We will elaborate the three models we applied: Bernoulli Naive Bayes, Multinomial Naive Bayes, Logistic Regression, and Neural Network models.

II. Data Understanding

We downloaded the datasets from UCI machine learning repository. It contains five databases of five YouTube videos . They are extracted and collected using the YouTube Data API v3 [1]. It, therefore, is reliable for our business problem as it is not from any simulation but is completely public, real, and

[1]https://developers.google.com/youtube/v3/



COMMENT_ID	AUTHOR	DATE	CONTENT	CLASS
z13uwn2heqndtr5g304cc	Corey Wilson	2015-05-28T21:3	<a href="http://www.youtube.com/watch?v=KQ6zr6kCPj8&amp;t=2m19s">	0
z124jvczaz3dxhnbc04cff	Epic Gaming	2015-05-28T20:0	wierd but funny	0
z13tczjy5xj0vjmu5231un	LaS Music	2015-05-28T19:2	Hey guys, I&#39;m a human.<br /><br /><br />But I don&#39;t want to be	1
z13tzr0hdpnayhqqc04cd	Cheryl Fox	2015-05-28T17:4	Party Rock....lol...who wants to shuffle!!!	0

Fig. 1. Data sample. "CONTENT" is our features and "CLASS" is our target variable.

non-encoded data. Each instance represents a text comment and was labeled manually as spam or non-spam. The labelling process was outsourced to a machine learning tool website [2]. So it is labelled manually by collaborators like grad students. With this trustworthy label, this dataset is sufficient for supervised learning as required. Figure.1 is a sample of our dataset.

In our case, we only need "CONTENT" to get our features and "CLASS" to be our target variable. "COMMENT_ID" can also be used to check whether there is overlap among train/validation/test dataset. One of limitations of our data is not being up to date as the latest data is in 2015. But we chose not to consider this problem chronologically so it will not affect too much. Another limitation is relatively small dataset size. Only 5 videos' comments and about 2000 instances at total. We planned to take "AUTHOR" as a feature about spammer, but it turned out to be of no much effect as no reduplicative author appeared. Besides, in our case, many spam disseminated by real users rather than bots in other social spam problem, so its similarity to legitimate content makes it hard to be detected.

III. Data Preparation

Our original dataset is five separate csv files with each contains comments down below a music video on YouTube. They all have same features, in our case, we just need "content" and "class" for data mining. In function loaddata, We integrated five files into one combined file by implementing *glob* function to searching csv files. By *train_test_split* function from sklearn to splitting dataset randomly, we got our training dataset, which is used to train model and cross-validation, and testing dataset, which is to check the final performance, without overlapping.

We also tried another in-sample/out-of-sample datasets splitting mechanism to prove general applicability. We set arbitrary four databases of five videos as training datasets and the left one as testing dataset. The AUCs for models only fluctuated within 0.0001. In conclusion, our models' performances would not incline to vary with the changing of dataset. For those YouTube videos in same genre, the spam of every single video instance do not possibly have particular patterns, instead, they are mostly identical.

In baseline model (Naive Bayes and Logistic Regression), we used *countvectorizer* and *tfidfvectorizer* to turn raw text to a matrix indicates the frequencies of words. Rows of the matrix are every single comment. And columns of matrix are our vocabulary.

In Neural Network model, we wrote a *getvocabulary* function to extract a vocabulary. In this function, we turned all the raw text into lowercase and deleted all the stop words, punctuation and html code like $< br >$ and etc. Also, we turned all the sentences to tokens, and put them into an index table. Then, we did words embedding, that is, mapping each index to a distinct vector with 50 dimensions. These embedding vectors would be a kind of parameter while we trained the model. We also used pre-train word embeddings, GloVe, as a way to improve our model.

For the definition of the target variable, 1 for spam or 0 for non-spam. Furthermore, we used

[2]http://lasid.sor.ufscar.br/labeling

stemming and lemmatization to improve our features, a process of reducing inflected words to their word stem, base or root form and grouping together the inflected forms of a word so that they all can be analyzed as a single item. Also, we applied n-gram method to extract the features and then chose the top k features by sorting their frequencies. That is, we created n-gram tokens and used the tokens occurs frequently instead of all of them.

IV. Modeling Evaluation

Possible data mining algorithm will be Naive Bayes, which is an old school classifier that will be good for a binary classification. It is very simple and can be realized easily. But since it assumes that features are independent, which is usually not the case, it may be biased. Our dataset is relatively small, so Logistic Regression would be a good choice too. It has been proven over and over to be very robust in small data problems. We also applied Neural Network to analyze this problem. Neural Network model is an advanced model to dealing with problems in natural language. For pros, Neural Network model can capture the relations among features. We just need to do basic featuring engineering and the Neural Network would finish the rest part. For cons, the model is more complicated and consumes more time.

A. Baseline Model, Performance and Evaluation

For text mining scenario, the baseline model applied in this project is Bernoulli Naive Bayes and Multinomial Naive Bayes as they have been considered as the most suitable data mining techniques for classification problem. The countvectorizer gave input for these two models as a matrix filled with term frequency. We used default parameters for both models and cross-validation to confirm it is not overfitted. The test results on hold-out dataset gave us AUCs of 0.972 (bnb) and 0.977 (mnb).

We would use AUC value to be our evaluation function, which can give us a comprehensive summary of how well our classifier ranks. The higher AUC is, the more accurate our result is, which means we can identify spam more efficiently.

B. Logistic Regression

Since Logistic Regression have been proven to be very predictive when it comes to small data set classification problem. We guessed it might be better than Naive Bayes. So we first used default *countvectorizer* to extract features. Also, we applied cross validation to avoid overfitting and make the best use of our samples. It gave us an AUC value of 0.981, which is better than Naive Bayes.

Then we tried to improve its performance by using *tfidfvectorizer*, since it can produce a measure that penalizes word that appear frequently. We also tried stemming, lemmatization, topk and ngram, it turned out that only stemming can improve our performance. At last, *tfidfvectorizer* with stemming gave us an AUC of 0.984, which was slightly better than before. Afterwards, we tuned the hyperparameter of Logistic Regression. We set C value to be e^{-10} to $e^{10}$and did a grid search cross validation on the different values of c. It turned out that c=100 gave us the best AUC, which is 0.989.

Lastly, we drew a graph, figure.2, corresponding to average log loss function value for different parameters in cross validation to test if there is any overfitting. The sweet point, where validation log loss went back up, is around logC=2.5, which is higher than our optimized parameter. So we don't have overfitting in our best Logistic Regression model.

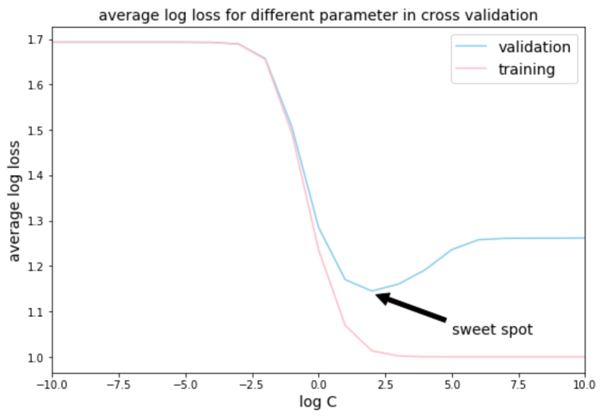


Fig. 2. Loss for train set and validation set

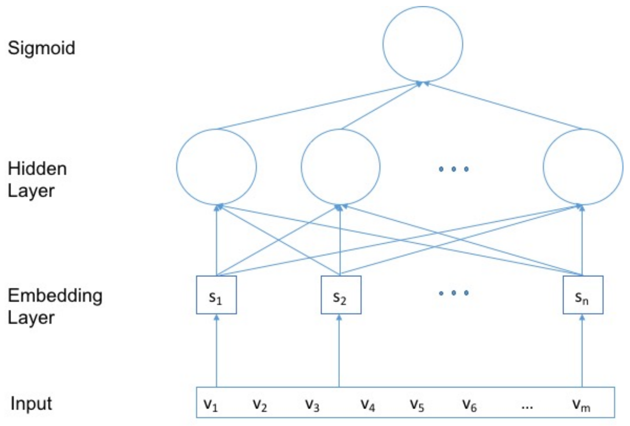


Fig. 3. Model architecture for Neural Network: inputting data into words embeddings, then hidden layer and sigmoid

C. Neural Network

1) Feature Engineering and GloVe: We used n-gram and top k words (occur more frequently than the others) as our core features. And then, let the Neural Network model finish the rest work. We thought not only the frequency of words is an important feature but also the co-occurrence for word to word. For example, the word “check” and “channel” could occur together frequently in a spam comment. The pre-train GloVe word embeddings, built by aggregating word to word co-occurrence statistics, provide us a way to examine our hypothesis.

2) Model Introduction: We used the well-known FastText model [1] implementing by Pytorch, which is a Neural Network Model with a hidden layer. Figure.3 shows the architecture of the model. In this model, we represented the sentence by embedded word vectors and then averaged them to form the hidden variable. This hidden layer is a linear layer which transforms the embedded feature to outputs.Then we used the Binary Cross Entropy as loss function. When choosing optimizer, we tried the ADAM(Adaptive Moment Estimation) and SGD(Stochastic Gradient Descent). However, the accuracy of SGD would volatile too much. Considering the SGD optimizer tackles only few batches of data for each step, it can explain the fluctuating phenomenon. Then, we chose the ADAM as the alternative.

3) Tuning Hyper Parameter of Neural Network model: Initially, we set the learning curve as 0.1, epoch numbers as 25, ngrams as 2, emb_dim as 50, vocabulary size as 5000. We find the validation accuracy is around 80% and fluctuate very much.

a) Learning curve: Since the validation accuracy of every experiments fluctuate very much, we thought it is because learning rate. So we tune it among [0.00001,0.0001,0.001,0.01,0.1]. Learning rate is an important parameter, which controls the step size of Stochastic Gradient Descent. If it’s too high, the accuracy would fluctuate very much. If it’s too low, it would be too slow to learn. From the figure4 above, we could get two points. On the one hand, all learning curves did not grow very rapidly after epoch is 100. In this case, we could set the epoch numbers as 100. On the other hand, when learning rate is 0.001, the green line is almost above all other learning curves. So we can get the optimal learning rate ,which is 0.001. Under this

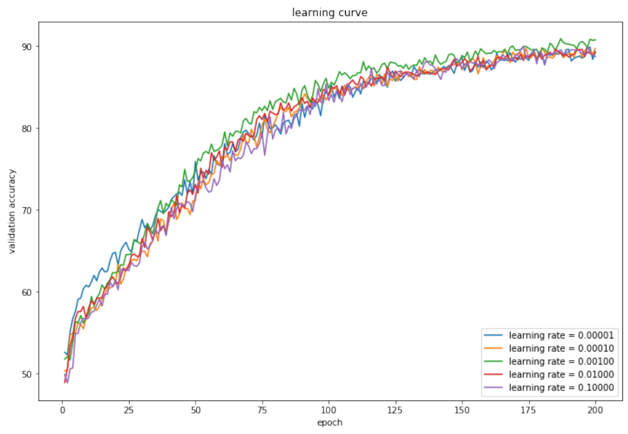


Fig. 4. Learning curve for Neural Network: different learning rate and accuracy grows as epoch grows

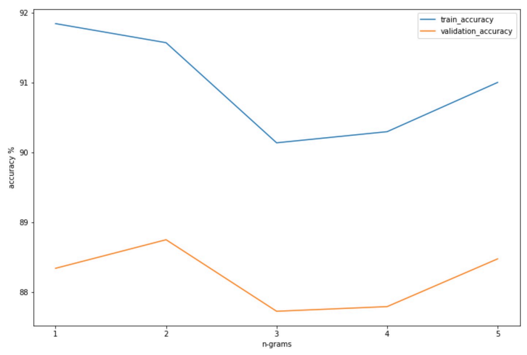


Fig. 5. accuracy in different n-gram, when n is 2, the validation set has the best performance

circumstance, the result is stable at 87%.

b) ngram: N-gram is one of our feature because we thought the order of words might contain information. We tuned the number N of N-gram. As the N grows, the occurring frequency of each sequence words would decrease because they would be unique. For example, “check my” is more frequent than “check my website: http:” Thus, the sequence would be hard to show in our top-k words list. Moreover, in figure 5, we found the accuracy of validation set is the best when N is 2. We used N = 2 as our parameter.

c) embedding dimension and vocabulary size: Embedding dimension is a parameter that controls word2vec embedding, which would reduce the feature dimension helpfully and also contribute to reduce the complexity of the model. According to the figure.6 above, we could see that the accuracy is increasing as the embedding dimension grows. However, after dimension of 50, the growing speed slowed down. Consider the complexity and accuracy, we chose 50 as the optimal dimension. And also, the growing rate of train accuracy and validation accuracy were almost the same, so we can identify the model is not overfitted when embedding dimension is 50.

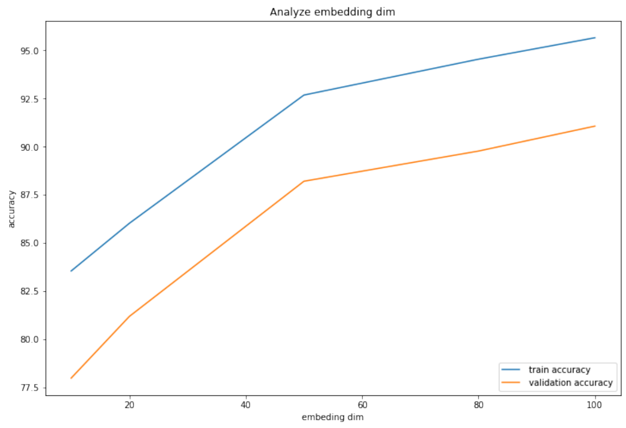


Fig. 6. accuracy with different embedding size

Vocabulary size is the parameter to control how many features we actually have. For example, if we choose vocabulary size 1,000, then we pick the top 1,000 words which is most occurs. When the size is too large, it may include most tokens of dataset, which is not meaningful as only some of them are actually predictive feature. But when it is too small, there may be not enough information for model to learn. After tuning this parameter, we get

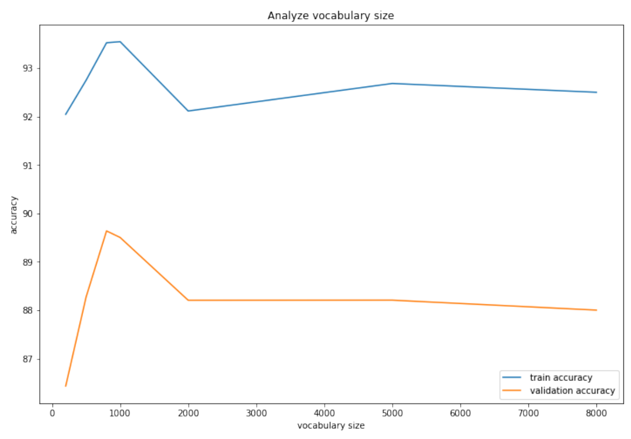


Fig. 7. accuracy with different vocaburary sizes

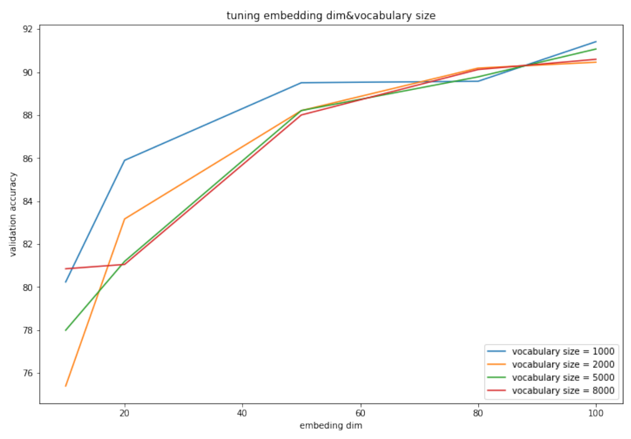


Fig. 8. accuracy with different vocabulary size and embedding size. Although the accuracy can grows up as the dimension and vocabulary size grows, the consuming time also grows up dramatically. However the accuracy grows a little bit.

the figure 7. We set the peak,1000, as the optimal, which improved the validation accuracy from 88.3% to 89.8%.

To get the optimal set of parameters, we need to tune them together. As we can see from figure.8, the vocabulary size of 1000 is the best choice. In terms of embedding dimension, the candidates are the point 50 and 100. But it's not worth to consume two times efforts to gain more 2% accuracy. As a result, the optimal set is vocabulary size of 1000 with embedding dimension of 50.

After tuning the parameter, we got the optimal set = [learning_rate = 0.001, emb_dim = 50, num_epoch = 100, voc_size = 1000, ngrams = 2]. Then we used all the training data to train model and tested it in the holdout dataset and got the accuracy of 93.46%.

d) Overfitting Detection: Our dataset is so limited, so using Neural Network model is easy to be overfitted. To avoid this, we tried cross validation. We splitted the training data to 4 folds and trained the model by three of them and then validated by the left one. To implement this, we need to refresh the model and process the data for each fold. To test whether we are overfitted, we got the following test. As we see figure.9, we could find that the validation accuracy of 4 folds are similar, which verified that the model is not overfitted.

4) Model Improvement:

a) GloVe: We tried to improve our model by using pre-train GloVe. However, the result is not better. The accuracy of validation (accuracy: 84.08%) is]lower than our baseline Neural Network model (accuracy: 93.46%). See the figure.10 describing AUC with accuracy of 84.08%. We thought the main reason is that YouTube comment is generally short and focus on some certain buzzwords. In contrast, the GloVe is trained for large corpus. Thus, the co-occurrence matrix for words based on large corpus is hard to present few particular words.

b) Improve the model in hidden layer: ReLu is a function converting negative values to zero. It's a non-linear transformation function and could help the model to capture non-linear relations. For Neural Network, over-fitting is a big issue. Hence, we dropout some neural receptors in our hidden layer. See the figure.11 with accuracy 81.68%.Unfortunately, the performance did not outperform our

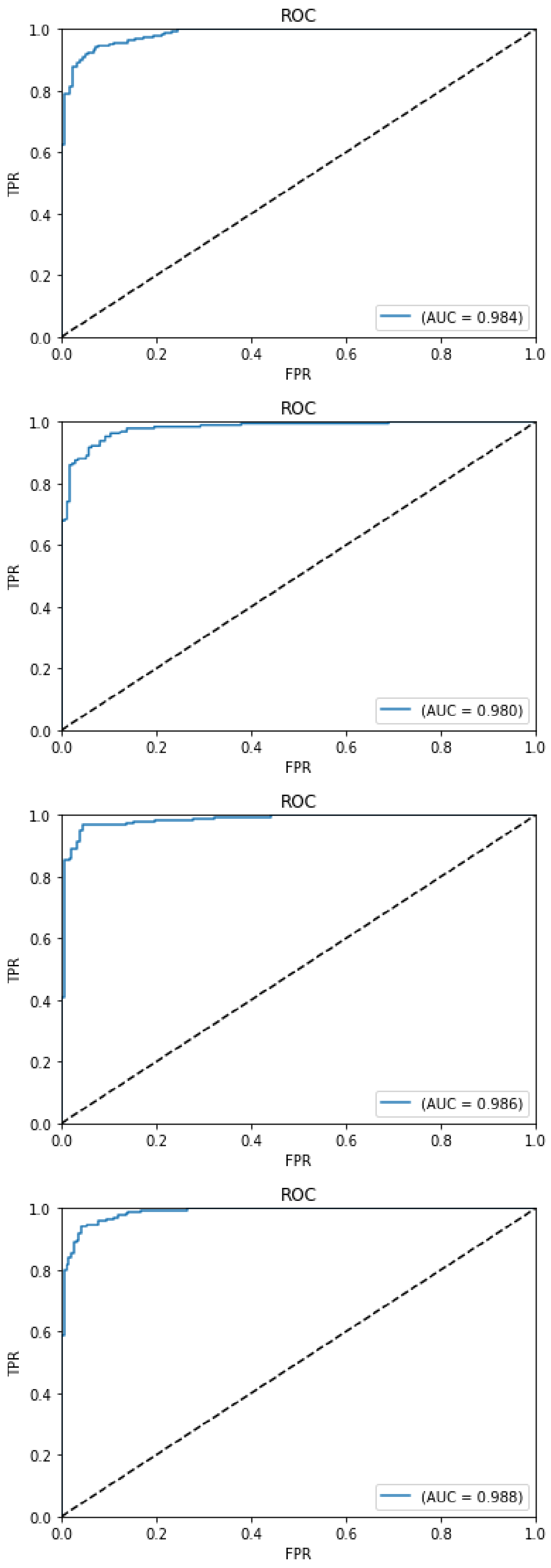


Fig. 9. AUC of cross validation for 4 folds. The validation accuracy of 4 folds are similar, which verify that the model is not overfitting.

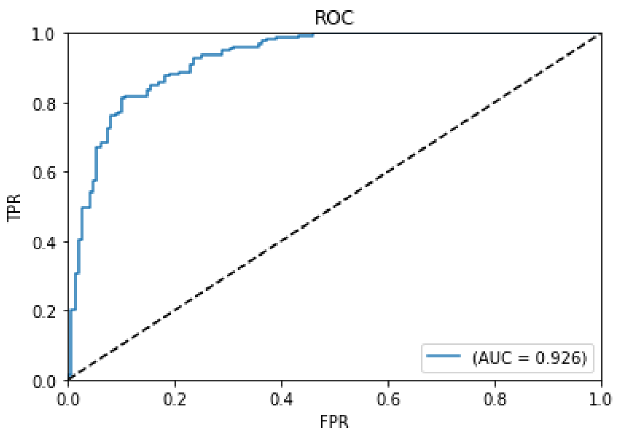


Fig. 10. pretrain glove (accuracy: 84.08%) is not better than our base Neural Network (accuracy: 93.46%)

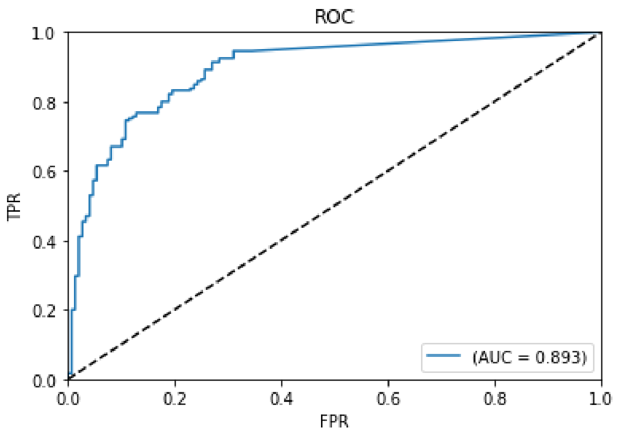


Fig. 11. pretrain model with new layer and relu the performance 92.84%

Neural Network baseline model(accuracy: 93.46%). The main reason is that we haven't been overfitted so dropping receptors will deteriorate the performance. Thus, the performance is even worse than the model only with pre-train GloVe.

c) Results: After improving models and ensure not to be overfitted, we verified that the best accuracy of Neural Network Model is 93.46% and used the ROC/AUC to evaluate it as following figure.12. The model is with our tuned hyper-parameters,

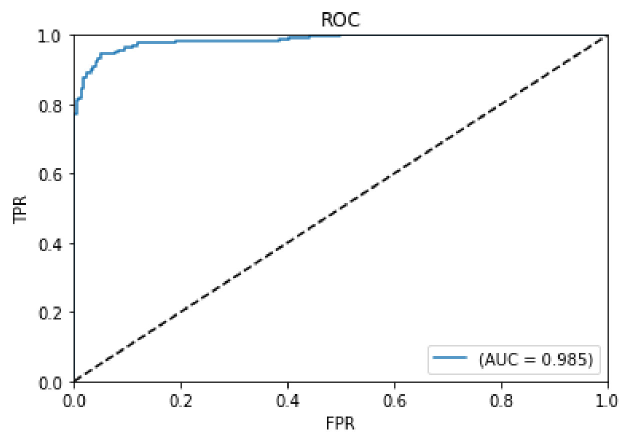
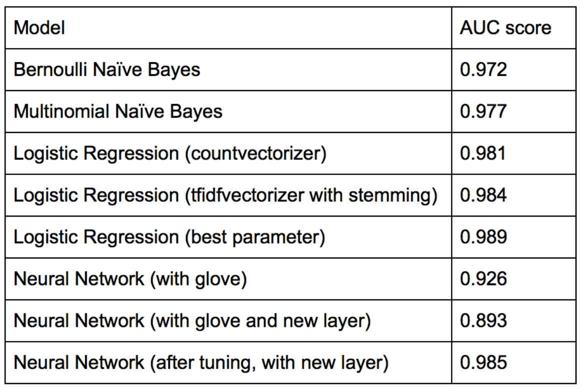


Fig. 12. the model we select for the Neural Network model



Model	AUC score
Bernoulli Naïve Bayes	0.972
Multinomial Naïve Bayes	0.977
Logistic Regression (countvectorizer)	0.981
Logistic Regression (tfidfvectorizer with stemming)	0.984
Logistic Regression (best parameter)	0.989
Neural Network (with glove)	0.926
Neural Network (with glove and new layer)	0.893
Neural Network (after tuning, with new layer)	0.985

Fig. 13. comparison AUC for all the model

vocabulary size 1,000, learning rate 0.001, bigram, word embedding dimension 50 and without pre-train GloVe and new layer.

D. Model Selection

We created the figure.13. After comparison of all the AUC scores, we found that Logistic Regression after tuning is the best.

E. Detect Problem

Since our AUC is extremely high, we reasonably suspected whether any features could leak our label. After studying researches about leakage, we determined that our case doesn't match with any normal type of it. The features are not designed by ourselves, instead, extracted from text. Therefore, it is unlikely to involve label information. On the other hand, our features are independent with chronological order, hence it would not leak anything from future. Trying to prove our standpoint, we yielded a vector to store all the correlation between features and label. We found that no correlation equals to 1 appears, which means no such a feature could leak label.

F. Deployment, Extension

Now, moving on to the business deployment scenarios. We can develop an online tool or even browser plug-in based on our model. It provides a solution for YouTube to moderate spam comments. When employing our model, spam can be detected instantly and filtered out in real time.

We can also extend it to manage spam in other short text problems like Instant Messaging apps or mobile messages. Or regard it as a pre-processing tool for semantic analysis, for example, it can assist to eliminate all the undesired content on IMBD at first when we wish to analyze the real feedback of audience. There is no ethical considerations or related issues when companies apply it as all the data is open and public.

G. Conclusion and Future Work

Finally, this paragraph will give the evaluation of our models' performances. After studying previous researches, we learnt that Naive Bayes models have been proved to be a suitable but stereotyped classifier for this problem. But in the revolution of spam or ham problem, it has became less frequently used as its Naive assumption can hardly hold in real life. We, therefore, set two Naive Bayes models as our baseline model and tried to figure out if there are any other more complicated models could be the game changer in this revolution. Seen from AUC table shown above, Logistic Regression model and Neural Network model both achieved very satisfying and impressive results in our project even though there is no huge improvement comparing with the baseline model. Restricted by the shortage and simplicity of our dataset, simple model could outperform sophisticated model. Considering from the aspect of business investment and return, Neural Network is far more sophisticated in coding and more time-consuming than Logistic Regression. It could even take over ten times time to run. So we conservatively believe Logistic Regression is more ideal model for our problem.

As for future work, we should try to collect more updated data from YouTube and cover different genres of videos. Subject to the shortage of data, it is unnecessary to employ text normalization techniques such as processing abbreviations, idioms, smiley, emoji, slangs, etc. as they have no chance to appear in top k. But in fact, they also carry decisive information to identify spam. We also intend to ensemble different classifiers to check if it outperforms single classification model.

V. Appendix

References

[1] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T., 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning, 2014. GloVe: Global Vectors for Word Representation.

[3] “1.9. Naive Bayes - scikit-learn 0.19.1 documentation”, Scikit-learn.org, 2017. *[Online]. Available:[Accessed: 08- Dec- 2017].*

[4] “Leakage — Kaggle”, Kaggle.com, 2017.*[Online] [Accessed: 08-Dec- 2017].*

[5] “sklearn.naive$_{B}ayes.BernoulliNB$” $scikit - learn$0.19.1$documentation, Scikit - learn.org$, 2017. *Online] [Accessed: 08- Dec- 2017].*

[6] Alberto, T.C., Lochter J.V., Almeida, T.A. TubeSpam: Comment Spam Filtering on YouTube. *Proceedings of the 14th IEEE International Conference on Machine Learning and Applications (ICMLA’15), 1-6, Miami, FL, USA, December, 2015.*

[7] T.A. ALMEIDA, T.P. SILVA, I. SANTOS and J.M. GOMEZ HIDALGO. Text Normalization and Semantic Indexing to Enhance Instant Messaging and SMS Spam Filtering. *Knowledge-Based Systems, Elsevier, 108(2016), 25-32, 2016.*

[8] F. Provost and T. Fawcett, Data science for business. Sebastopol, *CA: O’Reilly, 2013.*

VI. Contribution

A. Yu Sun

- Do research on word2vec and Neural Network model
- Process the data(tokenize, batchify, padding) to prepare for Neural Network model
- Read paper and Implement the Neural Network by Pytorch
- Tuning parameter of learning rate, epoch, emb_dim, vocabulary_size
- Implement the cross validation and evaluation methods of Neural Network
- Write the report of Neural Network model introduction, tuning parameter, cross validation and draw model architecture plot.

B. Yu-Ping Lin

- Do research on word2vec and Neural Network model
- Process the data(tokenize, batchify, padding) to prepare for Neural Network model
- Read paper and Implement the Neural Network by Pytorch
- Tuning parameter of ngrams
- Try Improve methods, like pre-train Glove, new relu layer and drop out
- Write the report of tuning parameters, improve Neural Network model
- Format the report in Latex.

C. Xiaoxue Ma

- Clean the data
- Write the *getvocabulary* function to turn raw text into an index table
- Build Logistic Regression model and tune parameters
- Implement cross validation and detect if there is overfitting or leakage.
- Write the report of data preparation, modeling evaluation, model selection and detect problem.

D. Lige Wang

- Wrote *loadindata* function to integrate databases
- Cleaned data
- Wrote codes and report of baseline model and its cross-validation
- Studied related papers and then identified leakage
- Wrote below sections of report: Business Understanding, Data Understanding, Data Preparation, Modeling Evaluation (Baseline model), Data Leakage, Deployment and Extension, Conclusion and Future Work

VII. Code Fragment

```python
class NeuralNetwork(nn.Module):
    def __init__(self,vocab_size,emb_dim):
        super(NeuralNetwork, self).__init__()
        self.vocab_size = vocab_size
        self.emb_dim = emb_dim
        self.embedding = nn.Embedding(vocab_size+1,emb_dim)
        self.dummy_layer = nn.Linear(emb_dim,1)

    def forward(self,data,length_list):
        emb_data = self.embedding(data)
        avg_emb = torch.sum(emb_data,1)/length_list.float().view(-1,1)
        layer1_data = self.dummy_layer(avg_emb)
        outputs = nn.functional.sigmoid(layer1_data)
        return outputs
```

```python
def trainmodel(model,optimizer,traindata,validationdata):
    trainacc = []
    validacc = []
    for epoch in range(num_epochs):
        for i,datum in enumerate(traindata):
            databatch = Variable(datum[0])
            lengthbatch = Variable(datum[1].resize_(len(datum[1]),1))
            labelbatch = Variable(datum[2].resize_(len(datum[2]),1))
            optimizer.zero_grad()
            outputs = model(databatch,lengthbatch)
            loss = criterion(outputs,labelbatch.float())
            loss.backward()
            optimizer.step()
            if (i % batch_size==0):
                train_acc = testmodel(model,traindata,False)
                if validationdata:
                    validate_acc = testmodel(model,validationdata,False)
                    print('Epoch: [{0}/{1}], Step: [{2}/{3}], Loss: {4}, Train Acc: {5}, Validation Acc:{6}'.format(
                        epoch+1, num_epochs, i+1, len(traindataset)//batch_size, loss.data[0],
                        train_acc, validate_acc))

                else:
                    print('Epoch: [{0}/{1}], Step: [{2}/{3}], Loss: {4}, Train Acc: {5}'.format(
                        epoch+1, num_epochs, i+1, len(traindataset)//batch_size, loss.data[0],
                        train_acc))

        if validationdata:
            trainacc.append(train_acc)
            validacc.append(validate_acc)
    return trainacc, validacc
```

```python
def crossvalidation(trainset, trainlabel, k,AUC = True):
    print(num_epochs)
    kf = KFold(n_splits=k)
    kth_fold = 1
    valid_AUC = []
    valid_acc = []
    train_acc = []
    for train_index, test_index in kf.split(trainset):
        model = NeuralNetwork(topk ,emb_dim)
        optimizer = optim.Adam(model.parameters(),lr = learning_rate)
        trainset = np.array(trainset)
        trainlabel = np.array(trainlabel)
        X_train, X_validation = trainset[train_index], trainset[test_index]
        y_train, y_validation = trainlabel[train_index], trainlabel[test_index]
        traindataset = CommentDataset(X_train.tolist(),y_train.tolist())
        validationset = CommentDataset(X_validation.tolist(),y_validation.tolist())
        train_data = torch.utils.data.DataLoader(dataset = traindataset,
                                         batch_size= batch_size,
                                         shuffle= True,
                                         collate_fn= AddPadding)
        validation_data = torch.utils.data.DataLoader(dataset = validationset,
                                         batch_size= batch_size,
                                         shuffle= True,
                                         collate_fn= AddPadding)
        print("{0}th fold".format(kth_fold))
        trainacc, validacc = trainmodel(model,optimizer,train_data,validation_data)
        kth_fold = kth_fold + 1
        if AUC:
            valid_AUC.append(testmodel(model,validation_data,True))
            print("the average Valid AUC is ",np.mean(valid_AUC))
        else:
            valid_acc.append(validacc)
            train_acc.append(trainacc)
            #print("the average Valid Accuracy is ",np.mean(valid_acc))
    return np.mean(train_acc,0), np.mean(valid_acc,0)
```