MULTI-LABEL CLASSIFICATION BASED ON TRIP ADVISOR HOTEL REVIEWS

CSCI 544 Course Project

Hang Dong, Jin Qi, Mengyue Liang, Yixin Yang

1. Data collection method

We tried to generate a web crawler in order to get the reviews from Trip-advisor.

The main way is to get HTML format web pages, crawl and resolve the target text data at relevant sites by using web crawlers. The code sends a certain amount of data directly to the server, and then retrieves the data returned by the server so that we can extract the desired information. If we have to be logged in to get some data, we impersonate the login as a normal user to get a response data from the server.

However, it seems that there is anti-crawler in their website. As a result, we managed to collect the dataset in data. World website (https://data.world/datafiniti/hotel-reviews)crawled by other people in 2017, which includes over 10,000 reviews of 500+ hotels, and used them as our data. Among them, there are repeated reviews. So, instead of selecting part of the data, we separate the whole data evenly among four of us and tried to label them as much as we could. During this time, the duplicate data or the data not applicable (for example, reviews contains other languages, emojis or other unknown symbols) was deleted from our dataset.

In the end, we have 5031 labeled, which is also the size of our data set.

2. Data labeling method

The collected data will be labeled manually with given labels, representing the aspects been discussed in that review. Labels and rules for labeling are explained below.

The labels we proposed:

1) service:

If the staff's manner was mentioned in a particular review, for example, "The receptionist is really nice and helpful", then the label for this review will add "service". Or, this label can also be inferred from what customer implied. Here is what we noticed during labeling: "Lucy is a really nice person and helped us carried the luggage." Clearly, this customer also talked about the service of that hotel.

2) location

Location is more like the geographic position of that hotel. For example, the hotel is located right by the sea, or "just ten minutes' walk from some scenery spot",

3) environment

This label mainly focusing on how the hotel makes customer feel. Whether it's clean or dirty? Fashion or old? Smells mildew? Safe or dangerous?

4) Facility

The label contains much information. For example, the bed is comfortable or not, the WIFI or AC worked good or not, the room is big or small, also the gym or swimming pool is also included in this part.

5) Food

Many reviews talked about this, and it is pretty simple to label them. Most travelers talked about how the breakfast taste and what kinds of food included, as long as food was provided by hotel. More than that, some also talked about the bar within the hotel, we also labeled them as “food”.

According to these five classes, we labeled data manually. Labeling them in csv file, separate different labels using this mark “[‘’]”, we wrote no label for those reviews which doesn’t match with any of our labels, also the repeated reviews. And finally transform the csv table into json file all at once.

About inter annotator agreement, we first randomly select 200 reviews from the data set and label those reviews at the same time using the standard mentioned above. Then we checked whether there is differences or conflictions between our label result. During this process, we can make sure that we have a unit standard on labeling and each of us would have a fully understanding about what each label represents. In this way, we make sure that all of our labeling data followed the exactly same rules and each annotator and the result given by them could be fully trusted.

As a result of inner-agreement test, there are 183 out of 200 reviews that are labeled exactly the same by all 4 annotators, which is a 91.5% agreement between annotators. Therefore, it is reliable to do separate annotation, and the results are compatible. Besides, when annotating the data we are going to use for training and testing, we set 500 reviews to be annotated by more than one annotator. Then after finish labeling, we discard those data that are labeled differently by different annotators, to make our data reliable and correct.

Last but not least, we double check each other’s labeling data, which helps to correct the spelling mistake and improved the performance of our network.

3. Classification Approaches

In this project, we used 3 different algorithms and came up with 4 classifiers for our TripAdvisor review classification task, including support vector machine (SVM), convolutional neural network (CNN), and long short-term memory (LSTM). We will discuss the theory, algorithms and our coded system configuration, and the final results of these classifiers respectively in the following parts.

3.1 Linear Support Vector Machine (baseline approach)

3.1.1 SVM Theory

Support vector machine is a kind of supervised learning models, which trains on labeled data and comes up with a hyperplane or a set of hyperplanes that separates data into different classes. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.[1] In our project, we use a one-verses-rest linear SVM classifier as our baseline approach. Below we will take a 2D, 2 classes problem as an example, to show the basic SVM model.

[1] Trevor Hastie, Robert Tibshirani, Jerome Friedman - The elements of Statistical Learning Pg. 134

Suppose we have a set of labeled binary-class training data and plot them on 2D coordinate system as shown on the right, and we want to construct a line to separate them, so that when new data comes, this line can still be used to predict label for the new data.[2] The format of training data and labels are as follows:

$$data: (\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), \dots, (\overrightarrow{x_n}, y_n)$$

$$\overrightarrow{x_i} = (x_{i1}, x_{i2}), \quad y = 1 \; or -1$$

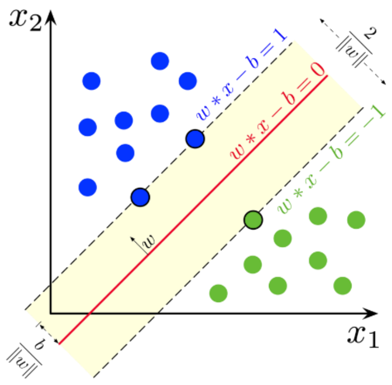


Fig.1 maximum-margin SVM

For 2D linear case, a line is the separator for two classes, and we want data points with $y = 1$ and $y = -1$ to be in different sides of the line. There are many choices for a line to be successfully separate these two classes, the general form is:

$$y = \vec{w}^T \cdot \vec{x} + b$$

where $\vec{w}$ is the weight vector of the line.

In SVM, we are not only finding the hyperplane to separate two classes, but also want the distance from the hyperplane to the nearest data point to be as large as possible, so that it can be “safer” to new data points.

If the two classes are linearly separable, then

$$\vec{w}^T \cdot \vec{x} + b \geq 1, \; if \; y = 1$$

$$\vec{w}^T \cdot \vec{x} + b \leq -1, \; if \; y = -1$$

so the distance between lines $y = 1$ and $y = -1$ is

$$d = \frac{2}{\|\vec{w}\|}$$

Then we want to minimize d subject to the constraint that $y_i(\vec{w}^T \cdot \vec{x}_i) \geq 0$.

On the other hand, if the two classes are not linearly separable, we cannot have a line that clearly divide two classes, so we introduce slack variable ξ_i such that:

$$\xi_i = 0, \quad if \; \vec{x}_i \; is \; on \; the \; correct \; side \; of \; margin$$
$$\xi_i = normalized \; distance \; from \; \vec{x}_i \; to \; margin, \quad if \; \vec{x}_i \; is \; on \; incorrect \; side \; of \; margin$$
$$\xi_i = 1, \quad if \; \vec{x}_i \; is \; on \; the \; decision \; boundary$$

Therefore, now we want to minimize the function

$$\lambda \|\vec{w}\|^2 + C \sum_{i=1}^{n} \xi_i$$

in which C is the penalty term to the data points that are in wrong side of margin (whose slack variable ξ_i is greater than 0), and the parameter λ determines the tradeoff between increasing the margin-size

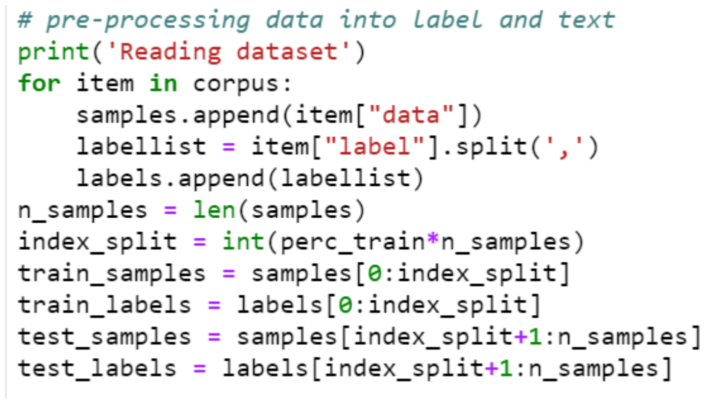
[2] Picture from https://en.wikipedia.org/wiki/Support_vector_machine#Definition

and ensuring that the $\vec{x}_i$ lie on the correct side of the margin[3]. Since C penalizes misclassified data, the larger C is, the more penalty will be added to misclassified data, the narrower the margin will be.

In our task, we are dealing with a multilabel classification, so we use one vs. rest to combine several SVM classifiers to form the final one. The strategy consists in fitting one classifier per class, and for each classifier, the class is fitted against all the other classes.[4]

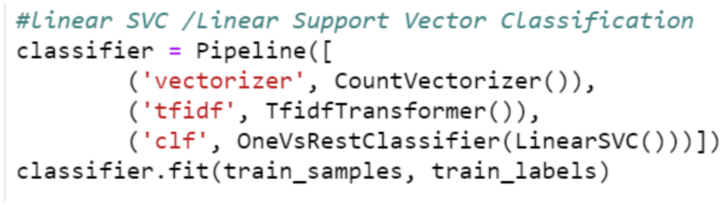
3.1.2 Algorithm Implementation

Before we start training, the first thing we should do is to preprocess the text data.



```
# pre-processing data into label and text
print('Reading dataset')
for item in corpus:
    samples.append(item["data"])
    labellist = item["label"].split(',')
    labels.append(labellist)
n_samples = len(samples)
index_split = int(perc_train*n_samples)
train_samples = samples[0:index_split]
train_labels = labels[0:index_split]
test_samples = samples[index_split+1:n_samples]
test_labels = labels[index_split+1:n_samples]
```

We read in reviews and their labels one by one and separate each review and label by "," respectively, to make it a list of words, then append lists of review words to big lists, containing all the reviews and corresponding labels. Then we split the whole dataset into training and test set by a ratio of 4:1. The structure of our linear support vector classifier is shown below. It's done in a pipeline of transformations followed by an estimator.



```
#Linear SVC /Linear Support Vector Classification
classifier = Pipeline([
        ('vectorizer', CountVectorizer()),
        ('tfidf', TfidfTransformer()),
        ('clf', OneVsRestClassifier(LinearSVC()))])
classifier.fit(train_samples, train_labels)
```

First, we convert the data from text to a matrix of token counts using `sklearn.feature_extraction.text.CountVectorizer`, which takes in text data and output a matrix that contains the counts of each token. After this vectorization, we can transform our text data into a numerical matrix, so that it can be fed into SVM classifier.

Then, `TfidfTransformer()` transforms the previous generated count matrix, into a normalized tf or tf-idf representation. Tf stands for term-frequency, and tf-idf means term frequency time inversed document frequency. The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and

[3] https://en.wikipedia.org/wiki/Support_vector_machine#Definition
[4] https://scikit-learn.org/stable/modules/multiclass.html

that are hence empirically less informative than features that occur in a small fraction of the training corpus.[5]

Finally comes our one-verses-rest linear SVM classifier. We use all the default values for our parameter, which is shown in Table 1.

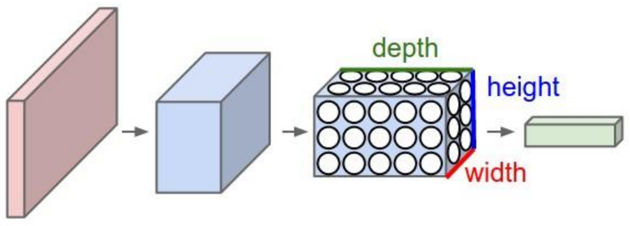
Table 1 Parameter Settings for One-V. S-Rest Linear SVM Classifier

C	1	Penalty for misclassified data points
penalty	L2	L2 regularization
loss	squared_hinge	
multi_class	ovr	
tol	1e-4	Tolerance for stopping criteria
fit_intercept	True	Whether to calculate interception
class_weight	balanced	To balance the number of data points in each class
max_iter	1000	Maximum number of iterations

3.2 Convolutional Neural Networks (CNN)

3.2.1 Architecture of Convolutional Neural Networks

As shown below, CNN arranges it neurons in three dimensions. What makes it different from normal feedforward neural network is the convolutional layer. Each neuron in the convolutional layer is not fully connected with every neuron in the input layer, but is looking at a subset of it, which is call receptive field. In convolutional layer, each filter performs dot product with the related field in the input layer and slides the filter then product with the next receptive field. This activity is similar to signal convolution, so it got the name convolutional layer.



[6]

Fig.2 structure of convolutional neural network

CNNs is just a set of convolutional modules, each containing one convolutional layer and other nonlinear activation layer, pooling layer, and some other functional layers, followed with different output layers which depends on the task CNN is dealing with. The other common used layers except for convolutional layer includes ReLU (or tanh) layer which acts as an activation function, max (or average) pooling layer which performs down sampling to the data, fully connected layer which is used as output layer for classification tasks, and regression layer which is the output layer of regression problems.

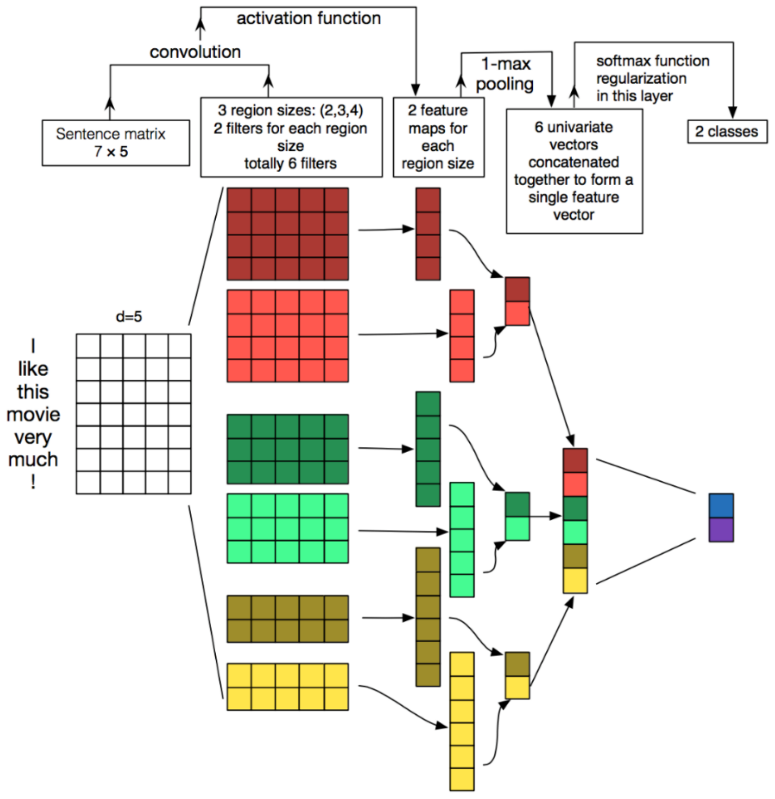
[5] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html
[6] Picture from http://cs231n.github.io/convolutional-networks/

During training, a CNN can automatically learn weights for the filter, then extract useful features by sliding filters and product with input layer. CNN is powerful because it gives you invariance to translation, rotation and scaling, and each filter composes a local patch of lower-level features into higher-level representation.[7]

3.2.2 CNN for NLP Tasks

CNN is widely used in computer vision field, but has been proved to be effective to NLP tasks as well.

For computer vision tasks, the input to the CNN is an image with pixels been represented by numeric values. So in NLP tasks, we can use word embedding to transform each word to a row vector, and the whole text as an n-by-m matrix, where n is the number of words in the text, and m is the dimensions of embedding. In this way, we can have our text "pretend" to be an image, and feed it into CNN for feature extraction and classification. The difference between computer vision and NLP task in this network is that, in vision, the filter is slide over receptive fields, but in NLP, the width of the filter is the embedding dimension, which is a whole row of the matrix, and the height (columns, how many words are in the filter) can vary, but 2-5 are the most commonly used ones. Below is an illustration of how CNN works in NLP tasks.


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

[8]

Fig.3 illustration of how CNN works on NLP task

[7] http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
[8] Picture from http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

3.2.3 Code Implementation

Since we are using CNN in this case, so we have to first preprocess our text data to be vector-represented.

```
for item in corpus:
    texts.append(item["data"])
    labellist = item["label"].split(',')
    temp = []
    for key in labels_index.keys() :
        if key in labellist :
            temp.append(1)
        else :
            temp.append(0)

    labels.append(temp)
```

The first part is almost the same as previous SVM preprocessing, where we split each review data into words and store them into a list. For labels, we store each label corresponding to a review data as a 1×5 vector, which means we have 5 possible labels in total, and for each review, if it has that label, the corresponding place in label vector will be 1, otherwise it is 0.

```
tokenizer = Tokenizer(num_words=None,lower = True)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
data = pad_sequences(sequences)
labels = np.array(labels)

x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=9000 )
```

Then we use Keras built-in function `Tokenizer()` to convert each review data into a vector, with each word been convert to an integer based on its counting rank over all word in our dataset. After done preprocessing, we split our data into training and test sets with ratio 4:1.

```
model.add(Embedding(input_dim=num_words + 1, output_dim=100, input_length=MAX_SEQUENCE_LENGTH,
                    embeddings_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.2))
model.add(Conv1D(128, 5, activation='tanh'))
model.add(GlobalMaxPool1D())
model.add(Dense(len(labels_index)))
model.add(Activation('sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['categorical_accuracy'])
```

The structure of our model is shown above. In this model we only build 1 convolutional module. The layers and constructions are shown in Table 2.

Table 2 Layers and Constructions of Simple CNN

Embedding layer	**Input dimension**	num_words + 1
	Embedding dimension	100
	L2 regularization	0.01
Dropout layer	**Dropout coefficient**	0.2
Convolutional layer	**Activation function**	tanh
Pooling layer	max pooling	

Dense layer	**Output dimension**	len(labels_index)
Activation layer	**Activation function**	sigmoid
Optimizer	Adam	
Loss function	binary_crossentropy	
Metrics	categorical_accuracy	

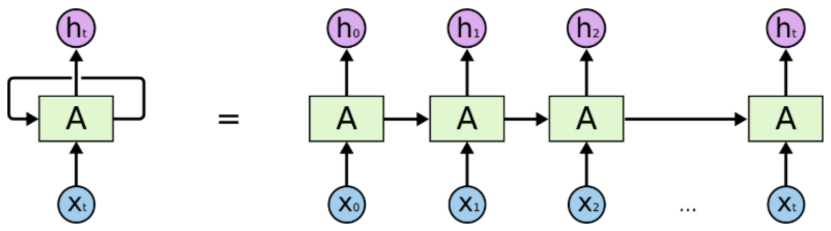
We also tried a 3 convolutional-module CNN as follows.

```
model.add(Embedding(input_dim=num_words + 1, output_dim=EMBEDDING_DIM, input_length=MAX_SEQUENCE_LENGTH))
model.add(Conv1D(128, 5, activation='tanh'))
model.add(MaxPooling1D(5))
model.add(Conv1D(128, 5, activation='tanh'))
model.add(MaxPooling1D(5))
model.add(Conv1D(128, 5, activation='tanh'))
model.add(MaxPooling1D(15))
model.add(Flatten())
model.add(Dense(128, activation='tanh'))
model.add(Dense(len(labels_index), activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['categorical_accuracy'])
```

3.3 Long Sort-Term Memory (LSTM)

3.3.1 Introduction to LSTM

In normal neural networks, it is impossible for them to remember a sequence of information, they can only make use of the specific input to make prediction, and assume the inputs are independent to each other. However in real world, information are not independent in most cases, and people always need to use previous knowledge (memory) to predict upcoming ones. These are big drawbacks of traditional neural networks. Recurrent neural networks (RNN) address this issue.



[9]

From the above diagram we can see that RNN is composed by multiple recurrent networks, so it can take information from several steps before the current state. Even though in theory, RNNs can deal with long-term dependency, but they are not able to do so in practice due to derivative vanishing problem, so that they cannot carry information from very far away states.

Long short-term memory (LSTM) network is an advanced version of RNN, which is designed to carry long-term dependencies. Below are two diagrams illustrate the structure difference between RNN and LSTM.

[9] All pictures in this section are from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

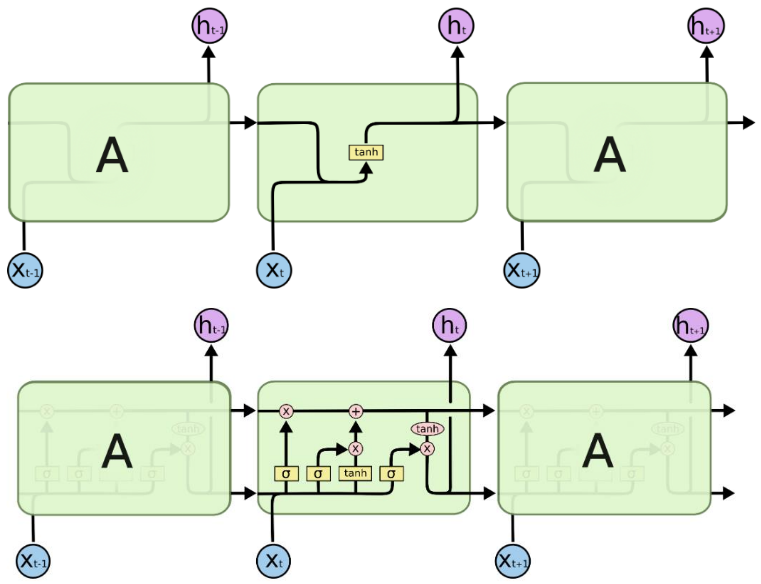


Fig.4 neural layers of RNN and LSTM

The upper one is the structure of RNN and the lower is LSTM. We can see that RNN only has one neural network layer in each network, while LSTM has 4 layers.

Input gate layer consists of a sigmoid layer and a tanh layer, which together decide what information to be stored.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\widetilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The forget get decides what information can be thrown away.

$$f_t = \sigma\,(W_f \cdot [h_{t-1}, x_t] + b_f)$$

This function will result in a number between 0 to 1, while 0 means totally forget this information, and 1 means completely keep it.

With the help of these layers, now we can update our cell state from $t-1$ to t.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}$$

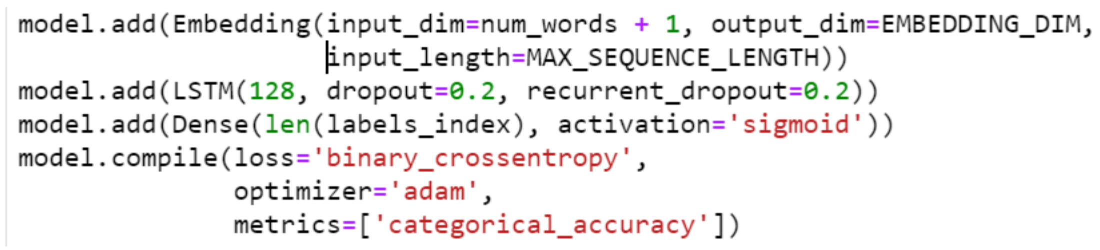
Finally, we will use a sigmoid layer followed by a tanh layer to decide what to output from this step.

$$O_t = \sigma\,(W_O \cdot [h_{t-1}, x_t] + b_O)$$

$$h_t = O_t * \tanh(C_t)$$

3.3.2 Code Implementation

The preprocessing part of LSTM network is the same as it in CNN, so we will skip this part here.



```
model.add(Embedding(input_dim=num_words + 1, output_dim=EMBEDDING_DIM,
                    input_length=MAX_SEQUENCE_LENGTH))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(len(labels_index), activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['categorical_accuracy'])
```

For the model, we implemented LSTM model from Keras, and a Dense layer to make dense connectivity, then use cross entropy as our loss function to classify the data.

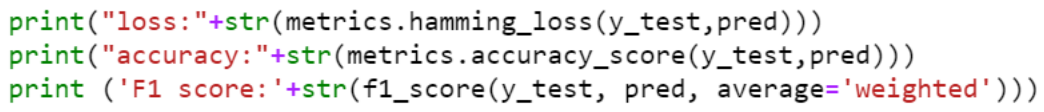
Table 3 Layers and Constructions of LSTM network

Embedding layer	**Input dimension**	num_words + 1
	Embedding dimension	100
LSTM	**Output dimension**	128
	Dropout factor	0.2
	Recurrent dropout factor	0.2
Dense layer	**Output dimension**	len(labels_index)
	Activation function	sigmoid
Optimizer	Adam	
Loss function	binary_crossentropy	
Metrics	categorical_accuracy	

4. Evaluation Methods and Results

4.1 Evaluation Metrics

In this project, we use evaluation metrics in `sklearn.metrics` to evaluate our classification result.



```
print("loss:"+str(metrics.hamming_loss(y_test,pred)))
print("accuracy:"+str(metrics.accuracy_score(y_test,pred)))
print ('F1 score:'+str(f1_score(y_test, pred, average='weighted')))
```

4.1.1 Loss

We use average hamming loss as our loss function in all the classifiers. Hamming loss is the fraction of labels that are incorrectly predicted. Because we are doing a multi label classification, it is hard to make the predicted label set exactly match the true label set, so we use hamming loss such that it penalizes individual incorrect label, instead of punishing entire label set if the predicted label set is not exactly match the true label set.

4.1.2 Accuracy

In multilabel classification, this accuracy function returns the subset accuracy, which means the predicted set of labels must strictly match with the true set of labels, otherwise is will be considered as incorrect prediction. The formula of calculating accuracy is:

$$acc(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y_i} = y_i)$$

4.1.3 F1 score

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.[10] The formula for F1 score is:

$$F_1 = 2 \times \frac{precision \ \times recall}{precision + recall}$$

For multi-label tasks, this is the average F1 score of each class with weights depending on the choice of "average" attribute. In our implementation, we use `weighted` to do average, which takes weighted mean to label-wise F1 score, considering the label imbalance.

$$precision = \frac{1}{\sum_{l \in L} |\hat{y_l}|} \sum_{l \in L} |\hat{y_l}| P(y_l, \hat{y_l})$$

$$recall = \frac{1}{\sum_{l \in L} |\hat{y_l}|} \sum_{l \in L} |\hat{y_l}| R(y_l, \hat{y_l})$$

$$F1 = \frac{1}{\sum_{l \in L} |\hat{y_l}|} \sum_{l \in L} |\hat{y_l}| F1(y_l, \hat{y_l})$$

where L is the set of labels, y_l is the set of true labels, $\hat{y_l}$ is the set of predicted labels. $P(y_l, \hat{y_l}) = \frac{|A \cap B|}{|A|}$, $R(y_l, \hat{y_l}) = \frac{|A \cap B|}{|B|}$.

4.2 Results and Analysis

The results of our 4 classifiers are shown in Table 4-7 respectively, and a comparison can be seen in fig.5.

Table 4 Results of One-VS-Rest SVM Classifier (baseline approach)

Loss	0.162226640159
Accuracy	0.436381709416
F1 score	0.799919964992

Table 5 Results of Simple (one conv. Module) CNN Classifier

Loss	0.155312810328
Accuracy	0.457795431976
F1 score	0.822835040053

Table 6 Results of Three Conv. Module CNN

Loss	0.160873882820
Accuracy	0.447864945382
F1 score	0.809543245817

[10] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score

Table 7 Results of LSTM Classifier

Loss	0.203376365442
Accuracy	0.370407149950
F1 score	0.770699702778

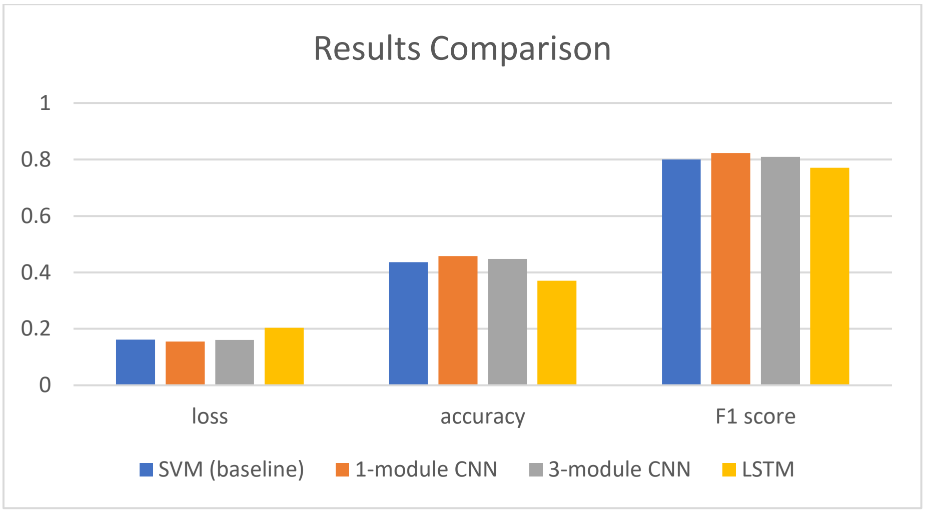


Fig.5 results comparison between 4 classifiers

From the results above and the plot below, we can see that two CNN classifiers both do better than the baseline approach, but LSTM does worse. This may because LSTM is more suitable for the situations that need long-term memory, and more commonly used in prediction instead of classification. CNN classifiers do a good job in this task, both give F1 score greater than 0.8. For accuracy, since we are doing multilabel classification, and the accuracy measurement requires all labels strictly matching the true label sets for the data to be considered as correct, it’s very hard to achieve, so the accuracies of our classifiers are not very high, but F1 score are all acceptable.