Mutual Fund Style Classification from Prospectus

Author: Mengjia Tan, Yifan Dong

Executive Summary

This project utilizes Natural Language Processing algorithms to analyze the text summary of mutual funds and predict their investment strategies. The data is splitted into training and testing sets, and notice that there are some funds that do not have labels that should be placed into the testing set. Then we transform cleaned text to vectors, and would be able to find closely related words in each context, and building knowledge bases allow us to score each sentence according to their distance to investment strategies. Obtaining the top sentences from each knowledge base, and splitting them into training and validation sets, we are ready to apply the Convolutional Neural Network(CNN) model. And we discuss the result in sections 3.4 and 3.5 in this report.

1.Data

We will work on 2 Corpuses (or set of texts). All the texts or publicly available texts dealing with mutual funds. More precisely, the text has been extracted from the prospectuses of Mutual funds.

MutualFundSummary
This file contains around 500 fund summaries in text files.

MutualFundLabels.csv
There are 4 main types: “Balanced Fund (Low Risk)”, “Fixed Income Long Only (Low Risk)”, “Equity Long Only (Low Risk)”, and Long Short Funds (High Risk), excluding one other type with only one fund.

1.1 Data Summary

Equity Long Only (Low Risk)	248
Fixed Income Long Only (Low Risk)	130
Balanced Fund (Low Risk)	84
Long Short Funds (High Risk)	4
Commodities Fund	1

Notice that the commodities fund(low risk) only takes one, so we firstly delete the corresponding data of this investment strategy, and the rest four types of labels are our Ys(our goal is to predict which investment strategy), and transform them into label 0,1,2,3 correspondingly. And our Xs is the ‘summary’ in summaries.

1.2 Data Splitting

We firstly perform the classic train-test-split, making 80% of the sample as the training set and 20% of the sample as the testing set.

While cleaning data we found that there are more summaries than fund names. We could use all of the summaries that have a corresponding label, but that would leave us with no test set to see the model's out-of-sample performance. Therefore, we opted to use 20% of the summaries with labels as the test set.

2. Build a word embedding dictionary by skip-gram model

2.1 Process skip-gram inputs

Our step 2 and early step 3 are essentially all data-preprocessing for the final CNN model. To begin with, we firstly tokenize the text by removing the stopwords and lemmatizations. The function tokenize gives out cleaned text words. Next, we want to transform words into vectors. To approach this, we need to build the vocabulary first by choosing embedding size, max vocabulary size, minimum occurrence for words(remove all words that do not appear at least this number of times), and how many times to reuse an input to generate a label, and also assign unique indices for each word. For the next step, following the deep_learning methodologies, we create OneHot vector from index, and define the batch_generator function.

2.2 Train the skip-gram model

Creating the compile autoencoder function, we are ready to train the skip-gram model with the vocabulary we created before. Using the encoder to vectorize each word, we get our word2vec dictionary, saved as "our_word2vec.txt". And we could compare this file with the "glove.6B.50d.txt" dictionary later.

3. Build knowledge bases

We want to build four knowledge bases for our training data, and thus four key_word lists to establish the desired knowledge bases.

The way we design to find keywords for each knowledge base is using tfidf to select common words in the summaries for different classes. From each class, we selected 50 most common words using the tfidf method. Then for each group of the 50 common words, we select 10 of them that do not appear more than once in the other 3 word groups. In simple terms, for each class we selected 10 common and "unique" words as the keywords to build out knowledge bases.

3.1 Measure distance of each summary to each knowledge base

Now we have built the four knowledge bases, we can select top sentences from the summaries. For each summary, we would select the top 5 sentences based on a knowledge base. Since we have 4 knowledge bases, we would select a total of 20 sentences for each summary. Once we have the featured sentences, we can transform them into vectors and feed them to the CNN model.

The number of sentences we select is debatable and critical to our model training. In class, we selected 5 sentences for a binary classification problem, whereas in the project, we are dealing with 4 classes so at first, selecting more featured sentences seemed natural to us. However, there are risks in selecting too many sentences. The risk is that if a knowledge base has words that appear more commonly in any kind of fund summaries, in other words, those words are more general, then selecting too many sentences would make them closer to that knowledge base, further affecting the prediction of the CNN model. Therefore, we could also say that the risk is that those "featured" sentences are not so special because we are selecting too many of them.

One way to modify the methodology is to select fewer featured sentences for each knowledge base. In this way we would make the "featured" sentences more valuable compared to other sentences in the summary. We think that could potentially improve the quality of the classification model.

3.2 Use CNN to predict investment strategy for each fund

In our understanding, there are infinitely many ways to build a CNN model. In the project we tried a few different models. All the models we tried have the structure as the CNN model in the class - two 1D convolutional layers and two densely connected NN layers. The difference of each model is its units.

By experimenting, we found that the more units the CNN has, the better in-sample performance it has. Furthermore, the more layers we add, the poorer the in-sample performance. This aligns to our intuition, the more complex the model is, the better it fits to the training data. However, the risk of overfitting increases as we add more and more units to the CNN model.

The appendix figure 1 shows the in-sample performance of the final CNN model.

3.3 CNN parameters tuning

After we were settled with the construction of CNN, we aimed to find the best parameter of the model, that is the batch size. By utilizing a for loop, we tried batch sizes from 8 to 20, and based on the validation accuracy, we concluded the best batch size to be 10. This CNN model achieved 82.7% accuracy in the sample.

The appendix figure 2 shows the different in-sample performance using different batch sizes.

3.4 Predict the investment strategy for each fund in the test data

The accuracy rate of the final CNN model is 49.5%. An ideal scenario would be that the model tries to classify different summaries as different groups but makes mistakes half of the time. A bad scenario would be the model always predicts summaries as the majority group of the class and achieved a 50% accuracy rate.

3.5 Further discussion

In the future, to avoid the bad scenario and to improve the out-of-sample performance, we think there are several options. First, we could use techniques to deal with imbalance data and see if the results get better. Second, we could try different methods to build knowledge bases. Last but not the least, we could try to build a simpler CNN model to enhance the out-of-sample accuracy, but that would also sacrifice the in-sample accuracy.

In conclusion, to improve the overall quality of the model, we could explore better ways to select keywords, control the number of featured sentences, and build different CNN models.

4. Appendix

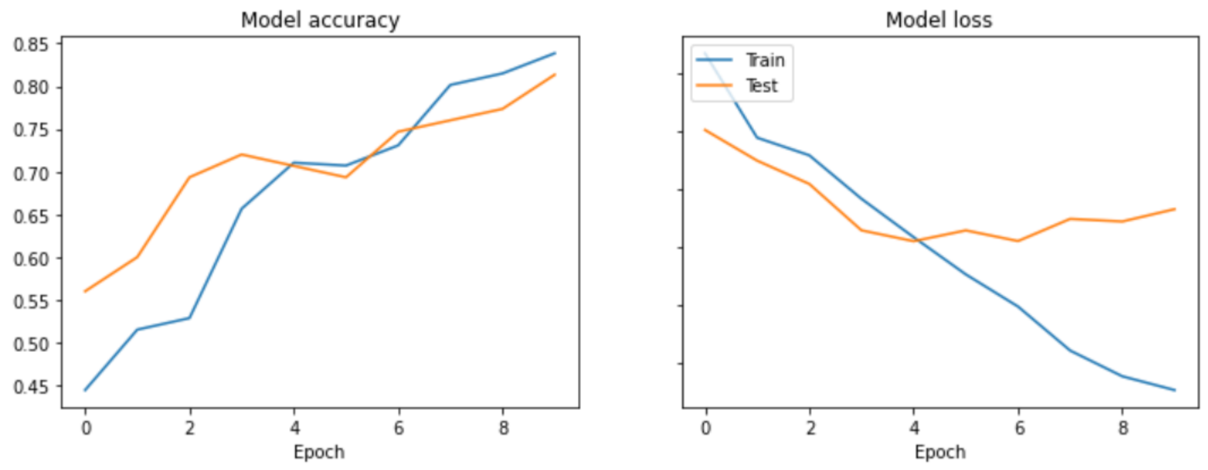

Figure 1: in-sample performance of the final CNN model

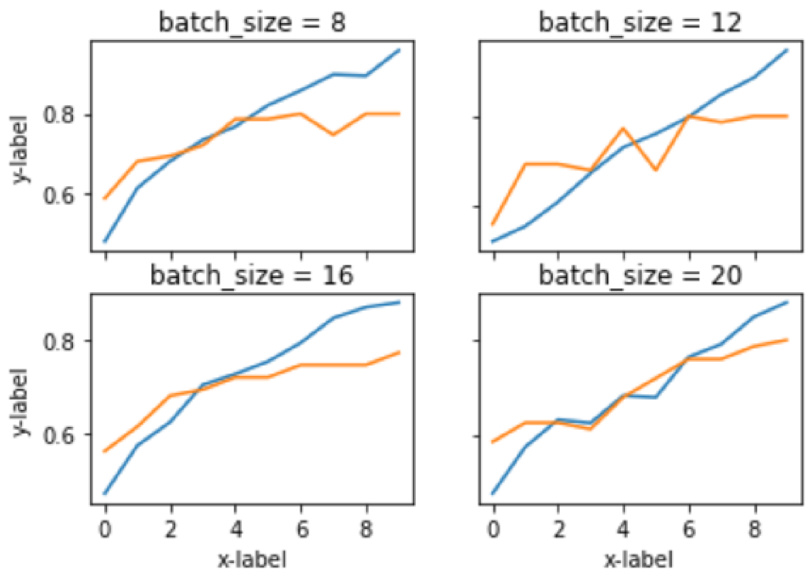

Figure 2: different in-sample performance using different batch sizes

5. Contribution

Mengjia Tan: data preprocessing(process skip-gram inputs), algorithm to find key_words, writing report

Yifan Dong: construct knowledge bases, parameter tuning, CNN tuning, testing result, writing reports