Mutual Fund Style Classification from Prospectus Applying NLP Techniques

MF 815 - Advanced Machine Learning

Zhiqi Duan, Shota Ichitaku

1.Abstract

Mutual fund prospectuses contain vital information regarding investment strategies, risk profiles, and portfolio compositions. Manually extracting and classifying this information across hundreds of funds is time-consuming and inefficient. This paper explores the application of Natural Language Processing (NLP) techniques to automatically classify mutual funds based on their investment strategies by understanding the context in summary prospectuses.

2.Executive Summary

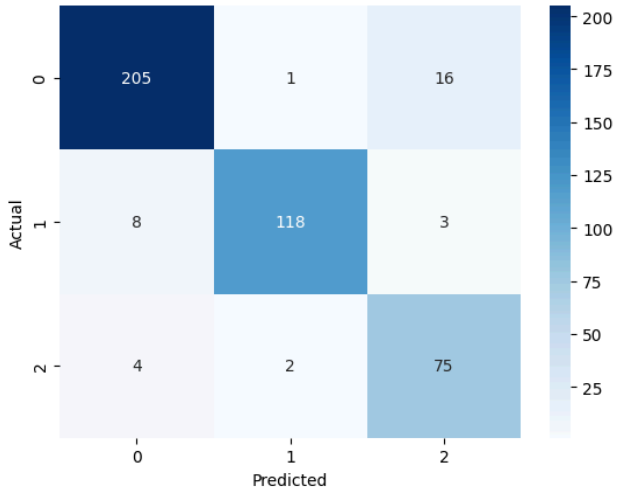
Our goal for this paper is to use Natural-Language-Processing techniques to classify mutual funds by their strategies. There are 5 strategies the mutual funds are taking: Balanced Fund (LowRisk), Fixed Income Long Only (Low Risk), Equity Long Only (Low Risk), and so on. We focused on the first three fund strategies to classify each mutual fund. We used the skip-gram model to build a word embedding dictionary from 259 mutual fund prospectus in the training set and build a knowledge base to extract important sentences from each document for our classification task. We used the extracted sentences to fit into the Machine-learning model and distinguished different fund strategies. We found that tuned support vector machines had the best performance and got the accuracy score of 0.75 and F1 score of 0.76.

3.Methodology

3.1 Pre-modelling

To build and evaluate our model, we first used Langchain and OpenAI API to apply RAG to all fund prospectus in the dataset. We asked the RAG algorithm to classify funds into Balanced Fund, Fixed Income Long Only, and Equity Long Only by asking a specific question to give us the answer as only the name of the fund strategies. In addition, we asked the RAG algorithm to return supporting evidence and the source it used to produce the result. We treat the answer from the RAG algorithm as a ground truth for later application. To verify the result from the RAG algorithm is reasonably making sense, we compared the result with “Mutual Fund Labels”, which contains strategy labels assigned by humans. To make the comparison easier, we assigned the major strategy names that we are interested in as numbers (0,1,2) and other names as null

values. The following heatmap shows the comparison of human made labels and labels from the RAG process.



We found that most of the time, the result from the RAG agrees with the mutual fund labels, however some of the evidence from the RAG was not strong enough to show the correct result and that caused the mismatch between human made labels and the answer from the RAG. To conclude our pre-modelling process, we updated our tables to contain the result from the RAG algorithm as numbers (0,1,2) and dropped all null values so that the table only contains the information about the mutual funds who are taking three fund strategies we are interested in.

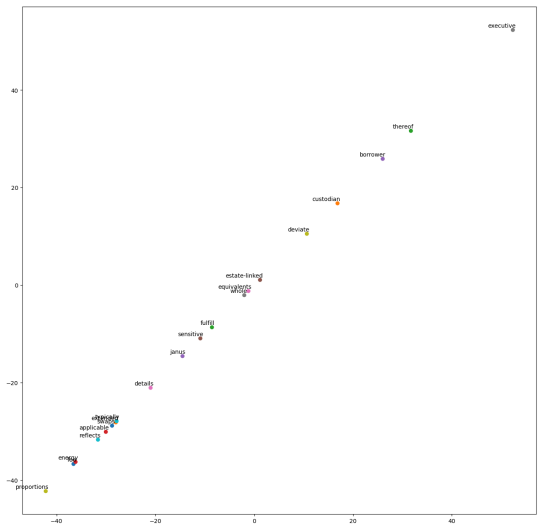
3.2 Data Splitting

To start building our model, we first load all files in the MutualFundSummary folder that contain all of the mutual fund prospectus documents. We loaded the file names and documents as lists and since we are only interested in three major strategies (Balanced Fund, Fixed Income Long Only, and Equity Long Only), we selected the funds that match with the table we created in the pre-modelling process. We then split the filtered lists into training, validation and test sets. We randomly selected 60% of the data as a training set and 20% each for the validation and test set.

3.3 Word2vec Skip-gram Model

After splitting the data to training, validation and test set, we used the training set to build a word-embedding dictionary by implementing a word2vec skip-gram model. In this process, our goal was to transform natural language to mathematical representations, specifically vectors and given a target word, predict its surrounding context words. The word2vec model will allow us to create unique vectors for each word in the given set of documents. These vectors will allow the machine to detect close meaning words in terms of the spatial distance. Here we used the documents in the training set (259 mutual fund prospectus) to build unique vectors for each word (word2vec). When we created word2vec, we applied a skip-gram model to In the word2vec skip-gram model, we want our model to learn a general context of a word. We input every word

in our documents and force it to output the local context of those words. We set window size as 3 to consider three words before and after the word as a local context. Once we have our input, output word pairs we then train the neural network with their one-hot representation (input, output). The following graph shows our spatial representation of the words in 2 dimensional space (PCA dimension decrease) after applying the word2vec skip-gram model.



(Restrictions)

- Limited the dimension of our word vectors to 50
- Only consider the 5000 most frequent words
- We set the window size to 3
- We set num_skips = 4, which will reuse four times of our input to generate a label

3.4 Building a Knowledge Bases

We want to allow machines to extract some relevant sentences from our mutual funds' prospectuses that are important for our machine to differentiate mutual funds' strategies. Constructing a knowledge base will give our machine an idea of how to classify mutual funds based on their strategies. By having a knowledge base, it will allow us to score each sentence according to their distance to our knowledge base. We wanted to extract the sentences that are close to our knowledge base. We first built a knowledge base by assigning some keywords that we think are important and related to fund strategy classification. The keywords we have selected were ['income', 'interest', 'bond', 'long','short', 'equity', 'stock','low', 'risk', 'risks','strategy', 'strategies','return', 'volatility', 'fund','funds','balance','balanced', 'yield','position']. Using our knowledge base and word2vec that we have created in an earlier step, we found 5 closer words in terms of distance in our word2vec space for each word in the keywords list and added those close words to our knowledge base. Later we used this expanded knowledge base to extract important and relevant sentences regarding fund strategy classification from each fund prospectus.

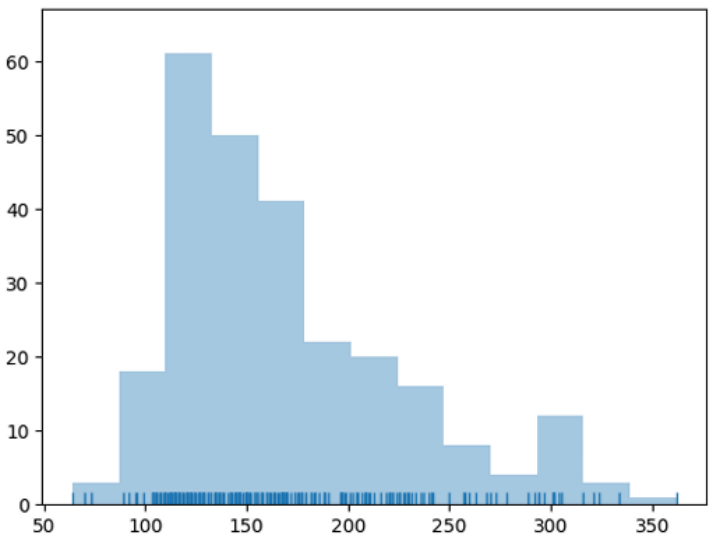
3.5 Sentence Distance

To extract important sentences from each mutual fund's prospectus, we can score each sentence according to their distance to the knowledge base. The process begins by segmenting the full prospectus text into individual sentences. For each sentence, we calculated a representative vector, known as its barycenter, by averaging the pre-trained word2vec embeddings of its constituent words. This sentence vector's relevance was then assessed by computing its cosine distance to the word2vec vector of each keyword within our knowledge base. A smaller cosine distance signifies greater semantic similarity. To assign a single score reflecting the sentence's overall relevance to the knowledge base, we averaged the distances to the 5 closest most similar keywords. Sentences were then ranked according to this average distance score in ascending order (lower scores indicate higher relevance). Finally, we selected and concatenated the top 10 sentences with the lowest scores, effectively creating a targeted summary focused on the concepts defined in our knowledge base. Lastly, we assigned our extracted sentences as a dependent variable to create a new data frame for training, validation, and test set for preparing for the modelling process.

4.Data Pre-Processing

4.1 Choosing the Maximum Sequence Length

To determine an appropriate value for the document length parameter, we visualized the distribution of the number of words per document in the training dataset. Specifically, we tokenized each document and counted the number of tokens. A histogram was then plotted to observe how document lengths are distributed, helping us select a reasonable truncation or padding length for input sequences in the model.



Based on the distribution, we set max len = 200, which captures the majority of documents while minimizing information loss and avoiding excessive padding. This value balances model efficiency and the preservation of meaningful content.

Documents longer than 200 words were truncated, while shorter ones were padded with zeros to ensure uniform input shape for the neural network. Then we applied this transformation to the tokenized training, validation, and the test sets, using post-padding and post-truncating to preserve the beginning of each sequence.

4.2 Construct Embedding Matrix

To represent words with meaningful vector embeddings, we built an embedding matrix using pre-trained word2vec vectors. The vocabulary was created using Keras's Tokenizer, which assigned a unique index to each of the 2,500 most frequent words in the training set.

```
embedding_matrix.shape
```

```
(2201, 50)
```

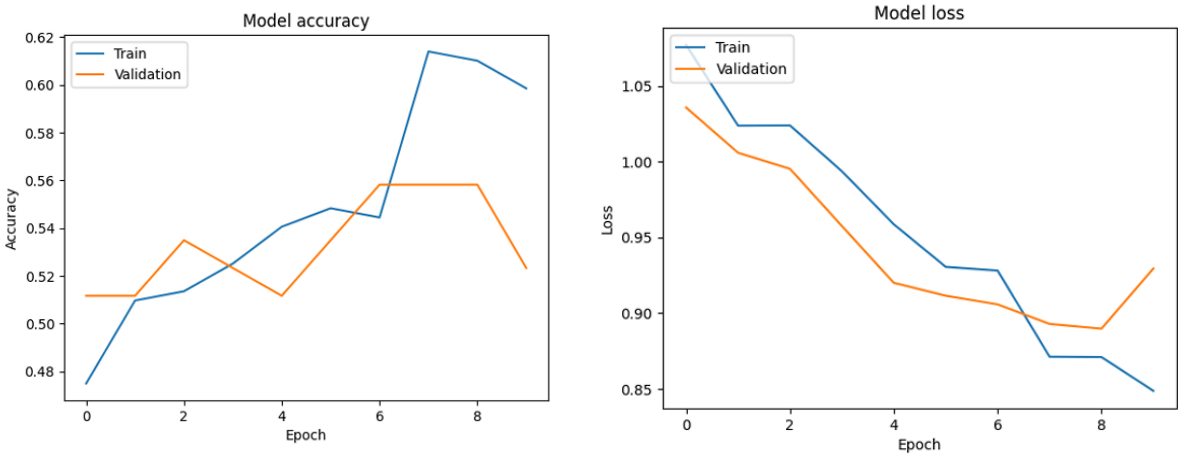
We initialized a matrix of size (len(word_index) + 1, word_dimension) to hold the embeddings. For each word in the vocabulary, we looked up its word2vec vector. If found, we stored it in the matrix; otherwise, the row remained zero. This embedding matrix was then used to initialize the embedding layer of our model, allowing it to start with informative word representations.

5.Model Building and Hyperparameter Tuning

5.1 One dimensional Convolutional Neural Network

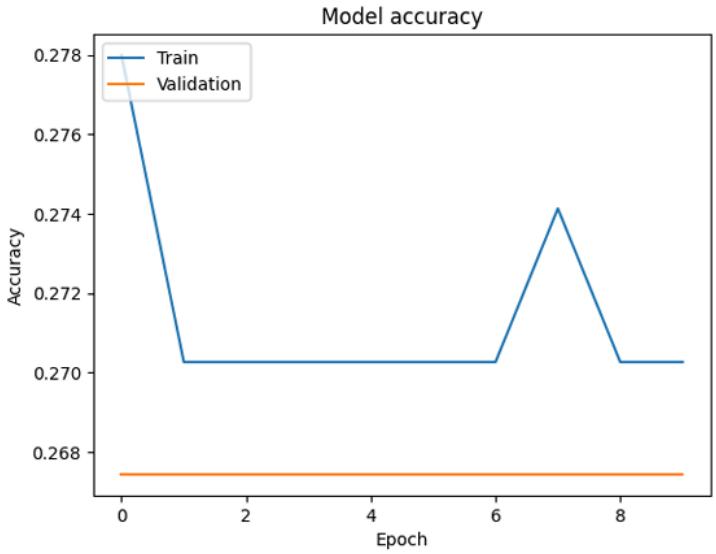
To perform text classification, we implemented a Convolutional Neural Network (CNN) using Keras. The model begins with an embedding layer initialized with pre-trained word vectors, ensuring that word representations capture semantic information from the start. This layer is not trainable to preserve the integrity of the pre-trained embeddings.

The architecture includes two 1D convolutional layers with 64 and 32 filters respectively, each using a kernel size of 5 and ReLU activation. These are followed by max-pooling layers to downsample the feature maps and reduce dimensionality. The output is then flattened and passed through a dense layer with 128 units and ReLU activation, followed by a dropout layer with a 50% rate to prevent overfitting. The final layer is a softmax layer with three output units, corresponding to the three target classes.



The model was compiled using the Adam optimizer and trained with the sparse categorical cross entropy loss function for 10 epochs with a batch size of 16. Accuracy on both the training and validation sets was monitored during training. While training accuracy steadily increased, validation accuracy plateaued around 55%, indicating a modest generalization performance. The training and validation accuracy over epochs were plotted to assess the model's learning dynamics.

5.2 Recurrent Neural Network



Similar to our previous approach using Convolutional Neural Networks (CNN), we constructed a Recurrent Neural Network (RNN) model employing a Bidirectional LSTM architecture for text classification. We visualized the training and validation accuracy over 10 epochs. As shown in the resulting plot, both training and validation accuracy remained consistently low and showed minimal improvement throughout training. This

indicates that the model failed to effectively learn from the data. Upon further analysis, we suspect that the choice of loss function may have contributed to this outcome. Specifically, we used binary cross entropy despite the task involving three classes, which likely led to poor optimization performance. This suggests that using a categorical loss function, such as categorical cross entropy, would be more appropriate for multi-class classification.

5.3 Support Vector Machine

5.3.1 Document Vectorization Methods for Text Classification

To enable the application of traditional machine learning classifiers on text data, we designed two methods to transform indexed text sequences into fixed-length document vectors using pre-trained word embeddings.

The first method computes the mean vector representation by averaging the embeddings of all non-zero word indices in a document.

The second method introduces a norm-weighted averaging approach, where each word vector is weighted by its L2 norm before aggregation, thus giving higher importance to more informative words. For both methods, documents without valid word indices are represented by zero vectors. These vectorized representations were then used as input features for subsequent classification tasks.

5.3.2 Linear Kernel

We applied a Support Vector Machine (SVM) classifier with a linear kernel and regularization parameter C=0.1 to the vectorized text data. The input features were derived from the text index representations using document-level embedding methods. This approach aims to evaluate the effectiveness of simple linear decision boundaries in classifying textual data when represented as fixed-length vectors.

5.3.3 Hyperparameter Tuning for SVM Classification

We applied a SVM classifier to the vectorized text data, aiming to find an optimal configuration through hyperparameter tuning. To begin, we defined a broad search space for key parameters, including the regularization strength C, kernel type (linear, rbf, poly), gamma values, polynomial degree, class_weight, and the use of the shrinking heuristic. This was motivated by the need to explore both linear and non-linear decision boundaries and to address possible class imbalance.

Using RandomizedSearchCV with 5-fold stratified cross-validation and 50 iterations, we searched for the best-performing model based on accuracy. The search was parallelized to accelerate the fitting process. The best model identified used a polynomial kernel with C ≈ 65.84, gamma = 1, degree = 3, class_weight = 'balanced', and shrinking = False.

```
SVC(C=np.float64(65.84106160121607), class_weight='balanced', gamma=1,
    kernel='poly', shrinking=False)
```

Finally, the selected model was retrained on the combined training and validation sets to fully utilize the available data. This procedure ensured a well-tuned model with a balance between complexity and generalization performance.

6. Model Comparison and Selection

To identify the most effective classification model, we compared four different approaches: a Convolutional Neural Network (CNN), a Recurrent Neural Network (RNN), a linear SVM, and a hyperparameter-tuned SVM. All models were trained and evaluated on the same text classification task using vectorized input representations. Performance was assessed based on validation set accuracy and stability across multiple runs to ensure robustness.

Both linear SVM and RNN models showed consistently low accuracy on the validation set. CNN achieved better performance, and the hyperparameter-tuned SVM model significantly outperformed all others. After conducting randomized hyperparameter search with cross-validation, the tuned SVM demonstrated superior generalization capability.

	precision	recall	f1-score	support
0.0	0.76	0.70	0.73	37
1.0	0.73	0.79	0.76	28
2.0	0.74	0.77	0.76	22
accuracy			0.75	87
macro avg	0.75	0.75	0.75	87
weighted avg	0.75	0.75	0.75	87

```
array([[26,  6,  5],
       [ 5, 22,  1],
       [ 3,  2, 17]])
```

We selected the tuned SVM as the final model and retrained it on the combined training and validation sets. Evaluation on the test set showed strong classification performance, with the detailed classification report and confusion matrix confirming balanced and accurate predictions across classes. This validates the robustness and effectiveness of the selected model for our text classification task.

Appendix

Task allocation

Section 3 (methodology) and data preparation was done by Shota Ichitaku and section 4 and 5 (data-pre processing and model building) was done by Zhiqi Duan. Each of us was responsible for the coding associated with our respective sections. The project involved an equal distribution of workload, and consistent communication between both members allowed us to efficiently integrate the different components and successfully complete the research.