MF815 Final Project Report
Advanced Machine Learning - Spring 2022
Chi Kang Kuo / Shuxian Hong / Zhihao Zhang / Zelin Zhao
Due: May 6, 5 pm

Mutual Fund Style Classification from Prospectus

1. Executive Summary

This project applies different machine learning methods to classify the investment strategy mutual funds use. There are four investment strategies in total, and our goal is to predict which style a mutual fund uses. As for specific steps, in the beginning, we implement a skip-gram model to build a word embedding dictionary by using data in the training set. After that, we create four knowledge bases, each closely associated with one of the four investment strategies. With the word embedding dictionary and knowledge bases, relevant sentences with the investment style in the summaries will be extracted using the match extraction method. Therefore, each data will have four unique numerical distances to the four knowledge bases, respectively, which will be used as the input data in the machine learning methods. We then divide all the input data into training and testing sets. Our machine learning models will feed on data in the training set and then test on those in the testing set. After completing training, the classification models can forecast the investment style for funds from data in the testing set. However, due to the fact that there are only four funds with the long-short strategy in our data, we further apply outlier detection methods to find them.

2. Methodology

2.1 Data

The data this project uses include 466 mutual fund summaries. There are five investment styles in total out of all the data: “Balanced Fund (Low Risk),” “Fixed Income Long Only (Low Risk),” “Equity Long Only (Low Risk),” “Long Short Funds (High Risk),” and “Commodities Fund (Low Risk).” Since only one fund belongs to the “Commodities Fund (Low Risk)” strategy, we delete it from our dataset. Thus only fund summaries with the four investment styles remain. These summaries are then used as input for the skip-gram model.

2.2 Skip-gram model & Knowledge base Set up

Word2vec skip-gram model is a deep learning model used to find the most related word from the word database for given words. After cleaning all the summaries collected from the mutual fund summary folder, we removed all stop words from the text, such as 'doe,' 'ha,' and 'wa,' which do not impact the meaning of the sentences. We set the maximum feature as 5000, as the vocabulary set will only contain 5000 most frequent words in further steps. Next, we tokenize all text and get a list of words remaining input of the skip-gram model. The model works as a technique to create each word a unique vector putting words with the same context close to each other in terms of spatial distance. For a given local context, the model will loop through each word in the corpus, source the closest n (window size) words before and after the target word, and form pairs of contexts as training samples.

Skip-Gram

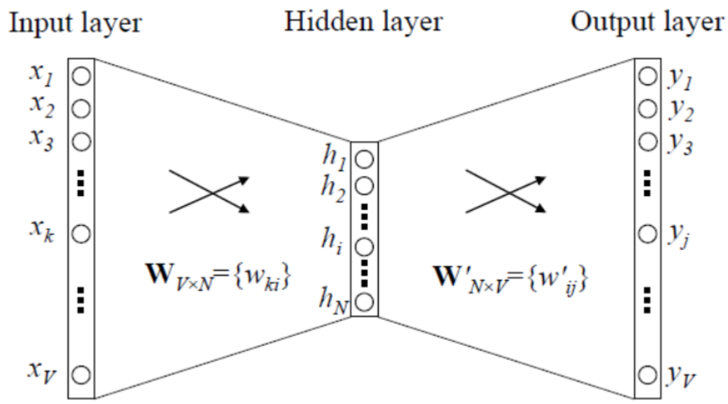


Figure 1

In our project, we set the skip window as 3, which will generate six pairs of contexts for a given word, and then we randomly select 4 of them. We start by building a vocabulary where each word in it was assigned to a unique identifier. Rare words were replaced with UNK tokens. The size of vocabulary created is 3458. Next, we use the batch_generator function to create the input of our model. We set batch size as 128, which means 128 rows in each batch. The function will create 128 training rows, each row as a one-hot word representation. After all these preparation works, we train the skip-gram model:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 3458)]            0

 dense (Dense)               (None, 50)                172950

 dense_1 (Dense)             (None, 3458)              176358

=================================================================
Total params: 349,308
Trainable params: 349,308
Non-trainable params: 0
_________________________________________________________________
```

Figure 2

We finally build word representation, generate all words' vectorial representation and store it in the word2vec file. To visualize how it works, we use the find closer word function and try to find the five closest words of 'equity,' 'stock,' and 'fixed-income' as examples. Following is the result.

```
words close to equity : equity, hedged, global, small, mid
words close to stock : stock, potentially, declining, supply, basic
words close to fixed-income : fixed-income, rely, lower-rated, emerging-market, bradley
```

Figure 3

Knowledge base is an essential part of our project. We first set a unique list of keywords for each strategy. Those keywords are selected based on the target strategy's definition and some most frequently used descriptions when interpreting the strategy. For instance, for Fixed Income Long Only strategy, we choose words such as 'fixed', 'principal', 'coupon', 'yield', 'bond', 'premium', 'income', 'derivative’. The keyword lists are fed as input of the following function to create knowledge bases by taking close neighbors of each word.

2.3 Sentence Scoring Function: Match Extraction

As we have the knowledge base, the next step is to extract the sentence that deals with investment strategies from the summaries. To achieve this goal, we need a method to score each sentence according to their distances to each of the four knowledge bases. One of the scoring methods is match extraction, which counts the number of words in the intersection of the knowledge base and the sentence. This function is highly dependent on the number of neighbors chosen to create the knowledge base. Thanks to the match extraction, related sentences to the investment strategy will be extracted according to their scores. The arithmetic means of these scores will then be used as input data for the classification models.

2.4 Classification Algorithm

After getting data from the summaries, our next step is to split the data into training and testing sets. The research chooses two-thirds of the data as the training data, and the rest is the test data. After that, the researcher tries two different classification models to compare the result with each other and decides which one has better performance on prediction accuracy.

2.4.1 Deep Neural Network Algorithm

In the simplest case, a neural network with a certain degree of complexity (usually with at least two layers) can be called a deep neural network (DNN). Deep networks process data in a complex way by using mathematical modeling.

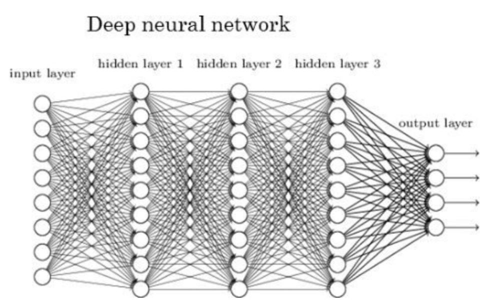


Figure 4

The research utilizes the randomized search on hyperparameters and optimizes them by 3-folded cross-validated searches over parameter settings.

```
rnd_search_cv.best_params_

{'drop': 0.37814891903848746,
 'learning_rate': 0.019533413235998425,
 'n_hidden': 1,
 'n_neurons': 36}
```

Figure 5

```
DNN_model.summary()

Model: "sequential_92"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_305 (Dense)           (None, 36)                180

 dropout_213 (Dropout)       (None, 36)                0

 dense_306 (Dense)           (None, 4)                 148

=================================================================
Total params: 328
Trainable params: 328
Non-trainable params: 0
_________________________________________________________________
```

Figure 6

After that, this research concludes the best estimated model for DNN and generates the prediction array for the testing dataset. By comparing the predicted result to the true labels, the accuracy of this model can reach 90.26%.

	Predicted Balanced	*Predicted Equity*	*Predicted Fixed Income*	*Predicted Long Short*
True Balanced	*18*	*6*	*2*	*0*
True Equity	*1*	*83*	*2*	*0*
True Fixed Income	*1*	*1*	*38*	*0*
True Long Short	*0*	*1*	*1*	*0*

Table 1

2.4.2 Logistic Regression Algorithm

Logistic regression is a classification algorithm that assigns observations to a discrete set of classes. This method as a predictive analysis algorithm transforms the output to return a probability value. The logistic regression hypothesis tends to limit the cost function between 0 and 1. Therefore, linear functions fail to represent it as they can have a value greater than one or less than 0, which is not possible per the logistic regression hypothesis.

The research tries to fit a linear model with coefficients to minimize the residual sum of squares between the data that matches each kind of fund group and code numbers for each fund group by the linear approximation. The best accuracy obtained by this model can reach 83%, and the confusion matrix for the prediction is as follows (Table 2).

	Predicted Balanced	*Predicted Equity*	*Predicted Fixed Income*	*Predicted Long Short*
True Balanced	*8*	*9*	*9*	*0*
True Equity	*0*	*84*	*2*	*0*
True Fixed Income	*1*	*3*	*36*	*0*
True Long Short	*0*	*1*	*1*	*0*

Table 2

2.5 Outlier Detection Algorithm

2.5.1 Motivation

After two classification algorithms were trained, our team discovers that the model fails to identify and distinguish the ‘Long Short Fund (High Risk)’ class (denoted as ‘outlier’ afterward) from the other three because of the rarity of its presentation inside the training set and testing set (only four data points). It requires methods other than the selected classification algorithm to detect and classify. As a result, our team decides to implement two clustering models, including Local Outlier Factor and K-means, for this uncommon strategy.

2.5.2 K-means Algorithm
2.5.2.1 Introduction of Algorithm
K-means algorithm is a clustering algorithm dedicated to group unlabeled data into clusters. It is widely used to label data and capture characteristics. The mathematical representation of the algorithm can be seen as follows

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

which the algorithm aims to partition n observation (x) into k set so as to minimize the within-cluster sum of square, in which $\boldsymbol{\mu}$i inside the equation denotes the centroid inside the ith cluster.

2.5.2.2 Implementation
The strategy we use to detect the outlier is very straightforward. First, by training K-means clustering with the training set, we can acquire k groups, the optimal k is decided by the average of the Silhouette Score. The ith Silhouette Score can be formulated as

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} d(i,j) \quad b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i,j) \quad s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

in which CI and CJ is defined to be different cluster, d(i, j) is the distance between data point i and j, and |CI| denote the number of points in cluster I. Secondly, by observing which group the real outlier is in, Our team will mark that group as an 'outlier group'. Third, by predicting the group number using the testing set, we can know which sample point of the testing set is in the 'outlier group', then, we will mark it as the 'Long Short Fund-(High Risk)' class.

2.5.2.3 Result
Our team tests k from 4 to 100 and calculates the Silhouette Score (figure 7). By searching the maximum of the score, we get our best k = 47.

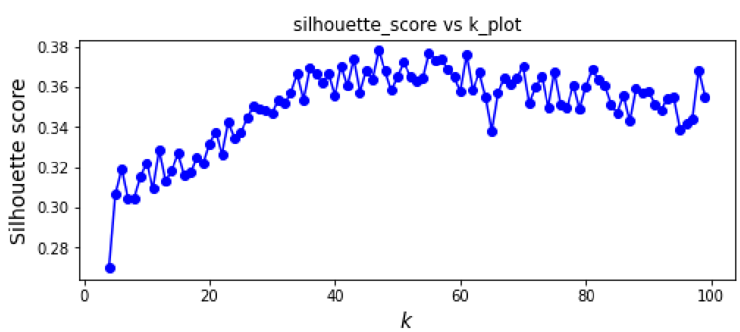


Figure 7

However, this method's prediction is not successful as the confusion matrix (Table 3) compares the predicted outlier and the real outlier, indicating that although the algorithm can mark only a very small amount of data as an outlier and minimizes the effect on other predictions, it cannot capture any real outlier and is therefore not very useful in this case.

	Predicted Outlier	*Predicted Other*
True Outlier	*0*	*2*
True Other	*6*	*146*

Table 3

2.5.3 Local Outlier Factor Algorithm

2.5.3.1 Introduction of Algorithm

Local Outlier factor (denoted as LOF afterwards) is an algorithm that aims at detecting the anomalies of the data set by observing the relative density of the data point compared to the neighbor. Its mathematical representation is as follows

$$\mathrm{lrd}_k(A) := 1 / \left(\frac{\sum_{B \in N_k(A)} \text{reachability-distance}_k(A, B)}{|N_k(A)|} \right) \qquad \mathrm{LOF}_k(A) := \frac{\sum_{B \in N_k(A)} \frac{\mathrm{lrd}_k(B)}{\mathrm{lrd}_k(A)}}{|N_k(A)|} = \frac{\sum_{B \in N_k(A)} \mathrm{lrd}_k(B)}{|N_k(A)| \cdot \mathrm{lrd}_k(A)}$$

In which Nk(A) denotes the set of k nearest neighbors. If LOFk(A) is bigger than one, then the algorithm distinguishes it as having a lower density than the neighbors and therefore judges it as an outlier.

2.5.3.2 Implementation

First, our team decides what parameter k can yield the best performance by observing the error rate in the testing data using multiple k values. Secondly, by searching the DNN predicted probability of those data points for each class that's marked as outliers, if there exists no probability that's larger than our setup threshold, meaning there is no certainty in that specific prediction, our team will alter its prediction result from the neural network and leave those that exceed such threshold. Third, our team will mark those that are still identified as outliers as the 'Long Short Fund (High Risk)' class.

2.5.3.3 Result

By trying k from 1 to 100, The research found that setting k to be 1 can yield the lowest error rate for predicting outliers to our testing set (figure 8). Secondly, by simple trial and error, we discover that setting the probability threshold to be 90.1% can balance the inclusion of outliers prediction and the prediction precision. With hyper-parameters tuned, the confusion matrix is as follows (table 4)

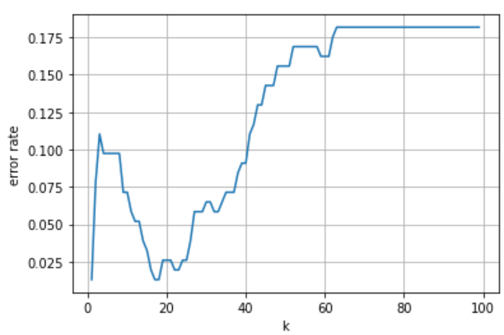


Figure 8

	Predicted Outlier	*Predicted Other*
True Outlier	*1*	*1*
True Other	*7*	*145*

Table 4

shows that LOF can capture one out of two outliers in the testing set with 8 prediction numbers compared with 6 by k-means algorithm. Moreover, it reduces the prediction accuracy by only about 2% percent overall (to 88.31%) and can therefore be useful in future prediction. The following (table5) is the final prediction using the DNN-LOF hybrid model.

	Predicted Balanced	*Predicted Equity*	*Predicted Fixed Income*	*Predicted Long Short*
True Balanced	*16*	*4*	*2*	*4*
True Equity	*1*	*83*	*2*	*0*
True Fixed Income	*1*	*0*	*36*	*3*
True Long Short	*0*	*0*	*1*	*1*

Table 5

3. Result & Discussion

By combining NLP, classification, and clustering methods, our team managed to achieve roughly 88-90% of predicting accuracy with the hybrid SG-DNN-LOF model. The LOF can also be modulated by the users to fit their perspective of importance for predicting rare events. The final predicted confusion matrix is presented in table 5. There are indeed other methods that come to our team's mind, such as calculating cosine distance instead of word matching, combining multiple CNN and RNN algorithms that act on predicting one to all, or the mixture model for outlier prediction. However, these methods cannot yield meaningful predictions in our research process and require more sophisticated tuning techniques or better quality of data. As a result, the model this research constructed shows the potential of combining models in the language processing and predicting for mutual fund style and further business use.

4. Teammate Contribution

Our group worked efficiently and effectively on this project. The division of the task went well, and all team members contributed equally (25% for each):

Zhihao Zhang: Data Cleaning / Tokenizing, Knowledge base Set up
Shuxian Hong: Skip-gram model, Word Matching Extraction
Zelin Zhao: Classification Algorithm
Chikang Kuo: Outlier Detection Algorithm

Appendix

Setup

In [ ]:
```
# Import the libraries

import os
import sys
from IPython.display import HTML, display
from sklearn.metrics import silhouette_score


import numpy as np
import pandas as pd
import tensorflow as tf
from math import ceil
from scipy.spatial.distance import cosine

import matplotlib.pyplot as plt
import seaborn as sns

import collections
import random
import time
import string
import re

import tensorflow as tf
from tensorflow import keras
from keras.callbacks import EarlyStopping, ModelCheckpoint

from scipy.stats import reciprocal, uniform
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.cluster import KMeans


import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from sklearn.linear_model import LogisticRegression

from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn import metrics

from sklearn.neighbors import LocalOutlierFactor
from sklearn.metrics import plot_confusion_matrix, plot_roc_curve, roc_curve, auc, confusion_matrix

from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Input, Embedding, Dense, Convolution1D, MaxPooling1D, GlobalMaxPooling1
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

Define function to access file

In [ ]:
```
# We now set the directory to access the data
def find(name, path):
    for root, dirs, files in os.walk(path):
        if name in files:
            return os.path.join(root, name)


#SUMMARY_PATH = os.path.join(DIRECTORY, "Data", "MutualFundSummary")
SUMMARY_PATH = '/content/drive/MyDrive/728, 793, 703, 815/MutualFundSummary'
SUMMARY_LABELS_PATH = '/content/drive/MyDrive/728, 793, 703, 815/MutualFundLabels.csv'

glove_word2vec = 'glove.6B.50d.txt'
our_word2vec = 'word2vec_perso.txt'
```

In [ ]:
```
# Progress bar
def progress(value, max=100):
    return HTML("""
        <progress
            value='{value}'
            max='{max}',
            style='width: 100%'
        >
            {value}
        </progress>
    """.format(value=value, max=max))

# Save a word2vec dictionary.
def save_word2vec(filename):
    with open(os.path.join('/content/drive/MyDrive/728, 793, 703, 815', filename),'a' , encoding='utf-8') a
        for k, v in word2vec.items():
            line = k+' '+str(list(v)).strip('[]').replace(',','')+'\n'
            f.write(line)

# Load a word2vec dictionary.
def load_word2vec(filename):
    word2vec = {}
    with open(os.path.join('/content/drive/MyDrive/728, 793, 703, 815', filename), encoding='utf8') as f:
        for line in f:
            try :
                values = line.split()
                word = values[0]
                vec = np.asarray(values[1:], dtype='float32')
                word2vec[word] = vec
            except :
                None
    return word2vec

# read the repo in PATH and append the texts in a list
def get_data(PATH):
    list_dir = os.listdir(PATH)
    texts = []
    fund_names = []
    out = display(progress(0, len(list_dir)-1), display_id=True)
    for ii, filename in enumerate(list_dir) :
        with open(PATH+'/'+filename, 'r', encoding="utf8") as f :
            txt = f.read()
            try :
                txt_split = txt.split('<head_breaker>')
                summary = txt_split[1].strip()
                fund_name = txt_split[0].strip()
            except :
                summary = txt
                fund_name = ''
        texts.append(summary)
        fund_names.append(fund_name)
        out.update(progress(ii, len(list_dir)-1))
    return fund_names, texts
```

Create list of stop words

In [ ]:
```
stop_words = set(stopwords.words("english")+list(string.punctuation)+['``',"''"]+["]","[","*"]+['doe', 'ha
```

In [ ]:
```
max_features = 5000 # we will only consider the 5000 most frequent words to create the vectors.
# This value is the size of the vocabulary that we use to vectorize.
```

In [ ]:
```
# Get the summaries
fund_names, summaries = get_data(SUMMARY_PATH)
```

tokenizer function

In [ ]:
```
# clean and tokenize the text -> we don't want to lemmatize
def tokenizer(txt):
    txt = txt.replace('\n', ' ').replace('\t', ' ').lower()
    word_tokens = word_tokenize(txt)
    filtered_sentence = [w for w in word_tokens if not w in stop_words]
    filtered_sentence = [w for w in filtered_sentence if re.sub("[^A-Za-z ]+",'',w) != '']
    return filtered_sentence
```

In [ ]:
```
# tokenize the text in all summaries
text_words = np.concatenate([tokenizer(summary) for summary in summaries])
```

In [ ]:
```
# check test words
print(text_words[:20])
```

```
['ab' 'arizona' 'portfolio' 'investment' 'objective' 'investment'
 'objective' 'portfolio' 'earn' 'highest' 'level' 'current' 'income'
 'exempt' 'federal' 'income' 'tax' 'state' 'arizona' 'personal']
```

Process Skip-Gram model Input

In [ ]:
```
# Training Parameters
batch_size = 128 # The model will be trained batch per batch and one batch contains 128 rows
num_epochs = 2 # The model will go through all the data twice
```

In [ ]:
```
# Word2Vec Parameters
embedding_size = 50 # Dimension of the embedding vector
max_vocabulary_size = 5000 # Total number of different words in the vocabulary
min_occurrence = 10 # Remove all words that does not appears at least n times
skip_window = 3 # How many words to consider left and right
num_skips = 4 # How many times to reuse an input to generate a label
```

In [ ]:
```
# Build the dictionary and replace rare words with UNK token
count = [('UNK', -1)]
# Retrieve the most common words
count.extend(collections.Counter(text_words).most_common(max_vocabulary_size - 1))
# Remove samples with less than 'min_occurrence' occurrences
for i in range(len(count) - 1, -1, -1):
    if count[i][1] < min_occurrence:
        count.pop(i)
    else:
        # The collection is ordered, so stop when 'min_occurrence' is reached
        break
```

In [ ]:
```
# give a unique id to each words in the vocabulary
word2id = dict()
for i, (word, _)in enumerate(count):
    word2id[word] = i
id2word = dict(zip(word2id.values(), word2id.keys()))
vocab_size = len(id2word)
```

In [ ]:
```
# check size of vocabulary
print ('size of the vocabulary : '+str(vocab_size))
```

```
size of the vocabulary : 3458
```

create data

In [ ]:
```
# create data
data = list()
unk_count = 0
for word in text_words:
    # Retrieve a word id, or assign it index 0 ('UNK') if not in dictionary
    index = word2id.get(word, 0)
    if index == 0:
        unk_count += 1
    data.append(index)
count[0] = ('UNK', unk_count)
```

In [ ]:
```
print(data[:20])
```

```
[1285, 0, 7, 3, 123, 3, 123, 7, 1085, 289, 471, 181, 26, 563, 77, 26, 61, 264, 0, 1087]
```

Build OneHot vector and generate training batch

In [ ]:
```
# build OneHot vector from index
def to_one_hot(data_point_index, vocab_size):
    temp = np.zeros(vocab_size)
    temp[data_point_index] = 1
    return temp
```

In [ ]:
```
# Generate training batch for the skip-gram model
def batch_generator(batch_size, num_skips, skip_window, vocab_size):
    data_index = 0
    while True :
        assert batch_size % num_skips == 0
        assert num_skips <= 2 * skip_window
        # batch is filled with 128 inputs
        batch = np.ndarray(shape=(batch_size), dtype=np.int32)
        # labels is filled with 128 outputs
        labels = np.ndarray(shape=(batch_size), dtype=np.int32)
        span = 2 * skip_window + 1
        # buffer keep track of the visited indexes visited
        buffer = collections.deque(maxlen=span)
        if data_index + span > len(data):
            data_index = 0
            # We stop the loop when we went through all the corpus
            break
        buffer.extend(data[data_index:data_index + span])
        data_index += span
        for i in range(batch_size // num_skips):
            # Take the context current word
            context_words = [w for w in range(span) if w != skip_window]
            # Randomly select num_skips words in the context
            words_to_use = random.sample(context_words, num_skips)
            for j, context_word in enumerate(words_to_use):
                # Creates one raw data
                batch[i * num_skips + j] = buffer[skip_window]
                labels[i * num_skips + j] = buffer[context_word]
            if data_index == len(data):
                buffer.extend(data[0:span])
                data_index = span
            else:
                buffer.append(data[data_index])
                data_index += 1
        # Backtrack a little bit to avoid skipping words in the end of a batch
        data_index = (data_index + len(data) - span) % len(data)

        # translate word index to on-hot representation
        batch_one_hot = np.array([to_one_hot(b, vocab_size) for b in batch])
        labels_one_hot = np.array([to_one_hot(l, vocab_size) for l in labels])

        # output one batch
        yield batch_one_hot, labels_one_hot
```

Train the skip-gram model

In [ ]:
```
# Create en compile the Autoencoder
def creat_word2vec_model():
```

```
    input_word = Input(shape=(vocab_size,))

    encoded = Dense(embedding_size, activation='linear')(input_word)
    decoded = Dense(vocab_size, activation='softmax')(encoded)

    # The autoencoder is the whole model with hidden layer contected to the output layer.
    autoencoder = Model(input_word, decoded)
    # The encoder is just the input layer connected to the hidden layer. One the Autoencoder will be traine
    # the encoder to create our word vectors
    encoder = Model(input_word, encoded)

    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
    return encoder, autoencoder
```

In [ ]:
```
# create the model
encoder, autoencoder = creat_word2vec_model()
```

In [ ]:
```
# model summary
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 3458)]            0

 dense (Dense)               (None, 50)                172950

 dense_1 (Dense)             (None, 3458)              176358

=================================================================
Total params: 349,308
Trainable params: 349,308
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
# train the model by feeding it with our batch generator
autoencoder.fit_generator(batch_generator(batch_size, num_skips, skip_window, vocab_size), steps_per_epoch=
```

```
Epoch 1/2

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: UserWarning: `Model.fit_generator` is dep
recated and will be removed in a future version. Please use `Model.fit`, which supports generators.


7481/7481 [==============================] - 71s 9ms/step - loss: 0.0125
Epoch 2/2
7481/7481 [==============================] - 58s 8ms/step - loss: 0.0021
```

Out[23]: `<keras.callbacks.History at 0x7f258f2e3fd0>`

Use the encoder to vectorize

In [ ]:
```
# Create the Vectorizer function (prediciton of the encoder)
def vecotrize(word):
    word_one_hot = to_one_hot(word2id[word], vocab_size)
    return encoder.predict(np.array([word_one_hot]))[0]
```

In [ ]:
```
# Create the word2vec dictionary
word2vec = {w : vecotrize(w) for w in word2id.keys()}

# generate all words' vectorial representation.
```

In [ ]:
```
# save the word2vec dictionary
save_word2vec(our_word2vec)
```

visualization - find n closer words

In [ ]:
```
# for a given word, output the n closer words in the word2vec maping.
def get_n_closer(w, n, word2vec):
    vect = word2vec[w]
    distances_dict = {k: cosine(v, vect) for k, v in word2vec.items()}

    closer_words = []
    for _ in range(n):
        min_key = min(distances_dict.keys(), key=lambda k: distances_dict[k])
```

```
        closer_words.append(min_key)
        del distances_dict[min_key]
    return closer_words
```

In [ ]:
```
# check closer words for 'expense', 'derivatives', 'equity'
words_neighbors_1 = get_n_closer('expenses', 10, word2vec)
print('words close to expenses : ' +str(', '.join(words_neighbors_1)))
words_neighbors_2 = get_n_closer('derivatives', 10, word2vec)
print('words close to derivatives : ' +str(', '.join(words_neighbors_2)))
words_neighbors_3 = get_n_closer('equity', 20, word2vec)
print('words close to equity : ' +str(', '.join(words_neighbors_3)))
```

```
words close to expenses : expenses, annual, operating, total, fees, reflected, none, table, expenses1, de
scribes
words close to derivatives : derivatives, specialized, improvement, inadequate, impacting, fixed-rate, ta
x-exempt, software, able-a, collateralized
words close to equity : equity, hedged, global, small, international, mid, large, designed, methodology,
concentrated, descriptions, japan, terms, appear, bond, domestic, determined, weight, approximately, divi
dend
```

Create keywords lists and build knowledge base

In [ ]:
```
# keywords list for four strategies
balanced_kb = ['balance', 'balanced', 'asset','allocation','hybrid','basket','security','bond','equity','m

fixed_kb = ['fixed', 'long','principal','coupon','yield','bond','premium','income','interest','rate','deri

equity_kb = ['index', 'shareholder', 'stock', 'equity', 'asset','liability', 'growth', 'value' , 'capital',

longshort_kb = ['long','short','leverage', 'hedge', 'risk', 'short', 'margin', 'neutral', 'fee', 'spread',
```

In [ ]:
```
# Creates the knwoledge base by taking the num_neighbors closes neighbors of each key_words in word2vec
def create_knowledge_base(num_neighbors, word2vec, key_words):
    knowledge_base = set()
    out = display(progress(0, len(key_words)-1), display_id=True)
    for ii, key_word in enumerate(key_words) :
        knowledge_base.add(key_word)
        neighbors = []
        try :
            neighbors = get_n_closer(key_word, num_neighbors, word2vec)
        except :
            print(key_word + ' not in word2vec')

        knowledge_base.update(neighbors)

        out.update(progress(ii, len(key_words)-1))
    return knowledge_base
```

In [ ]:
```
# build knowledge base
keywords_list = [balanced_kb,fixed_kb,equity_kb,longshort_kb]

knowledge_base = []
for keywords in keywords_list:
  knowledge_base.append(create_knowledge_base(10, word2vec, keywords))
```

```
diversify not in word2vec
```

In [ ]:
```
# check knowledge base
print(knowledge_base[0])
```

```
{'denominated', 'balanced', 'small', 'resale', 'repayment', 'equity', 'exempt', '-are', 'competitive', 'w
arrant', 'allocation', 'loaned', 'expects', 'net', 'mortgage-related', 'concentrated', 'lender', 'curve',
'extent', 'combination', 'changes', 'rebalance', 'alternative', 'stream', 'assurance', 'largely', 'depth
', 'sheet', 'controlled', 'otc', 'dividends', 'fiscal', 'convertible', 'acute', 'impairment', 'formerly',
```

In [ ]:
```
# We create here the dataframe tha contains the summaries along with their labels
df_extraction = pd.DataFrame({'fund_name' : fund_names, 'summary':summaries})
df_label = pd.read_csv(SUMMARY_LABELS_PATH)
df = df_label.merge(df_extraction, on='fund_name', how='left').dropna()
df.head()
```

Out[ ]:

	id	fund_name	Performance fee?	Ivestment Strategy	Leverage?	Portfolio composition	Concentration	summary
0	0000051931-18-000151	American Funds College 2018 Fund	None	Balanced Fund (Low Risk)	Yes	Investment grade securities	Diversified	American Funds College 2018 Fund\n \nInvestment...
1	0000051931-18-000151	American Funds College 2021 Fund	None	Balanced Fund (Low Risk)	Yes	Investment grade securities	Diversified	American Funds College 2021 Fund\n \nInvestment...
2	0000051931-18-000151	American Funds College 2024 Fund	None	Balanced Fund (Low Risk)	Yes	Investment grade securities	Diversified	American Funds College 2024 Fund\n \nInvestment...
3	0000051931-18-000151	American Funds College 2027 Fund	None	Balanced Fund (Low Risk)	Yes	Investment grade securities	Diversified	American Funds College 2027 Fund\n \nInvestment...
4	0000051931-18-000151	American Funds College 2030 Fund	None	Balanced Fund (Low Risk)	Yes	Investment grade securities	Diversified	American Funds College 2030 Fund\n \nInvestment...

In [ ]:
```
# remove 'Commodities Fund (Low Risk)' because where is only one such fund, no need to train
df = df[df['Ivestment Strategy'] != 'Commodities Fund (Low Risk)']
```

Match extraction

In [ ]:
```
# find the most related sentense comparing to the knowledge base, use the scoring function to calcualte the
def extract_sentence_match(summary, knowledge, num_sent):
    sentences = sent_tokenize(summary)
    sentence_scores = []
    for j, sentence in enumerate(sentences):
        set_tokens = set(tokenizer(sentence))

        # Find the number of common words between the knowledge base and the sentence
        inter_knwoledge = set_tokens.intersection(knowledge)

        sentence_scores.append(len(inter_knwoledge))

    sentence_scores, sentences = zip(*sorted(zip(sentence_scores, sentences)))
    top_sentences = sentences[len(sentences)-num_sent-1:]
    return np.mean(sentence_scores)
```

In [ ]:
```
# add all sentense score as numerical value to the dataframe created before
df['balance_sentences_match'] = df.apply(lambda x : extract_sentence_match(x['summary'], knowledge_base[0],
df['fixed_sentences_match'] = df.apply(lambda x : extract_sentence_match(x['summary'], knowledge_base[1], r
df['equity_sentences_match'] = df.apply(lambda x : extract_sentence_match(x['summary'], knowledge_base[2],
df['longshort_sentences_match'] = df.apply(lambda x : extract_sentence_match(x['summary'], knowledge_base[3
```

In [ ]:
```
# export dataframe out as csv document for convenience
df.to_csv('/content/drive/MyDrive/728, 793, 703, 815/df.csv')
```

In [ ]:
```
# Standardize function
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
      self.attribute_names = attribute_names
  def fit(self, X, y=None):
      return self
  def transform(self, X):
      return X[self.attribute_names]

num_pipeline = Pipeline([
  ('normalize', StandardScaler())
])
```

Classification Algorithm

Logistic regression

In [ ]:
```
# set four columns as X and change strategies into number 0,1,2,3
X = df[['balance_sentences_match','fixed_sentences_match','equity_sentences_match', 'longshort_sentences_ma
df['Ivestment Strategy code'] = df['Ivestment Strategy'].astype('category').cat.codes
y = df['Ivestment Strategy code'].values
y
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 1, 2, 0, 2, 0, 2, 1, 1, 1, 1, 1, 2,
       1, 1, 0, 1, 1, 1, 1, 2, 0, 1, 1, 0, 1, 1, 2, 2, 0, 0, 2, 0, 2, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2,
       1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1,
       1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 1, 0, 1, 1, 1, 2, 2, 2, 2,
       2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 0, 1, 2, 1,
       2, 2, 1, 1, 1, 0, 1, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 2, 1, 1, 2, 0, 1, 1, 1, 2, 2, 1, 1, 2, 1,
       1, 1, 1, 1, 0, 2, 2, 2, 2, 1, 2, 2, 1, 3, 1, 1, 1, 2, 1, 1, 0, 2,
       1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1,
       1, 1, 1, 2, 2, 2, 2, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

In [ ]:
```
# split train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state=2)
```

In [ ]:
```
# Standardize features
X_train = num_pipeline.fit_transform(X_train)
X_test = num_pipeline.fit_transform(X_test)
```

In [ ]:
```
# run logistic regression using 'libliear' to work with multiclass problem
lm = LogisticRegression(multi_class='ovr', solver='liblinear')
lm.fit(X_train, y_train)
```

Out[47]:
```
LogisticRegression(multi_class='ovr', solver='liblinear')
```

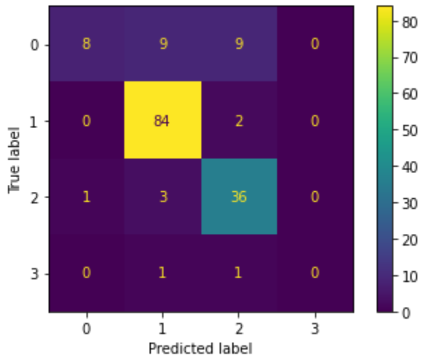
In [ ]:
```
# check prediction accuracy
lm.score(X_test, y_test)
```

Out[48]:
```
0.8311688311688312
```

In [ ]:
```
# check confusionmatrix
disp = metrics.plot_confusion_matrix(lm, X_test, y_test)
disp.confusion_matrix
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_conf
usion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in
1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from
_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[49]:
```
array([[ 8,  9,  9,  0],
       [ 0, 84,  2,  0],
       [ 1,  3, 36,  0],
       [ 0,  1,  1,  0]])
```



DNN

In [ ]:
```
keras.backend.clear_session()
```

In [ ]:
```
#set random seed
np.random.seed(42)
tf.random.set_seed(42)
```

In [ ]:
```
# build DNN model
def build_model(n_hidden=3, n_neurons= 30, input_shape=[4],learning_rate=3e-3, drop = 0.5):
  model = keras.models.Sequential()
  model.add(keras.layers.InputLayer(input_shape=input_shape))
  for i in range(n_hidden):
    model.add(keras.layers.Dense(n_neurons ,activation='relu'))
    model.add(Dropout(drop))
  model.add(keras.layers.Dense(4, activation="sigmoid"))
  optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
  model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer, metrics =['accuracy'])
  return model
```

In [ ]:
```
# set callbacks rule
callbacks = [EarlyStopping(monitor='val_loss', patience=10)]
```

In [ ]:
```
# test DNN prediction accuray
keras_cl = keras.wrappers.scikit_learn.KerasClassifier(build_model)
keras_cl.fit(X_train, y_train, epochs=100,
              validation_data=(X_test, y_test),
              callbacks=callbacks,
              batch_size=16)
```

```
Epoch 1/100

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: KerasClassifier is de
precated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. See https://www.adriangb.com/scik
eras/stable/migration.html (https://www.adriangb.com/scikeras/stable/migration.html) for help migrating.
  """Entry point for launching an IPython kernel.
```

```
20/20 [==============================] - 1s 23ms/step - loss: 1.4219 - accuracy: 0.3280 - val_loss: 1.223
7 - val_accuracy: 0.5584
Epoch 2/100
```

In [ ]:
```
# set parameters
param_distribs = {
  "n_hidden": [1, 2, 3, 4],
  "n_neurons": np.arange(10, 60),
  "learning_rate": reciprocal(3e-4, 3e-2),
  'drop': uniform(0.2, 0.5)
}
```

In [ ]:
```
# parameters tuning
rnd_search_cv = RandomizedSearchCV(keras_cl, param_distribs, n_iter=30, cv=3, verbose=2)
rnd_search_cv.fit(X_train, y_train, epochs=40, validation_data=(X_test, y_test),callbacks=callbacks, batch_
```

```
Streaming output truncated to the last 5000 lines.
13/13 [==============================] - 0s 7ms/step - loss: 0.3515 - accuracy: 0.8696 - val_loss: 0.5263
- val_accuracy: 0.8831
4/4 [==============================] - 0s 6ms/step - loss: 0.4528 - accuracy: 0.8750
[CV] END drop=0.22333283160680772, learning_rate=0.026584732357599207, n_hidden=3, n_neurons=56; total ti
me=   2.9s
Epoch 1/40
13/13 [==============================] - 1s 24ms/step - loss: 0.9097 - accuracy: 0.6425 - val_loss: 0.696
2 - val_accuracy: 0.8247
Epoch 2/40
13/13 [==============================] - 0s 7ms/step - loss: 0.6769 - accuracy: 0.8261 - val_loss: 0.5667
- val_accuracy: 0.8052
Epoch 3/40
13/13 [==============================] - 0s 8ms/step - loss: 0.6282 - accuracy: 0.8068 - val_loss: 0.4899
- val_accuracy: 0.8636
Epoch 4/40
13/13 [==============================] - 0s 7ms/step - loss: 0.5650 - accuracy: 0.8309 - val_loss: 0.4709
- val_accuracy: 0.8636
Epoch 5/40
```

In [ ]:
```
# check best DNN parameters
rnd_search_cv.best_params_
```

Out[59]:
```
{'drop': 0.2115312125207079,
 'learning_rate': 0.0033625641252688094,
 'n_hidden': 3,
 'n_neurons': 51}
```

In [ ]:
```
# predict using best parameters
DNN_model = rnd_search_cv.best_estimator_.model
y_prob_DNN = DNN_model.predict(X_test)
```

In [ ]:
```
# check DNN model summary with best parameters
DNN_model.summary()
```

```
Model: "sequential_91"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_331 (Dense)           (None, 51)                255

 dropout_240 (Dropout)       (None, 51)                0

 dense_332 (Dense)           (None, 51)                2652

 dropout_241 (Dropout)       (None, 51)                0

 dense_333 (Dense)           (None, 51)                2652

 dropout_242 (Dropout)       (None, 51)                0

 dense_334 (Dense)           (None, 4)                 208

=================================================================
```

In [ ]:
```
# change DNN probability into categorial lable
y_pred_DNN = np.array([list(x).index(max(x)) for x in y_prob_DNN])
y_pred_DNN
```

Out[62]:

```
array([0, 2, 1, 1, 1, 1, 2, 1, 0, 0, 1, 2, 1, 1, 1, 0, 1, 0, 1, 2, 0, 1,
       1, 1, 1, 2, 2, 1, 2, 1, 1, 0, 1, 1, 1, 2, 0, 1, 1, 2, 1, 1, 1, 2,
       1, 1, 1, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 0, 2, 2, 1, 2, 1, 0, 2,
```

In [ ]:
```
# check accuracy comparing to actual labels
np.mean(y_pred_DNN == y_test)
```

Out[63]:
```
0.9025974025974026
```

In [ ]:
```
# build confusion matrix
confusion_matrix(y_test, y_pred_DNN)
```

Out[64]:
```
array([[18,  6,  2,  0],
       [ 1, 83,  2,  0],
       [ 1,  1, 38,  0],
       [ 0,  1,  1,  0]])
```

SVC

In [ ]:
```
# run additional SVC model to compare results
svm_clf = SVC(gamma="scale", probability=True)
svm_clf.fit(X_train, y_train)
```

Out[141]:
```
SVC(probability=True)
```

In [ ]:
```
param_distributions = {"gamma": reciprocal(0.001, 0.1), "C": uniform(1, 10), 'degree': [1, 2, 3]}
rnd_search_cv_svm = RandomizedSearchCV(svm_clf, param_distributions, n_iter=30, verbose=2, cv=3)
rnd_search_cv_svm.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
[CV] END C=8.70967179954561, degree=3, gamma=0.0012046674587990317; total time=   0.0s
[CV] END C=8.70967179954561, degree=3, gamma=0.0012046674587990317; total time=   0.0s
[CV] END C=8.70967179954561, degree=3, gamma=0.0012046674587990317; total time=   0.0s
[CV] END C=8.106628896857874, degree=2, gamma=0.0011241862095793058; total time=   0.0s
[CV] END C=8.106628896857874, degree=2, gamma=0.0011241862095793058; total time=   0.0s
[CV] END C=8.106628896857874, degree=2, gamma=0.0011241862095793058; total time=   0.0s
[CV] END C=2.0789142699330445, degree=3, gamma=0.061876706758809484; total time=   0.0s
[CV] END C=2.0789142699330445, degree=3, gamma=0.061876706758809484; total time=   0.0s
[CV] END C=2.0789142699330445, degree=3, gamma=0.061876706758809484; total time=   0.0s
[CV] END C=5.7537022318211175, degree=1, gamma=0.01040258761588384; total time=   0.0s
[CV] END C=5.7537022318211175, degree=1, gamma=0.01040258761588384; total time=   0.0s
[CV] END C=5.7537022318211175, degree=1, gamma=0.01040258761588384; total time=   0.0s
[CV] END C=10.07566473926093, degree=3, gamma=0.016174645036343024; total time=   0.0s
[CV] END C=10.07566473926093, degree=3, gamma=0.016174645036343024; total time=   0.0s
[CV] END C=10.07566473926093, degree=3, gamma=0.016174645036343024; total time=   0.0s
[CV] END C=6.398410913016731, degree=1, gamma=0.002868113482103007; total time=   0.0s
[CV] END C=6.398410913016731, degree=1, gamma=0.002868113482103007; total time=   0.0s

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The least popu
```

In [ ]:
```
rnd_search_cv_svm.best_params_
```

Out[63]:
```
{'C': 3.5178229582536416, 'degree': 2, 'gamma': 0.02657915561441958}
```

In [ ]:
```
svm_mod = rnd_search_cv_svm.best_estimator_.fit(X_train, y_train)
y_pred_svm = svm_mod.predict(X_test)
```

In [ ]:
```
np.mean(y_pred_svm == y_test)
```

Out[65]:
```
0.8311688311688312
```

Outlier Detection

##Local Outlier Factor

In [ ]:
```
# check true outlier (longshort strategy fund )
y_lof = np.array([-1 if x == 3 else 1 for x in y_test])
```

In [ ]:
```
y_lof
```

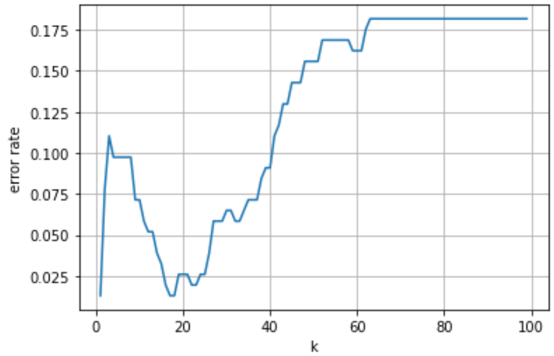
...

In [ ]:
```
# LOF function
error_rate = []
for i in range(1, 100):
```

```
    clf = LocalOutlierFactor(n_neighbors=i, novelty=True)
    clf.fit(X.values)
    lof_pred = clf.predict(X_test)
    error_rate.append(1 - np.mean(lof_pred == y_lof))

plt.plot(range(1, 100), error_rate)
plt.ylabel('error rate')
plt.xlabel('k')
plt.grid()
plt.show()

best_n_neighbor = error_rate.index(min(error_rate)) + 1
best_n_neighbor
```



Out[82]: 1

In [ ]:
```
lof_pred
```

Out[70]:
```
array([ 1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1, -1,  1,  1, -1,  1,
        1, -1,  1,  1,  1, -1,  1,  1,  1, -1,  1,  1,  1,  1, -1,  1,  1,
        1,  1, -1, -1,  1,  1,  1,  1,  1,  1,  1, -1,  1, -1,  1,  1,  1,
        1,  1,  1,  1, -1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1, -1,  1,  1,  1,  1,  1,  1,  1, -1,
        1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1, -1,  1, -1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1, -1, -1,  1,  1,  1,  1, -1,  1,  1,  1,
        1,  1,  1,  1, -1,  1, -1,  1,  1,  1, -1,  1, -1,  1,  1,  1,  1,
        1])
```

In [ ]:
```
# predict outliers
clf = LocalOutlierFactor(n_neighbors=1)
lof_pred = clf.fit_predict(X_test.values)
```

In [ ]:
```
# check model precision
print(confusion_matrix(y_lof, lof_pred))
print('Model Precision =', (confusion_matrix(y_lof, lof_pred)[0][0] + confusion_matrix(y_lof, lof_pred)[1]
```

...

In [ ]:
```
# check all predicted outliers' index
index = np.where(lof_pred == -1)[0]
index
```

...

In [ ]:
```
# check true outliers' index
np.where(y_lof == -1)[0]
```

Out[90]: array([62, 126])

In [ ]:
```
# set threshold if the model miss count inliers as outliers, prediction correction
error_threshold=[]
for prob in np.linspace(0.8, 0.99, 50):
    final_pred = []

    for i, pred in enumerate(y_pred_DNN):
        if lof_pred[i] == 1:
            final_pred.append(pred)
        else:
            if max(y_prob_DNN[i]) >= prob:
                final_pred.append(pred)
            else:
                final_pred.append(3)

    error_threshold.append(1 - np.mean(np.array(final_pred) == y_test))
```

```
plt.plot(np.linspace(0.8, 0.99, 50), error_threshold)
plt.ylabel('error rate')
plt.xlabel('threshold')
plt.grid()
plt.show()

best_threshold = 0.901
```

...

In [ ]:
```
# set LOF outlier dection model using the threshold set above
final_pred = []

for i, pred in enumerate(y_pred_DNN):
    if lof_pred[i] == 1:
      final_pred.append(pred)
    else:
      if max(y_prob_DNN[i]) >= best_threshold:
        final_pred.append(pred)
      else:
        final_pred.append(3)

print(np.mean(np.array(final_pred) == y_test))
print(sum(np.array(final_pred) == 3))
```

```
0.8831168831168831
8
```

In [ ]:
```
y_prob_DNN[[62, 126]]
```

Out[101]:
```
array([[0.3972624 , 0.42612275, 0.7191548 , 0.05769338],
       [0.87252146, 0.90092325, 0.01546596, 0.4136835 ]], dtype=float32)
```

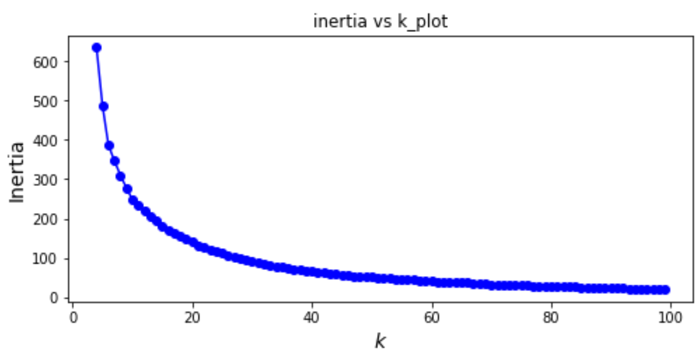
In [ ]:
```
# check classification result after applying the outlier detection
confusion_matrix(y_test, final_pred)
```

Out[102]:
```
array([[16,  4,  2,  4],
       [ 1, 83,  2,  0],
       [ 1,  0, 36,  3],
       [ 0,  0,  1,  1]])
```

outlier detection using KNN

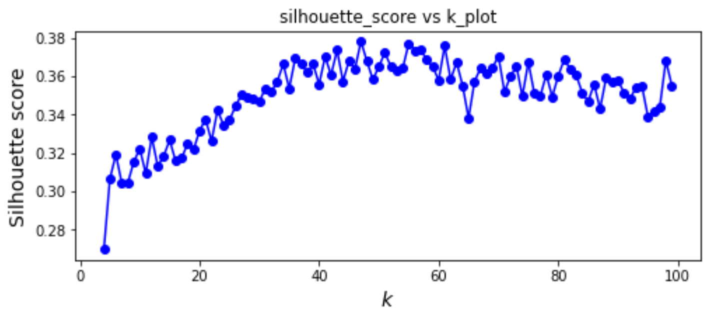
In [ ]:
```
# set k range
kmeans_per_k = [KMeans(algorithm="elkan", n_clusters=k, random_state=42).fit(X_train)for k in range(4, 100)
```

In [ ]:
```
# test k from 4 to 100 and plot inertia
inertias = [model.inertia_ for model in kmeans_per_k]
plt.figure(figsize=(8, 3.5))
plt.plot(range(4, 100), inertias, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inertia", fontsize=14)
plt.title("inertia vs k_plot")
plt.show()
```



In [ ]:
```
# test k from 4 to 100 and calculated the Silhouette Score
silhouette_scores = [silhouette_score(X_train, model.labels_) for model in kmeans_per_k]
plt.figure(figsize=(8, 3))
plt.plot(range(4, 100), silhouette_scores, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Silhouette score", fontsize=14)
```

```
plt.title("silhouette_score vs k_plot")
plt.show()
```



In [ ]:
```
# find k with best score
max_silhouette_score = max(silhouette_scores)
best_index = silhouette_scores.index(max_silhouette_score)
best_k = (best_index + 4)
print('K with the best score :', best_k)
```

```
K with the best score : 47
```

In [ ]:
```
np.where(y_train == 3)[0]
```

Out[116]:
```
array([ 36, 306])
```

In [ ]:
```
# run kmeans outlier dection
kmeans = KMeans(n_clusters=best_k, random_state=0).fit(X_train)
kmeans.predict(X_train[np.where(y_train == 3)[0]])
```

Out[118]:
```
array([43, 19], dtype=int32)
```

In [ ]:
```
# check prediction results
kmean_pred = np.array([-1 if kmeans.predict(X_test)[i] == 36 or kmeans.predict(X_test)[i] == 65 else 1 for
print(kmean_pred)
```

```
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1 -1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 -1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1 -1  1  1  1  1  1 -1]
```

In [ ]:
```
# check prediction accuracy and draw confusion matrix
confusion_matrix(y_lof, kmean_pred)
```

Out[120]:
```
array([[  0,   2],
       [  6, 146]])
```