Homework 2

In [ ]:
```
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score, confusion_matrix, classification_re
from sklearn.mixture import GaussianMixture
from scipy.stats import norm

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pydotplus
import seaborn as sns
```

Load Data

The feature space is already lagged. Therefore, you do not need to lag the variables yourself. The data track these companies over 3 years (2018-2020). We will train the data in 2018, validate in 2019, and forecast 2020. Further instructions are given in the questions.

In [ ]:
```
dat = pd.read_csv('hw2_data.csv')
```

In [ ]:
```
dat
```

Out[ ]:

	permno	fyear	label	ret	acc	agr	cfp	ep	
0	15417	2018.0	0	0.102609	0.053313	0.252472	0.253973	0.148027	0.09511
1	89031	2018.0	1	0.067922	-0.001165	0.215518	0.005945	0.002326	0.01225
2	11154	2018.0	0	-0.071429	0.000948	0.032913	0.184267	0.024266	0.00857
3	13556	2018.0	0	-0.326014	0.045752	-0.095131	-1.258587	-1.340418	1.52723
4	14296	2018.0	0	0.249443	0.016353	-0.020653	0.519260	0.449600	0.85453
...	...	...	...	...	...	...	...	...	.
2238	14532	2020.0	-1	-0.080498	-0.064369	-0.050658	0.064593	0.028561	0.00000
2239	41355	2020.0	0	0.015946	0.043579	0.147297	0.082415	0.064142	-0.03016
2240	13983	2020.0	0	0.112274	0.003504	-0.219388	-0.345708	-0.429181	0.00465
2241	47619	2020.0	0	-0.057480	0.007940	-0.033335	-0.010621	-0.123947	0.00000
2242	80362	2020.0	0	0.057283	0.014109	0.062698	0.115561	0.070647	-0.00725

2243 rows × 56 columns

In [ ]:
```
dat.shape
```

Out[ ]: (2243, 56)

Generative Learning

1. Keep only the labels between -1 and 3.

Split the data into Train-Validation-Test:

- Training data should contain features in 2018, do not forget to remove ‘label’
- Training labels should only contain ‘label’ in 2018
- Validation data should contain features in 2019, do not forget to remove ‘label’ • Validation labels should only contain ‘label’ in 2019
- Test data should contain features in 2020, do not forget to remove ‘label’
- Test labels should only contain ‘label’ in 2020

In [ ]:
```
# filter label between -1 and 3
dat_1 = dat.loc[(dat['label'] >= -1) & (dat['label'] < 3)]

# Training data
train_feature = dat_1.loc[dat_1['fyear'] == 2018]
train_label = train_feature['label']
train_feature = train_feature.drop(columns=['label'])

# validation data
valid_feature = dat_1.loc[dat_1['fyear'] == 2019]
valid_label = valid_feature['label']
valid_feature = valid_feature.drop(columns=['label'])

# test data
test_feature = dat_1.loc[dat_1['fyear'] == 2020]
test_label = test_feature['label']
test_feature = test_feature.drop(columns=['label'])
```

2. Compute and report the prior probabilities π_j for all labels in the Training set.

In [ ]:
```
all_label = pd.DataFrame(train_label).groupby('label').size()
all_label = pd.DataFrame(all_label, columns=['count'], index=all_label.index)
all_label['prior'] = all_label['count'] / sum(all_label['count'])

pi = all_label['prior']
pi
```

Out[ ]:
```
label
-1     0.070159
 0     0.820393
 1     0.088868
 2     0.020580
Name: prior, dtype: float64
```

3. Using the Training set, calculate the likelihood for feature ‘ret’ to be 0.1 conditional on each value of the label $P_j = P(ret = 0.1|y = j)$ (Use the Normal PDF found in the following link: Scipy). Report the likelihood for each

value of the label. You need to code this by hand, 0 points will be given if you use the pre-coded scipy function.

In [ ]:
```
train = dat_1.loc[dat_1['fyear'] == 2018]
ret_mean = train.groupby(['label']).mean()['ret']
ret_var = train.groupby(['label']).var()['ret']

P = pd.DataFrame()
P['mean'] = ret_mean
P['var'] = ret_var
P['0.1_likelihood'] = norm.pdf((0.1 - P['mean']) / np.sqrt(P['var']))
P
```

Out[ ]:

label	mean	var	0.1_likelihood
-1	0.014262	0.013064	0.301106
0	0.010048	0.010146	0.267751
1	0.027497	0.012392	0.322697
2	-0.010655	0.008859	0.199889

4. Use Guassian naive bayes from the scikit-learn library (found here:scikitlearnfunction) to classify the test data. Report the accuracy. You need to use the train+validation set.

In [ ]:
```
gnb = GaussianNB()
y_pred = gnb.fit(
    pd.concat([train_feature, valid_feature], ignore_index=True),
    pd.concat([train_label, valid_label], ignore_index=True)
    ).predict(test_feature)

test_accuracy = accuracy_score(test_label, y_pred)

print(f'test accuracy is {round(test_accuracy, ndigits=4)}')
```

```
test accuracy is 0.2887
```

5. Compute the confusion matrix (as shown in the lectures) and report the top 2 pairs with most (absolute number) incorrect classifications.

In [ ]:
```
mat = confusion_matrix(test_label, y_pred, labels=[i for i in range(-1, 3)])
mat
```

Out[ ]:
```
array([[ 1,  0,  0,  7],
       [ 1, 22,  2, 52],
       [ 0,  0,  1,  7],
       [ 0,  0,  0,  4]])
```

In [ ]:
```
mat = confusion_matrix(test_label, y_pred, labels=[i for i in range(-1, 3)])

def find_n_max(matrix, n):

    pairs = []

    for i in range(n):
```

```
        max_incorrect = 0
        max_row = 0
        max_col = 0

        for row in range(len(matrix)):

            for column in range(len(matrix[0])):

                val = matrix[row, column]

                if (val > max_incorrect) & (row != column):

                    max_incorrect = val
                    max_row = row
                    max_col = column

        matrix[max_row, max_col] = 0 # change the biggest value to 0 and loop a
        pairs.append([max_row, max_col]) # row and column

    return pairs

find_n_max(mat, 2)
```

Out[ ]:
```
[[1, 3], [0, 3]]
```

Based on the data given by `pairs`, we know that the top 2 pairs with most incorrect are [2, 0] and [2, -1].

6. Implement Gaussian Mixture model on the data as shown in class.

Tune the covariance type parameter on the validation data.

Use the selected value to compute the test accuracy. As always, train the model on train+validation data to compute the test accuracy. Train the model twice, the first model should use the covariance type that yielded the highest accuracy in the validation stage. The second model should use the covariance type that yielded the second highest accuracy in the validation stage.

Comment on the accuracy on the test set of the the models you ran.

Hint: Use ‘n components=3, init params=”kmeans”, random state=34’.

In [ ]:
```
for cov in ['full', 'tied', 'diag', 'spherical']:

    clf = GaussianMixture(n_components=4, covariance_type=cov, init_params='kme
    clf.means_init = np.array([train_feature[train_label == i].mean(axis=0) for
    clf.fit(train_feature, train_label)
    pred = clf.predict(valid_feature)

    print ('Validation accuracy for covariance type '+ cov + ' = ' + str(accura
```

```
Validation accuracy for covariance type full = 0.6842105263157895
Validation accuracy for covariance type tied = 0.09952153110047847
Validation accuracy for covariance type diag = 0.02966507177033493
Validation accuracy for covariance type spherical = 0.1799043062200957
```

Based on the above results, we can see that the first model is `full` and the second is `tied`.

```
In [ ]: X = pd.concat([train_feature, valid_feature], ignore_index=True)
        y = pd.concat([train_label, valid_label], ignore_index=True)

        # first model
        clf1 = GaussianMixture(n_components=4, covariance_type='full', init_params='kme
        clf1.means_init = np.array([X[y == i].mean(axis=0) for i in range(-1, 3)])
        clf1.fit(X, y)
        pred1 = clf1.predict(test_feature)

        print(f'Test accuracy for covariance type full is {accuracy_score(test_label, p

        # second model
        clf2 = GaussianMixture(n_components=4, covariance_type='tied', init_params='kme
        clf2.means_init = np.array([X[y == i].mean(axis=0) for i in range(-1, 3)])
        clf2.fit(X, y)
        pred2 = clf2.predict(test_feature)

        print(f'Test accuracy for covariance type tied is {accuracy_score(test_label, p
```

```
Test accuracy for covariance type full is 0.1958762886597938
Test accuracy for covariance type tied is 0.08247422680412371
```

As we can see, the test accuracy decreases sharply after we rerun the models using train+vaild data.

We should be awared that the dataset was separated into three sets with same length. And there can be some similar pattern within this year thus causing overfit.

After we use two-year data as the training data, the model could perform very bad.

7. Bonus Question: Apply Linear Discriminant Analysis model on the train+validation data and report the accuracy obtained on test data. Report the transformation matrix (w) along with the intercept.

```
In [ ]: clf = LinearDiscriminantAnalysis(solver='eigen')
        clf.fit(X, y)
        y_pred = clf.predict(test_feature)
        print(f'Test accuracy for discriminant analysis is {accuracy_score(test_label,
```

```
Test accuracy for discriminant analysis is 0.7731958762886598
```

```
In [ ]: clf.coef_
```

```
Out[ ]: array([[ 2.26680976e-03,  8.39045315e+03, -7.07258418e+02,
                 8.99626216e+00,  6.34848515e+02, -5.23096410e+03,
                 4.63106899e+03,  2.60911958e+02, -8.26841552e+01,
                -5.85193945e+02,  2.09892141e+01, -1.27152966e+01,
                -3.46604694e+02, -1.22320953e-02,  7.57087533e+01,
                 1.96856917e+01, -4.39056956e+01,  8.86628905e+02,
                -1.54733704e+02, -3.38125142e+01,  5.89892746e+03,
                -4.74527389e+01, -3.84712575e+01,  8.69048746e+02,
                 6.35946546e+02,  6.27384783e-05,  2.60933259e+01,
                 1.58321785e+02, -2.62289300e+03,  1.22177948e+03,
                 2.67774506e+03,  1.54828547e+01, -2.93670725e+02,
                 1.34123417e+01,  9.83080979e+01, -4.71810209e+01,
                 8.04493418e+00, -1.54480307e+01, -1.51340882e+00,
                -5.76851123e+01,  9.60669302e+01,  4.19385694e-02,
                -1.76722372e-01,  2.24674843e-01, -1.78178848e+02,
                 2.39731427e+02, -2.88929038e+01,  1.01686606e+02,
                -4.34962272e+03,  1.71502384e+02, -9.82155462e+02,
                -1.72892547e+03, -1.44038304e+02,  4.61419076e+02,
                 1.35381904e+00],
               [ 2.26470177e-03,  8.39039905e+03, -7.08155718e+02,
                 9.12159682e+00,  6.37535459e+02, -5.22457640e+03,
                 4.62385274e+03,  2.55339064e+02, -8.26496316e+01,
                -5.85341052e+02,  2.24144863e+01, -1.21442857e+01,
                -3.44820130e+02, -1.23526662e-02,  7.49249277e+01,
                 1.92538463e+01, -4.38863708e+01,  8.86254648e+02,
                -1.55284723e+02, -3.48363727e+01,  5.99390596e+03,
                -4.71853246e+01, -3.87006186e+01,  8.69657868e+02,
                 6.35508612e+02,  6.64142363e-05,  2.37444824e+01,
                 1.58446205e+02, -2.62890432e+03,  1.22496355e+03,
                 2.68252693e+03,  1.55070951e+01, -2.93822163e+02,
                 1.34143634e+01,  9.80224708e+01, -4.71569934e+01,
                 8.01399580e+00, -1.54597264e+01, -1.53794065e+00,
                -5.76272309e+01,  9.55338551e+01,  4.18169867e-02,
                -1.77210213e-01,  2.24604501e-01, -1.78213021e+02,
                 2.40622446e+02, -2.93823630e+01,  1.02030186e+02,
                -4.34529817e+03,  1.71341441e+02, -9.76307688e+02,
                -1.72937856e+03, -1.43449268e+02,  4.60328117e+02,
                 1.38225554e+00],
               [ 2.26579515e-03,  8.39041959e+03, -7.09284437e+02,
                 9.21092301e+00,  6.37409364e+02, -5.22345870e+03,
                 4.62197892e+03,  2.55663892e+02, -8.25039767e+01,
                -5.86249688e+02,  2.51513373e+01, -1.18854670e+01,
                -3.45139928e+02, -1.23765020e-02,  7.50844460e+01,
                 1.93565879e+01, -4.39775992e+01,  8.86691277e+02,
                -1.55030243e+02, -3.49329437e+01,  5.98817263e+03,
                -4.70089835e+01, -3.86534281e+01,  8.70597493e+02,
                 6.32100833e+02,  6.77636484e-05,  2.44234659e+01,
                 1.59686988e+02, -2.63011049e+03,  1.22466677e+03,
                 2.68012812e+03,  1.55141466e+01, -2.93816228e+02,
                 1.34156139e+01,  9.70756624e+01, -4.71998775e+01,
                 8.01689852e+00, -1.54654066e+01, -1.40848621e+00,
                -5.71635076e+01,  9.49628853e+01,  4.18540721e-02,
                -1.74210839e-01,  2.25047850e-01, -1.78337564e+02,
                 2.38748449e+02, -2.96503328e+01,  1.01802950e+02,
                -4.34372703e+03,  1.71491794e+02, -9.77041614e+02,
                -1.72984914e+03, -1.43125469e+02,  4.59951531e+02,
                 1.40268124e+00],
               [ 2.27023878e-03,  8.39033660e+03, -7.10704452e+02,
                 8.86071965e+00,  6.37616089e+02, -5.20840913e+03,
                 4.60815595e+03,  2.55147951e+02, -8.23599581e+01,
```

```
        -5.84728887e+02,  2.95834284e+01, -1.12828216e+01,
        -3.44749687e+02, -1.23544749e-02,  7.44875998e+01,
         1.85700630e+01, -4.37941175e+01,  8.86613384e+02,
        -1.55333856e+02, -3.46327603e+01,  5.96561976e+03,
        -4.71693891e+01, -3.87306546e+01,  8.66780199e+02,
         6.39897378e+02,  5.98242476e-05,  2.44218150e+01,
         1.59587712e+02, -2.62454200e+03,  1.22497486e+03,
         2.67169617e+03,  1.55417855e+01, -2.93906671e+02,
         1.34119682e+01,  1.00146818e+02, -4.72690012e+01,
         8.00586406e+00, -1.54599860e+01, -1.42345994e+00,
        -5.71134686e+01,  9.48546653e+01,  4.17884551e-02,
        -1.67408717e-01,  2.24731828e-01, -1.78351039e+02,
         2.38898090e+02, -3.13627318e+01,  1.02050283e+02,
        -4.34711448e+03,  1.71355642e+02, -9.77358926e+02,
        -1.72981993e+03, -1.43565851e+02,  4.59603465e+02,
         1.36281267e+00]])
```

In [ ]:

```
clf.intercept_
```

Out[ ]:

```
array([-8468247.51331814, -8468133.5827645 , -8468180.1162174 ,
       -8468016.42805395])
```

In [ ]:

```
# simple computation of mean for the features in each class
mean_vectors = []
for cl in range(-1, 3):

    mean_vectors.append(np.mean(X[y == cl], axis=0))
```

In [ ]:

```
# within class scatter matrix S_W

S_W = np.zeros((53, 53)) # within class and between class
for cl, mv in zip(range(-1, 3), mean_vectors):

    class_sc_mat = np.zeros((53, 53))                  # scatter matrix for eve

    for row in np.array(X[y == cl]):

        row, mv = row.reshape(53, 1), np.array(mv).reshape(53, 1) # make column
        class_sc_mat += (row - mv).dot((row - mv).T)

    S_W += class_sc_mat                               # sum class scatter matrice

print('within-class Scatter Matrix:\n', S_W)
```

```
within-class Scatter Matrix:
 [[ 3.37090280e+01  2.26961249e+00  2.08486945e+00 ... -2.87292510e+00
  -3.06798859e+01 -1.92177013e+01]
 [ 2.26961249e+00  8.20671455e+02  7.08876880e+00 ... -1.85227247e+00
   3.62100656e+01 -6.39508675e+01]
 [ 2.08486945e+00  7.08876880e+00  2.56472760e+02 ... -4.29710349e+00
  -4.74012464e+01 -9.26123489e+01]
 ...
 [-2.87292510e+00 -1.85227247e+00 -4.29710349e+00 ...  1.62216372e+02
   2.37906197e+01 -1.36474141e+02]
 [-3.06798859e+01  3.62100656e+01 -4.74012464e+01 ...  2.37906197e+01
   1.46418988e+03 -1.07294057e+03]
 [-1.92177013e+01 -6.39508675e+01 -9.26123489e+01 ... -1.36474141e+02
  -1.07294057e+03  3.90703411e+04]]
```

In [ ]:

```
# between-class scatter matrix S_B
```

```python
overall_mean = np.mean(X, axis=0)

S_B = np.zeros((53, 53))
for i, mean_vec in enumerate(mean_vectors):

    n = np.array(X[y == i - 1]).shape[0]
    mean_vec = np.array(mean_vec).reshape(53, 1) # make column vector
    overall_mean = np.array(overall_mean).reshape(53, 1) # make column vector
    S_B += n * (mean_vec - overall_mean).dot((mean_vec - overall_mean).T)

print('between-class Scatter Matrix:\n', S_B)
```

```
between-class Scatter Matrix:
 [[ 1.60678891e-01  1.08073875e-01  1.04489939e-01 ... -9.76099627e-02
   1.47898062e+00  1.08505095e+00]
 [ 1.08073875e-01  1.18967756e-01  1.72230009e-01 ...  1.27061446e-01
  -1.03087175e+00  3.22165415e+00]
 [ 1.04489939e-01  1.72230009e-01  4.44357839e-01 ...  4.78574905e-01
  -4.21202614e+00  7.76959298e+00]
 ...
 [-9.76099627e-02  1.27061446e-01  4.78574905e-01 ...  9.52773411e-01
  -9.88454415e+00  1.09363717e+01]
 [ 1.47898062e+00 -1.03087175e+00 -4.21202614e+00 ... -9.88454415e+00
   1.05613427e+02 -1.06462894e+02]
 [ 1.08505095e+00  3.22165415e+00  7.76959298e+00 ...  1.09363717e+01
  -1.06462894e+02  1.57831711e+02]]
```

In [ ]:

```python
eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))

for i in range(len(eig_vals)):
    eigvec_sc = eig_vecs[:,i].reshape(53, 1)

for i in range(len(eig_vals)):
    eigv = eig_vecs[:,i].reshape(53, 1)
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)

W = np.hstack((eig_pairs[0][1].reshape(53, 1), eig_pairs[1][1].reshape(53, 1)))

W
```

```
Out[ ]: array([[ 2.30980460e-02+0.j, -1.87114021e-02+0.j],
       [-1.51556576e-03+0.j, -1.47933131e-03+0.j],
       [-2.96345734e-02+0.j, -2.29568808e-02+0.j],
       [-1.11506276e-01+0.j,  4.98448069e-02+0.j],
       [ 1.25225281e-01+0.j, -4.85266274e-02+0.j],
       [ 6.08224241e-02+0.j,  4.64768781e-02+0.j],
       [-1.97884662e-03+0.j,  2.81086455e-03+0.j],
       [ 6.82809213e-03+0.j, -4.93035745e-03+0.j],
       [-4.93522368e-02+0.j,  5.74399164e-02+0.j],
       [-9.91386841e-03+0.j,  3.31033711e-03+0.j],
       [-1.79712796e-02+0.j, -1.71002894e-02+0.j],
       [ 1.52886473e-06+0.j,  7.05213261e-07+0.j],
       [ 8.58350842e-03+0.j,  5.59601876e-03+0.j],
       [ 5.49628505e-03+0.j,  6.74053740e-04+0.j],
       [ 2.36901594e-04+0.j, -6.27171761e-04+0.j],
       [ 4.21602112e-04+0.j,  9.87152123e-03+0.j],
       [ 4.54243613e-03+0.j,  6.86732317e-03+0.j],
       [ 1.18260316e-02+0.j,  8.56657543e-03+0.j],
       [-9.74268625e-01+0.j, -9.80222225e-01+0.j],
       [-4.28345799e-03+0.j, -3.44262562e-04+0.j],
       [ 2.31666206e-03+0.j,  2.17154971e-03+0.j],
       [-7.80269670e-03+0.j, -1.13597723e-02+0.j],
       [ 2.00267257e-02+0.j, -7.29519345e-03+0.j],
       [-3.79791653e-08+0.j, -5.45130204e-08+0.j],
       [ 2.04131015e-02+0.j,  3.01970062e-02+0.j],
       [-1.23240972e-02+0.j,  1.85320074e-02+0.j],
       [ 6.76119941e-02+0.j,  5.91897374e-02+0.j],
       [-3.39479199e-02+0.j, -2.91609724e-02+0.j],
       [-1.58316926e-02+0.j, -1.23822283e-01+0.j],
       [-3.92757678e-04+0.j,  6.47702323e-05+0.j],
       [ 1.85998437e-03+0.j,  9.04843361e-04+0.j],
       [-2.81424730e-05+0.j, -2.21579064e-05+0.j],
       [ 5.57912391e-03+0.j,  3.81244839e-03+0.j],
       [ 2.53994666e-04+0.j, -1.26525629e-03+0.j],
       [ 3.44139690e-04+0.j,  2.37130149e-04+0.j],
       [ 1.73238751e-04+0.j,  3.60900533e-05+0.j],
       [-8.48592269e-04+0.j,  2.25981439e-03+0.j],
       [-4.90361226e-03+0.j,  7.26409371e-03+0.j],
       [ 1.13310764e-02+0.j, -5.49287835e-03+0.j],
       [ 1.16772981e-06+0.j,  1.24496638e-06+0.j],
       [-3.48898525e-05+0.j,  8.95992984e-05+0.j],
       [-2.56279513e-06+0.j,  6.01460543e-06+0.j],
       [ 1.53938621e-03+0.j, -1.80843779e-03+0.j],
       [ 6.40838560e-03+0.j, -3.68867284e-02+0.j],
       [ 1.13128600e-02+0.j, -9.89664168e-03+0.j],
       [-2.33445510e-03+0.j, -5.07527171e-03+0.j],
       [-5.60146062e-02+0.j, -3.09854587e-02+0.j],
       [ 7.41411253e-04+0.j,  3.15911034e-03+0.j],
       [-5.86232854e-02+0.j, -6.07412206e-02+0.j],
       [ 9.39660496e-03+0.j, -3.56673013e-03+0.j],
       [-8.69675648e-03+0.j, -2.47293196e-03+0.j],
       [ 1.63551899e-02+0.j,  1.13721279e-03+0.j],
       [-4.27016377e-04+0.j, -1.52202858e-04+0.j]])

In [ ]:
```