ECEN689-605 Final Computer Project

Shutong Jiao

Assignment 1

1

Linear Model Fit Result

Predictor	fitted coefficient	mean residual sum of squares	R Square
C	-16.20	169.08	0.01
N	20.09	164.31	0.03
Ni	2.34	85.48	0.50
Fe	-1.44	101.26	0.41
Mn	0.35	168.83	0.01
Si	-3.52	168.73	0.01
Cr	0.52	169.03	0.01

Figure 1: Linear Model fit result for each element

In the table above, we could observe that Ni is the best predictor of SFE, with a biggest R^2 of 0.50. The plot of SFE response against each of the seven variables is shown below. We could find that except Ni and Fe, other elements show a very weak linear relationship with SFE response, which is justified by their very low R^2 value like 0.01. We could imagine like just take an average of SFE is basically using these variable for regression.

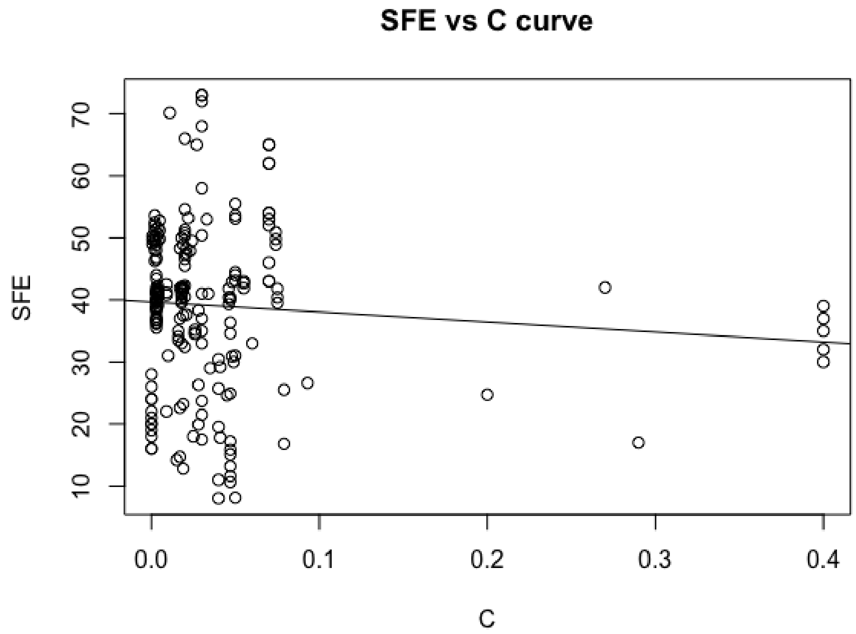

Figure 2: SFE response against C

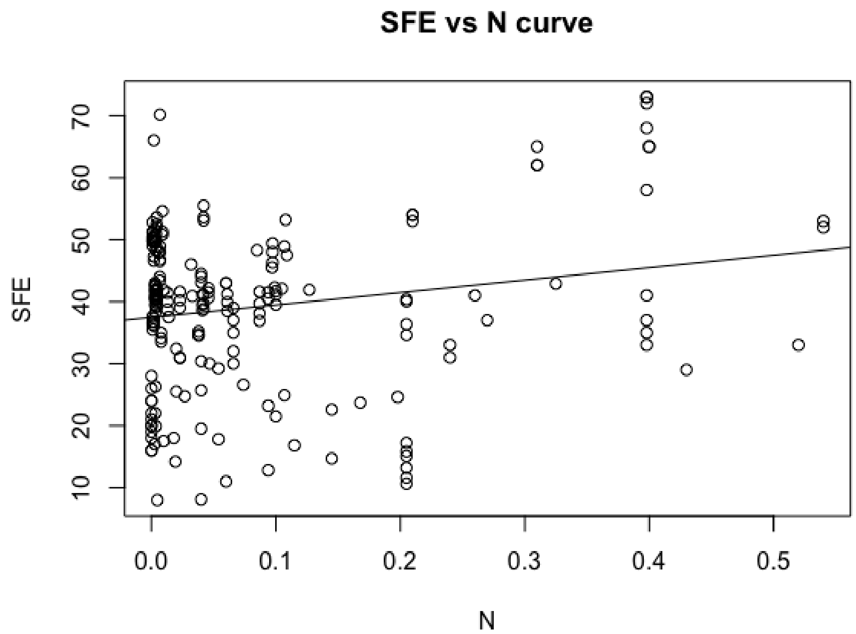

Figure 3: SFE response against N

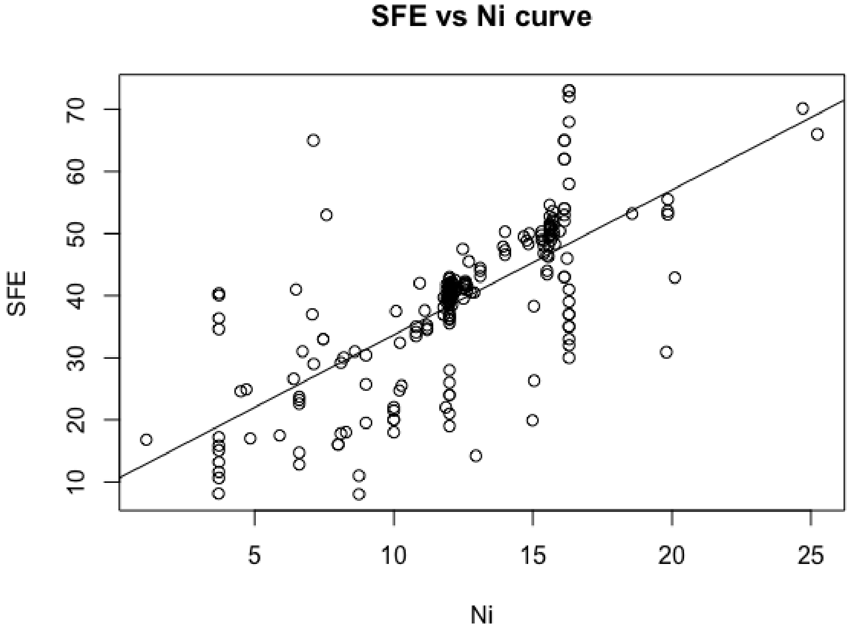

Figure 4: SFE response against Ni

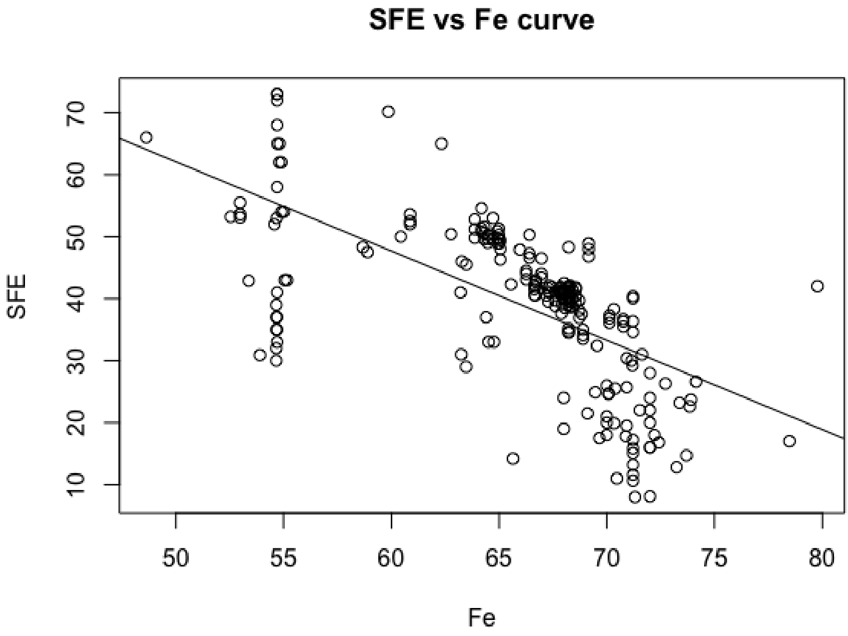

Figure 5: SFE response against Fe

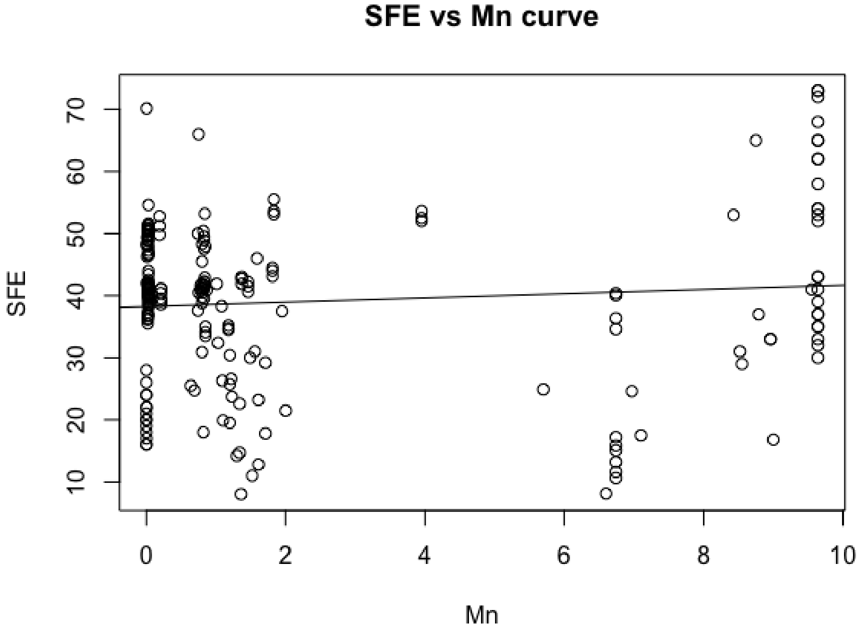

Figure 6: SFE response against Mn

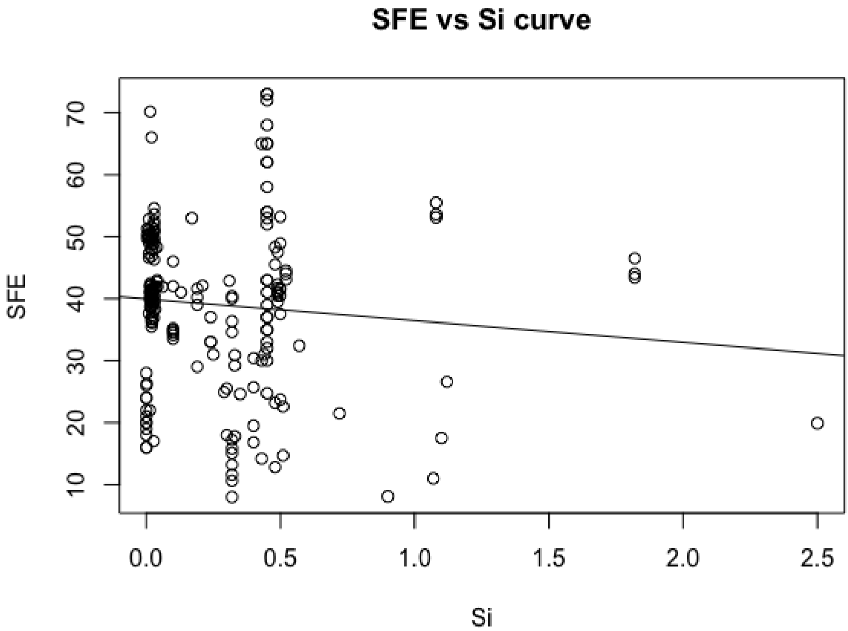

Figure 7: SFE response against Si

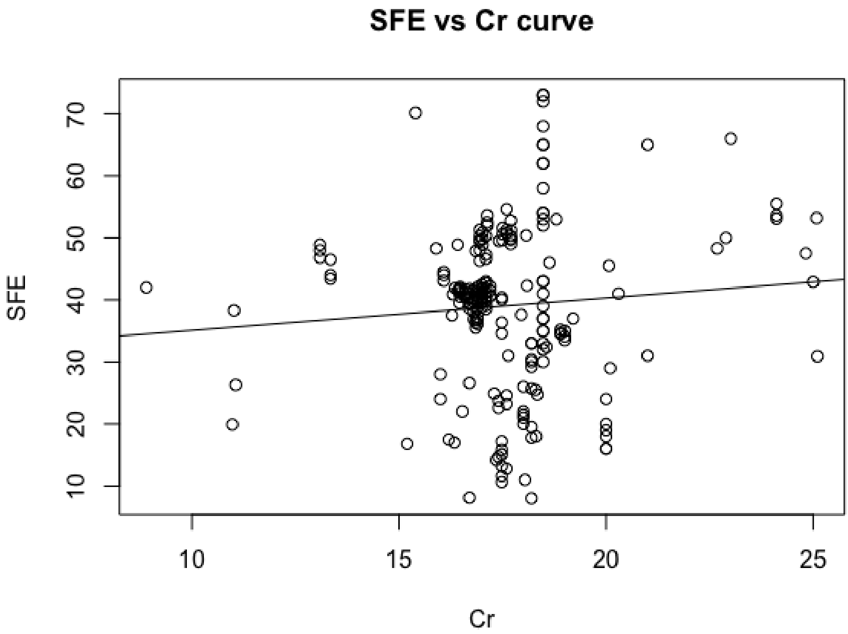

Figure 8: SFE response against Cr

2

Linear Model with Exh Fit Result

Predictors	fitted coefficient	mean residual sum of squares	R Square	Adjusted R Square
Ni	2.33534	85.47935	0.49797	0.49556
Ni N	2.388696 25.381187	76.38693	0.55351	0.54922
Fe Mn Cr	-2.352598 -1.415786 -2.367095	69.71190	0.59449	0.58861
C Fe Mn Cr	-41.609043 -2.403498 -1.112652 -2.50073	63.53840	0.63219	0.62504
C Fe Mn Cr Si	-39.9924238 -2.4441419 -0.9586294 -2.6265382 -6.7141602	58.64691	0.66215	0.65391

Figure 9: Linear model fit result using exhaustive search method

Linear Model with SFS Fit Result

Predictors	fitted coefficient	mean residual sum of squares	R Square	Adjusted R Square
Ni	2.33534	85.47935	0.49797	0.49556
Ni N	2.388696 25.381187	76.38693	0.55351	0.54922
Ni N Si	2.400898 29.278772 -5.977791	72.50772	0.57823	0.57823
Ni N Si C	2.421394 29.958250 -5.325052 -23.219941	70.46972	0.59206	0.58414
Ni N Si C Fe	1.9282493 19.1690646 -5.5221036 -29.8198461 -0.4570711	69.42260	0.60007	0.59032

Figure 10: Linear model fit result using sequential forward search method

According to the adjusted R^2 , the feature set for the most predictive model is (C,Fe,Mn,Cr,Si) with an adjusted R^2 of 0.65.
We could find that unlike classification problem, for both the exhaustive search and sequential forward search method, R^2 and adjusted R^2 increases when more features are introduced. This is intuitive in the sense that we do not have many features and thus we will not likely be over fitting. So more features will lead to better accuracy. But this increase is relatively very small considering the fact that we could use only one feature (Ni) to achieve a R^2 of 0.5. After we add four features we could only improve it by about 0.1. Therefore we could see that in the feature sets the main regression task is done by one feature. And this could also explain the fact that often different feature give very close R^2.
Compared with the related classification problem in project2, basically there is hard to find same feature set. But I do observe that when $d = 5$ feature set given by SFS using LDA is very similar to the corresponding feature set in our problem. But this maybe just due to that we only have 7 features and many features have little effect, which will be shown in the following section about Ridge and Lasso method for feature selection.
Also, like I have mentioned as there is not much difference from many feature set in this regression problem, feature selection techniques will perhaps not help much to improve R^2 and adjusted R^2. But for the case of classification problem, we could see that proper feature selection do improve the accuracy of the classifier.
Besides we could find that the best test set error rate in classification problem is 0.06, which basically means a good classifier. But our regression model not have a R^2 of about 0.6, which is hardly a good regression model. Maybe this is because regression is a harder problem compared with classification. Or because that we have exclude the middle SFE data in classification problem. I believe that if we also exclude the middle SFE data in regression problem , a better regression model could be made.

3

Ridge regression coefficients for each value of λ

Lambda	C	N	Ni	Fe	Mn	Si	Cr
50.00	-0.04037	0.03166	1.03807	-1.01388	-0.25198	-0.17911	-0.59256
30.00	-0.06779	0.05550	1.03164	-1.12808	-0.33766	-0.29773	-0.81472
15.00	-0.13552	0.11608	0.88255	-1.36366	-0.52528	-0.58287	-1.17099
7.00	-0.28466	0.25503	0.49601	-1.78331	-0.87686	-1.16825	-1.66260
3.00	-0.63169	0.59938	-0.17306	-2.44023	-1.43856	-2.29794	-2.33535
1.00	-1.70072	1.73729	-1.09704	-3.32285	-2.19853	-4.34598	-3.20011
0.30	-4.65207	4.91609	-1.71327	-3.91176	-2.73872	-6.22988	-3.79119
0.10	-9.88799	10.05361	-1.89007	-4.08811	-2.96996	-7.06420	-4.00456
0.03	-17.00608	15.46740	-1.88293	-4.09356	-3.05366	-7.36371	-4.06132
0.01	-21.89916	17.83560	-1.86424	-4.08348	-3.06583	-7.43119	-4.07778

1

Figure 11: Regression coefficients using ridge method

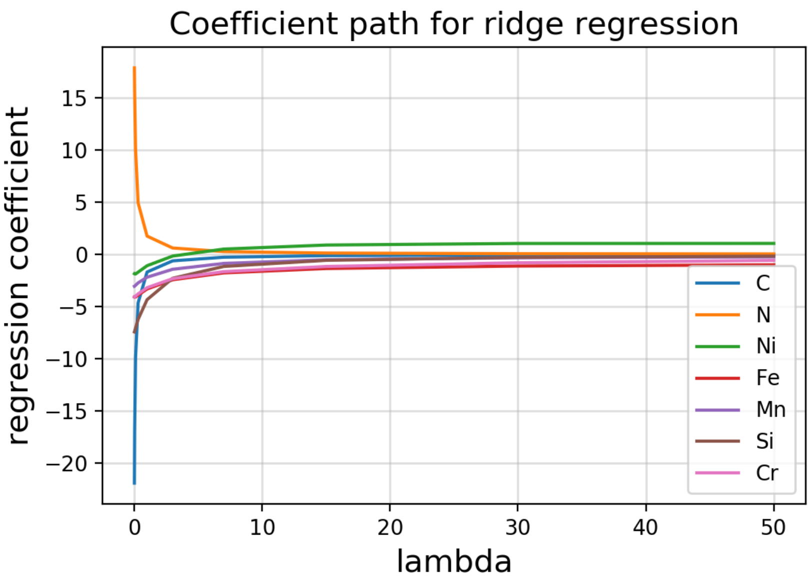

Figure 12: Coefficient path for ridge method

Lasso regression coefficients for each value of λ

Lambda	C	N	Ni	Fe	Mn	Si	Cr
50.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
30.00000	0.00000	0.00000	0.00000	-0.52980	0.00000	0.00000	0.00000
15.00000	0.00000	0.00000	0.68020	-0.65391	0.00000	0.00000	0.00000
7.00000	0.00000	0.00000	1.22048	-0.63367	0.00000	0.00000	0.00000
3.00000	0.00000	0.00000	1.46152	-0.66001	0.00000	0.00000	-0.11437
1.00000	0.00000	0.00000	0.27577	-1.94152	-1.01421	0.00000	-1.75993
0.30000	0.00000	0.00000	-0.55283	-2.82514	-1.69976	-4.51707	-2.75492
0.10000	-5.93751	6.87133	-1.57867	-3.79206	-2.65897	-6.58567	-3.70436
0.03000	-19.95486	15.41763	-1.72836	-3.95453	-2.90590	-7.17953	-3.94149
0.01000	-23.96039	17.83629	-1.77956	-4.00858	-2.98263	-7.35177	-4.01569

1

Figure 13: Regression coefficients using lasso method

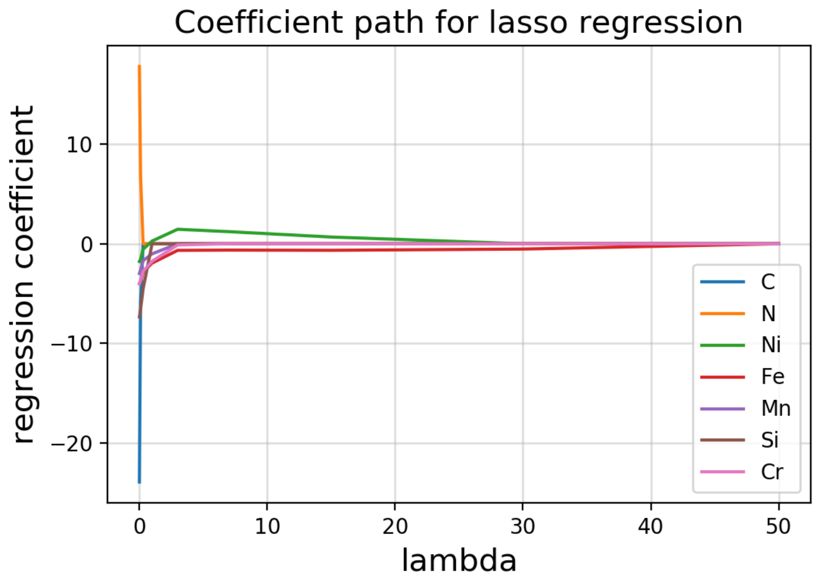

Figure 14: Coefficient path for lasso method

The regression coefficient and coefficient path for the lasso and ridge regression method is shown in the figures above. Comparing the two coefficient paths, we could find that in the case of Lasso, as the L1 norm is utilized, many coefficients have been quickly driven to zero. However, in the case of ridge, coefficients quickly drop to a relative low but non-zero value with the increase of regularization parameter λ. This difference could be explained that L1 norm defines a linear boundary while L2 norm defines a circular boundary.
So we will then discuss how this table help us perform feature selection.
For the case of Ridge, we could see that when λ is high the coefficients for C and N is very low and Fe/Ni is relatively high. So we could exclude C and N. But is then very hard for us to decide whether to discard other features as their coefficients are not that close to zero.
For the case of Lasso, it shows very clear that Ni and Fe are the top two feature as they are the last two to become zero as λ increases. This also agrees with our feature selection result for the classification problem in the previous projects.

Therefore we could conclude that for this problem Lasso is more helpful with regard to feature selection than Ridge.

Python Code

```
#import lib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from scipy.stats import ttest_ind
import math
from numpy.linalg import inv
from sklearn.linear_model import LinearRegression
from itertools import chain, combinations
import rpy2

#preprocessing for SFE data
SFE_res = pd.read_excel("SFE_Dataset.xlsx")

header = list(SFE_res.columns.values)
n_col = SFE_res.shape[1]
n_row = SFE_res.shape[0]
nonzero_header = SFE_res.astype(bool).sum(axis=0);
for i in range(0,n_col):
    if nonzero_header.get(i) < 0.6 * n_row :
        del SFE_res[header[i]]
SFE_res;

n_col_new = SFE_res.shape[1]

nonzero_index = SFE_res.astype(bool).sum(axis=1)
nonzero_index
todrop = []
for i in range(0,n_row):
    if nonzero_index.get(i) < n_col_new :
        todrop.append(i)


drop_data = SFE_res.drop(SFE_res.index[todrop])

SFE_final = drop_data.reset_index(drop=True)

#use rpy2 to run R in jupyter notebook

%load_ext rpy2.ipython

%R SFE.final =
    read.csv("/Users/jiaoshutong/Documents/ECEN689doc/final_data.csv",header=T);


#get linear regression model to each of the seven variables and make the plot

%%R

lm_C <- lm(SFE ~ C, data=SFE.final)
av_C <- anova(lm_C)
sm_C <- summary(lm_C)
C_coeff = c(sm_C$"coefficients"[2],av_C$"Mean Sq"[2],sm_C$"r.squared" )
C_coeff
plot(SFE.final$C,SFE.final$SFE,xlab = "C",ylab = "SFE", main = "SFE vs C curve")
abline(lm_C)

lm_N <- lm(SFE ~ N, data=SFE.final)
av_N <- anova(lm_N)
sm_N <- summary(lm_N)
```

```
N_coeff = c(sm_N$"coefficients"[2],av_N$"Mean Sq"[2],sm_N$"r.squared" )
N_coeff
plot(SFE.final$N,SFE.final$SFE,xlab = "N",ylab = "SFE", main = "SFE vs N curve" )
abline(lm_N)


lm_Ni <- lm(SFE ~ Ni, data=SFE.final)
av_Ni <- anova(lm_Ni)
sm_Ni <- summary(lm_Ni)
Ni_coeff = c(sm_Ni$"coefficients"[2],av_Ni$"Mean Sq"[2],sm_Ni$"r.squared" )
Ni_coeff
plot(SFE.final$Ni,SFE.final$SFE,xlab = "Ni",ylab = "SFE", main = "SFE vs Ni curve" )
abline(lm_Ni)

lm_Fe <- lm(SFE ~ Fe, data=SFE.final)
av_Fe <- anova(lm_Fe)
sm_Fe <- summary(lm_Fe)
Fe_coeff = c(sm_Fe$"coefficients"[2],av_Fe$"Mean Sq"[2],sm_Fe$"r.squared" )
Fe_coeff
plot(SFE.final$Fe,SFE.final$SFE,xlab = "Fe",ylab = "SFE", main = "SFE vs Fe curve" )
abline(lm_Fe)


lm_Mn <- lm(SFE ~ Mn, data=SFE.final)
av_Mn <- anova(lm_Mn)
sm_Mn <- summary(lm_Mn)
Mn_coeff = c(sm_Mn$"coefficients"[2],av_Mn$"Mean Sq"[2],sm_Mn$"r.squared" )
Mn_coeff
plot(SFE.final$Mn,SFE.final$SFE,xlab = "Mn",ylab = "SFE", main = "SFE vs Mn curve" )
abline(lm_Mn)

lm_Si <- lm(SFE ~ Si, data=SFE.final)
av_Si <- anova(lm_Si)
sm_Si <- summary(lm_Si)
Si_coeff = c(sm_Si$"coefficients"[2],av_Si$"Mean Sq"[2],sm_Si$"r.squared" )
Si_coeff
plot(SFE.final$Si,SFE.final$SFE,xlab = "Si",ylab = "SFE", main = "SFE vs Si curve" )
abline(lm_Si)

lm_Cr <- lm(SFE ~ Cr, data=SFE.final)
av_Cr <- anova(lm_Cr)
sm_Cr <- summary(lm_Cr)
Cr_coeff = c(sm_Cr$"coefficients"[2],av_Cr$"Mean Sq"[2],sm_Cr$"r.squared" )
Cr_coeff
plot(SFE.final$Cr,SFE.final$SFE,xlab = "Cr",ylab = "SFE", main = "SFE vs Cr curve" )
abline(lm_Cr)


#define function for exhaustive search and sequential forward search
def RsqEst(sfe_table,var_idx_set):
   LR = LinearRegression()
   LR.fit(sfe_table.iloc[:,var_idx_set],sfe_table.iloc[:,7])
   return LR.score(sfe_table.iloc[:,var_idx_set],sfe_table.iloc[:,7])


def SFS (sfe_table, est, n_features):
   return SFS_helper(sfe_table,est,n_features,[])

def SFS_helper (sfe_table, est, n_features,curr_features):
   if len(curr_features) == n_features:
      return curr_features
   else:
      total_features = sfe_table.columns.size - 1
```

```
        iterlist = []
        for i in range(0,total_features):
            iterlist.append(i)
        iterlist = [x for x in iterlist if x not in curr_features]
        optimalSet = []
        minErr = 1
        for feature in iterlist:
            curr_features.append(feature)
            currErr = 1 - RsqEst(sfe_table,curr_features)
#             print(curr_features)
#             print(currErr)
#             print('-----')
            if currErr < minErr:
                minErr = currErr
                optimalSet = list(curr_features)
            curr_features.remove(curr_features[len(curr_features) - 1])
        return SFS_helper(sfe_table, est, n_features,optimalSet)

def exhaustiveSearch (sfe_table, est, n_features):
    total_features = sfe_table.columns.size - 1
    iterlist = []
    for i in range(0,total_features):
        iterlist.append(i)
    subsets = combinations(iterlist,n_features)
    minErr = 1
    optimalSet = []
    for subset in subsets:
        currErr = 1 - RsqEst(sfe_table,list(subset))
#         print(subset)
#         print(currErr)
#         print('-----')
        if currErr < minErr:
            minErr = currErr
            optimalSet = subset
    return list(optimalSet)

# for case of 1 varaible to 5 variable

# 1 varaible

sfs_res = SFS(SFE_final,RsqEst,1)
print(SFE_final.columns[sfs_res])

exh_res = exhaustiveSearch(SFE_final,RsqEst,1)
print(SFE_final.columns[exh_res])

%%R
lm <- lm(SFE ~Ni, data=SFE.final)
av <- anova(lm)
sm <- summary(lm)
lm_coeff = c(av$"Mean Sq"[2],sm$"r.squared",sm$"adj.r.squared" )
print(lm_coeff)
print(sm$"coefficients")
# print(sm)
print(av)
# print(av$"Mean Sq"[2])

# 2 variable

sfs_res = SFS(SFE_final,RsqEst,2)
print(SFE_final.columns[sfs_res])

exh_res = exhaustiveSearch(SFE_final,RsqEst,2)
```

```
print(SFE_final.columns[exh_res])

%%R
lm <- lm(SFE ~Ni+N, data=SFE.final)
av <- anova(lm)
sm <- summary(lm)
lm_coeff = c(av$"Mean Sq"[3],sm$"r.squared",sm$"adj.r.squared" )
print(lm_coeff)
# print(sm)
# print(av)
# print(av$"Mean Sq"[3])

# 3 variable

sfs_res = SFS(SFE_final,RsqEst,3)
print(SFE_final.columns[sfs_res])

exh_res = exhaustiveSearch(SFE_final,RsqEst,3)
print(SFE_final.columns[exh_res])

%%R
lm <- lm(SFE ~Ni+N+Si, data=SFE.final)
av <- anova(lm)
sm <- summary(lm)
lm_coeff = c(av$"Mean Sq"[4],sm$"r.squared",sm$"adj.r.squared" )
print(lm_coeff)
print(sm$"coefficients")

%%R
lm <- lm(SFE ~Fe+Mn+Cr, data=SFE.final)
av <- anova(lm)
sm <- summary(lm)
lm_coeff = c(av$"Mean Sq"[4],sm$"r.squared",sm$"adj.r.squared" )
print(lm_coeff)

# 4 variable

sfs_res = SFS(SFE_final,RsqEst,4)
print(SFE_final.columns[sfs_res])

exh_res = exhaustiveSearch(SFE_final,RsqEst,4)
print(SFE_final.columns[exh_res])

%%R
lm <- lm(SFE ~Ni+N+Si+C, data=SFE.final)
av <- anova(lm)
sm <- summary(lm)
lm_coeff = c(av$"Mean Sq"[5],sm$"r.squared",sm$"adj.r.squared" )
print(lm_coeff)
print(sm$"coefficients")

%%R
lm <- lm(SFE ~C+Fe+Mn+Cr, data=SFE.final)
av <- anova(lm)
sm <- summary(lm)
lm_coeff = c(av$"Mean Sq"[5],sm$"r.squared",sm$"adj.r.squared" )
print(lm_coeff)
print(sm$"coefficients")

# 5 variable

sfs_res = SFS(SFE_final,RsqEst,5)
print(SFE_final.columns[sfs_res])
```

```
exh_res = exhaustiveSearch(SFE_final,RsqEst,5)
print(SFE_final.columns[exh_res])

%%R
lm <- lm(SFE ~Ni+N+Si+C+Fe, data=SFE.final)
av <- anova(lm)
sm <- summary(lm)
lm_coeff = c(av$"Mean Sq"[6],sm$"r.squared",sm$"adj.r.squared" )
print(lm_coeff)
print(sm$"coefficients")

%%R
lm <- lm(SFE ~C+Fe+Mn+Cr+Si, data=SFE.final)
av <- anova(lm)
sm <- summary(lm)
lm_coeff = c(av$"Mean Sq"[6],sm$"r.squared",sm$"adj.r.squared" )
print(lm_coeff)
print(sm$"coefficients")


#use glmnet lib to perform lasso and regression
%%R
library(glmnet)
x = model.matrix(~C+N+Ni+Fe+Mn+Si+Cr,SFE.final)
y = SFE.final$SFE

%%R
fit.lasso <- glmnet(x[,-1],y, family="gaussian", alpha=1,lambda = c(50, 30, 15, 7, 3, 1,
    0.30, 0.10, 0.03, 0.01), standardize = FALSE)
plot(fit.lasso, xvar = "lambda", label = TRUE)
lasso_res = fit.lasso$beta
# write.table(as.matrix(coef(fit.lasso)), file = "lasso_coeff_new2.csv", sep = ",",
    col.names = NA)
as.matrix(coef(fit.lasso))

%%R
fit.ridge <- glmnet(x[,-1],y, family="gaussian", alpha=0,lambda = c(50, 30, 15, 7, 3, 1,
    0.30, 0.10, 0.03, 0.01), standardize = FALSE)
plot(fit.ridge, xvar = "lambda", label = TRUE)
ridge_res = fit.ridge$beta
# write.table(as.matrix(coef(fit.ridge)), file = "ridge_coeff_new2.csv", sep = ",",
    col.names = NA)
as.matrix(coef(fit.ridge))

lam_var = np.array([50, 30, 15, 7, 3, 1, 0.30, 0.10, 0.03, 0.01])

lasso_coeff = pd.read_csv("lasso_coeff_new2.csv")

lasso_coeff = lasso_coeff.iloc[:,1:11]
lasso_coeff = lasso_coeff.iloc[1:8,:]

ridge_coeff = pd.read_csv("ridge_coeff_new2.csv")

ridge_coeff = ridge_coeff.iloc[:,1:11]
ridge_coeff = ridge_coeff.iloc[1:8,:]
ridge_coeff

SFE_dict = {0 : "C", 1 : "N",2:"Ni",3:"Fe",4:"Mn",5:"Si",6:"Cr" }

plt.clf()
fig, ax = plt.subplots()
for i in range(0,7):
```

```
    ax.plot(lam_var,ridge_coeff.iloc[i,:],label = SFE_dict[i])
plt.title('Coefficient path for ridge regression',fontsize=15)
plt.xlabel('lambda',fontsize=15)
plt.ylabel('regression coefficient',fontsize=15)
plt.grid(linewidth=1,alpha = 0.4)
plt.legend(loc='best')
# plt.savefig('ridge_cf_new.png',dpi = 200)
plt.show()

plt.clf()
fig, ax = plt.subplots()
for i in range(0,7):
    ax.plot(lam_var,lasso_coeff.iloc[i,:],label = SFE_dict[i])
plt.title('Coefficient path for lasso regression',fontsize=15)
plt.xlabel('lambda',fontsize=15)
plt.ylabel('regression coefficient',fontsize=15)
plt.grid(linewidth=1,alpha = 0.4)
plt.legend(loc='best')
# plt.savefig('lasso_cf_new.png',dpi = 200)
plt.show()
```

Assignment2

(a)

CV error for all 6 classifier on 5 max pooling layer

	layer1	layer2	layer3	layer4	layer5
spheroidite - network classifier	0.5810	0.5805	0.5740	0.5795	0.0395
spheroidite - pearlite classifier	0.5860	0.5850	0.5755	0.5610	0.0420
spheroidite - spheroidite_widmanstatten classifier	0.3750	0.3750	0.3750	0.3750	0.0231
network - pearlite classifier	0.5770	0.5825	0.5775	0.5720	0.0290
network - spheroidite_widmanstatten classifier	0.3750	0.3750	0.3750	0.3750	0.0181
pearlite - spheroidite_widmanstatten classifier	0.3750	0.3750	0.3750	0.3750	0.1275

1

Figure 15: Cross Validation error estimate for each of the six pairwise two-label classifiers using features from different layer

From the table above, we could observe that the features from the 5th max pooling layer gives the lowest cross-validation error, which agrees with the results from Ling's paper. Also the overall cv error from the 5th layer is very low and thus we will use it to train the one vs one multi-label voting classifier.
For convolutional neuron networks (CNN), it is believed that the earlier layers tend to catch simpler patterns like edges and the later layers detect higher-level patterns. For the case of the steel data in our project, we could speculate that the micro-structure is a kind of higher level pattern and thus the feature from the latest block performs best.In Ling's paper, the author also thinks that the layers later in the CNN are more relevant to the steel data set than the powder data set because the steel textures, with lamellar and cell-like structures, are more complex.

(b)

Test set error for all 6 pairwise classifier and multilabel classifier

	test set size	test set error
spheroidite - network classifier	386	0.0337
spheroidite - pearlite classifier	298	0.0000
spheroidite - spheroidite_widmanstatten classifier	295	0.0712
network - pearlite classifier	136	0.0735
network - spheroidite_widmanstatten classifier	133	0.0602
pearlite - spheroidite_widmanstatten classifier	45	0.0889
One Vs One classifier	431	0.0719

1

Figure 16: Test error for each of the six pairwise two-label classifiers and the multilabel one-vs-one voting classifier

We could observe from the table above that regardless of test set and types of classifier, the SVM classifier using features from the 5th layer performs very well with regard to test set error.

(c)

multi-label classifier on mixed label micrographs

	multi-label classifier		pearlite vs. spheroidite classifier
Prediction Result	pearlite + spheroidite	pearlite + widmanstatten	pearlite + spheroidite
spheroidite	81	8	84
pearlite	23	16	23
spheroidite_widmanstatten	2	3	0
network	1	0	0

Figure 17: Prediction result for mixed micro-graphs using multi-label classifier and pairwise classifier

The first column in the table above shows the prediction result using the multi-label classifier on 'pearlite + spheroidite' and 'pearlite + widmanstatten' micro graphs.
For the first (pearlite and spheroidite) micro-graph, we could see that most of the prediction results are either pearlite or spheroidite, which corresponds accurately with the constituent in these kind of micro-graphs.For another mixed micro-graph(pearlite and widmanstatten), most of the prediction results are spheroidite and pearlite, which is not as accurate as the first one.

(d)

The second column of the previous table shows the prediction result by pearlite vs. spheroidite pairwise classifier. Compared with the corresponding result for multi-label classifier in part(c), their prediction is very close. The multi-label classifier only mistook three micro-structure according to the criterion of the pairwise classifier. So we could say that the multi-label classifier using voting method is very successful as it performs very close to one of its pairwise classifier.

(e)

multi-label classifier on mixed label micrographs

Prediction Result	martensite
spheroidite	17
pearlite	16
spheroidite_widmanstatten	3
network	0

Figure 18: Prediction result for martensite micrograph using multi-label classifier

The table above shows the prediction result for martensite micro-graph. Most of the results are spheroidite and pearlite, which may suggest that pattern of microstructure of martensite is similar to that of spheroidite and pearlite.

Python code

```
# coding: utf-8

# import lib


from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
import pandas as pd
import numpy as np
import os
import tensorflow as tf
import PIL
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.multiclass import OneVsOneClassifier as ovoclf


# define some useful global var


path_prefix = "/Users/jiaoshutong/Documents/ECEN689doc/final_2/micrograph/"

pics = os.listdir(path_prefix)

base_model = VGG16(weights='imagenet',include_top=False,input_shape=(484,645,3))


# import image and preprocessing


def getFeature(tensor):
    dim = tensor.shape
    num = dim[0] * dim[1] * dim[2]
    return np.einsum('ijkl->l',tensor)/num

def getFeatureSet(idx_set,layer_name):
    filenames = list(path.iloc[idx_set,0])
    featureSet = []
#     base_model = VGG16(weights='imagenet',input_shape = (484,645,3),include_top=False)
    model = Model(inputs=base_model.input,
         outputs=base_model.get_layer(layer_name).output)
    for filename in filenames:
        img_path = path_prefix + filename
        img = image.load_img(img_path)
        img = img.crop((0,0,645,484))

        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        x = preprocess_input(x)
        pool_features= model.predict(x)
        feature = getFeature(pool_features)
        featureSet.append(feature)
    return np.array(featureSet)


def imgToFeature(img_path,layer_name):
    base_model = VGG16(weights='imagenet',input_shape = (484,645,3),include_top=False)
```

```python
    model = Model(inputs=base_model.input,
        outputs=base_model.get_layer(layer_name).output)

    img = image.load_img(img_path)
    img = img.crop((0,0,645,484))

    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    pool_features= model.predict(x)
    return getFeature(pool_features)


# import micrograph dataset and get index for each mirostructure
micrograph = pd.read_csv("/Users/jiaoshutong/Documents/ECEN689doc/final_2/micrograph.csv")

col = micrograph.columns;

path = micrograph[col[[1,9]]]

group = path.groupby(col[9]).groups

group.keys()

spheroidite_idx = group.get('spheroidite')
spheroidite_train_idx = spheroidite_idx[0:100]
spheroidite_test_idx = spheroidite_idx[100:len(spheroidite_idx)]

spheroidite_widmanstatten_idx = group.get('spheroidite+widmanstatten')
spheroidite_widmanstatten_train_idx = spheroidite_widmanstatten_idx[0:60]
spheroidite_widmanstatten_test_idx =
    spheroidite_widmanstatten_idx[60:len(spheroidite_widmanstatten_idx)]

network_idx = group.get('network')
network_train_idx = network_idx[0:100]
network_test_idx = network_idx[100:len(network_idx)]

pearlite_idx = group.get('pearlite')
pearlite_train_idx = pearlite_idx[0:100]
pearlite_test_idx = pearlite_idx[100:len(pearlite_idx)]

pearlite_spheroidite_idx = group.get('pearlite+spheroidite')
pearlite_spheroidite_test_idx = pearlite_spheroidite_idx

pearlite_widmanstatten_idx = group.get('pearlite+widmanstatten')
pearlite_widmanstatten_test_idx = pearlite_widmanstatten_idx

martensite_idx = group.get('martensite')
martensite_test_idx = martensite_idx



# calculate feature set from different lay for different microstructure
martensite_test_feature = []
for i in range (1,6):
    martensite_test_feature.append(getFeatureSet(martensite_test_idx,'block' +
        str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/martensite_test_feature_block'+
        str(i) + '.csv', martensite_test_feature[i - 1])
    print(i)

network_test_feature = []
```

```
for i in range (1,6):
    network_test_feature.append(getFeatureSet(network_test_idx,'block' + str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/network_test_feature_block'+
        str(i) + '.csv', network_test_feature[i - 1])
    print(i)

network_train_feature = []
for i in range (1,6):
    network_train_feature.append(getFeatureSet(network_train_idx,'block' +
        str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/network_train_feature_block'+
        str(i) + '.csv', network_train_feature[i - 1])
    print(i)

pearlite_test_feature = []
for i in range (1,6):
    pearlite_test_feature.append(getFeatureSet(pearlite_test_idx,'block' +
        str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/pearlite_test_feature_block'+
        str(i) + '.csv', pearlite_test_feature[i - 1])
    print(i)

pearlite_train_feature = []
for i in range (1,6):
    pearlite_train_feature.append(getFeatureSet(pearlite_train_idx,'block' +
        str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/pearlite_train_feature_block'+
        str(i) + '.csv', pearlite_train_feature[i - 1])
    print(i)

pearlite_spheroidite_test_feature = []
for i in range (1,6):
    pearlite_spheroidite_test_feature.append(getFeatureSet(pearlite_spheroidite_test_idx,'block'
        + str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/pearlite_spheroidite_test_feature_block'+
        str(i) + '.csv', pearlite_spheroidite_test_feature[i - 1])
    print(i)

pearlite_widmanstatten_test_feature = []
for i in range (1,6):
    pearlite_widmanstatten_test_feature.append(getFeatureSet(pearlite_widmanstatten_test_idx,'block'
        + str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/pearlite_widmanstatten_test_feature_block'+
        str(i) + '.csv', pearlite_widmanstatten_test_feature[i - 1])
    print(i)

spheroidite_test_feature = []
for i in range (1,6):
    spheroidite_test_feature.append(getFeatureSet(spheroidite_test_idx,'block' +
        str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/spheroidite_test_feature_block'+
        str(i) + '.csv', spheroidite_test_feature[i - 1])
    print(i)

spheroidite_train_feature = []
for i in range (1,6):
    spheroidite_train_feature.append(getFeatureSet(spheroidite_train_idx,'block' +
        str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/spheroidite_train_feature_block'+
        str(i) + '.csv', spheroidite_train_feature[i - 1])
    print(i)

spheroidite_widmanstatten_test_feature = []
```

```
for i in range (1,6):
    spheroidite_widmanstatten_test_feature.append(getFeatureSet(spheroidite_widmanstatten_test_idx,'block'
        + str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/spheroidite_widmanstatten_test_feature_block
        str(i) + '.csv', spheroidite_widmanstatten_test_feature[i - 1])
    print(i)

spheroidite_widmanstatten_train_feature = []
for i in range (1,6):
    spheroidite_widmanstatten_train_feature.append(getFeatureSet(spheroidite_widmanstatten_train_idx,'block'
        + str(i)+'_pool'))
    np.savetxt('/Users/jiaoshutong/Documents/ECEN689doc/features/spheroidite_widmanstatten_train_feature_bloc
        str(i) + '.csv', spheroidite_widmanstatten_train_feature[i - 1])
    print(i)




#(a)select corrsponding feature set for all 6 pairwise classifier based on cv error

# do 10-fold cv on sp - net clf

spheroidite_train_label = path.iloc[spheroidite_train_idx,1]
network_train_label = path.iloc[network_train_idx,1]
labels = list(spheroidite_train_label) + list(network_train_label)
sp_net_train_labels = labels
for layer in range(1,6):
    scores = 0
    for i in range(0,10):
        sp_net_clf = SVC()
        scores = scores +
            cross_val_score(sp_net_clf,np.concatenate((spheroidite_train_feature[layer -
            1],network_train_feature[layer - 1])),labels, cv=KFold(n_splits=10,
            shuffle=True)).mean()
    print('error of block ',str(layer) +' : ' , 1 - scores/10)

# do 10-fold cv on sp - pear clf
spheroidite_train_label = path.iloc[spheroidite_train_idx,1]
pearlite_train_label = path.iloc[pearlite_train_idx,1]
labels = list(spheroidite_train_label) + list(pearlite_train_label)

for layer in range(1,6):
    scores = 0
    for i in range(0,10):
        sp_oear_clf = SVC()
        scores = scores +
            cross_val_score(sp_net_clf,np.concatenate((spheroidite_train_feature[layer -
            1],
                                                       pearlite_train_feature[layer
                                                           - 1])),labels,
                                   cv=KFold(n_splits=10, shuffle=True)).mean()
    print('error of block ',str(layer) +' : ' , 1 - scores/10)


# do 10-fold cv on sp - sw clf
spheroidite_train_label = path.iloc[spheroidite_train_idx,1]
spheroidite_widmanstatten_train_label = path.iloc[spheroidite_widmanstatten_train_idx,1]
labels = list(spheroidite_train_label) + list(spheroidite_widmanstatten_train_label)

for layer in range(1,6):
    scores = 0
    for i in range(0,10):
```

```python
        sp_oear_clf = SVC()
        scores = scores +
            cross_val_score(sp_net_clf,np.concatenate((spheroidite_train_feature[layer -
            1],spheroidite_widmanstatten_train_feature[layer - 1])),labels,
            cv=KFold(n_splits=10, shuffle=True)).mean()
    print('error of block ',str(layer) +' : ' , 1 -scores/10)

# do 10-fold cv on net - pear clf
network_train_label = path.iloc[network_train_idx,1]
pearlite_train_label = path.iloc[pearlite_train_idx,1]
labels = list(network_train_label) + list(pearlite_train_label)

for layer in range(1,6):
    scores = 0
    for i in range(0,10):
        sp_oear_clf = SVC()
        scores = scores +
            cross_val_score(sp_net_clf,np.concatenate((network_train_feature[layer -
            1],pearlite_train_feature[layer - 1])),labels, cv=KFold(n_splits=10,
            shuffle=True)).mean()
    print('error of block ',str(layer) +' : ' , 1 - scores/10)



# do 10-fold cv on net - sw clf
network_train_label = path.iloc[network_train_idx,1]
spheroidite_widmanstatten_train_label = path.iloc[spheroidite_widmanstatten_train_idx,1]
labels = list(network_train_label) + list(spheroidite_widmanstatten_train_label)

for layer in range(1,6):
    scores = 0
    for i in range(0,10):
        sp_oear_clf = SVC()
        scores = scores +
            cross_val_score(sp_net_clf,np.concatenate((network_train_feature[layer -
            1],spheroidite_widmanstatten_train_feature[layer - 1])),labels,
            cv=KFold(n_splits=10, shuffle=True)).mean()
    print('error of block ',str(layer) +' : ' , 1 - scores/10)

# do 10-fold cv on pear - sw clf
pearlite_train_label = path.iloc[pearlite_train_idx,1]
spheroidite_widmanstatten_train_label = path.iloc[spheroidite_widmanstatten_train_idx,1]
labels = list(pearlite_train_label) + list(spheroidite_widmanstatten_train_label)

for layer in range(1,6):
    scores = 0
    for i in range(0,10):
        sp_oear_clf = SVC()
        scores = scores +
            cross_val_score(sp_net_clf,np.concatenate((pearlite_train_feature[layer -
            1],spheroidite_widmanstatten_train_feature[layer - 1])),labels,
            cv=KFold(n_splits=10, shuffle=True)).mean()
    print('error of block ',str(layer) +' : ' , 1 - scores/10)

# (b) calculate test set error for 6 pairwise classifier
spheroidite_test_label = path.iloc[spheroidite_test_idx,1]
network_test_label = path.iloc[network_test_idx,1]
pearlite_test_label = path.iloc[pearlite_test_idx,1]
spheroidite_widmanstatten_test_label = path.iloc[spheroidite_widmanstatten_test_idx,1]

sp_net_opt_clf = SVC()
sp_net_opt_clf.fit(np.concatenate((spheroidite_train_feature[4],network_train_feature[4])),list(spheroidite_t
    + list(network_train_label))
```

```
print(sp_net_opt_clf.score(spheroidite_test_feature[4],spheroidite_test_label))
print(sp_net_opt_clf.score(network_test_feature[4],network_test_label))
print(1 -
    sp_net_opt_clf.score(np.concatenate((spheroidite_test_feature[4],network_test_feature[4])),list(spheroidi
    + list(network_test_label)))
print(np.concatenate((spheroidite_test_feature[4],network_test_feature[4])).shape)
print("-----")


sp_pear_opt_clf = SVC()
sp_pear_opt_clf.fit(np.concatenate((spheroidite_train_feature[layer -
    1],pearlite_train_feature[layer - 1])),list(spheroidite_train_label) +
    list(pearlite_train_label))
print(sp_pear_opt_clf.score(spheroidite_test_feature[4],spheroidite_test_label))
print(sp_pear_opt_clf.score(pearlite_test_feature[4],pearlite_test_label))
print(1 - sp_pear_opt_clf.score(np.concatenate((spheroidite_test_feature[layer - 1],
                                    pearlite_test_feature[layer - 1])),
                        list(spheroidite_test_label) + list(pearlite_test_label)))
print(np.concatenate((spheroidite_test_feature[4],pearlite_test_feature[4])).shape)

print("-----")

sp_sw_opt_clf = SVC()
sp_sw_opt_clf.fit(np.concatenate((spheroidite_train_feature[layer -
    1],spheroidite_widmanstatten_train_feature[layer - 1])),list(spheroidite_train_label)
    + list(spheroidite_widmanstatten_train_label))
print(sp_sw_opt_clf.score(spheroidite_test_feature[4],spheroidite_test_label))
print(sp_sw_opt_clf.score(spheroidite_widmanstatten_test_feature[4],spheroidite_widmanstatten_test_label))
print(1 - sp_pear_opt_clf.score(np.concatenate((spheroidite_test_feature[layer -
    1],spheroidite_widmanstatten_test_feature[layer - 1])),
                        list(spheroidite_test_label) +
                            list(spheroidite_widmanstatten_test_label)))
print(np.concatenate((spheroidite_test_feature[layer -
    1],spheroidite_widmanstatten_test_feature[layer - 1])).shape)


print("-----")

network_pear_opt_clf = SVC()
network_pear_opt_clf.fit(np.concatenate((network_train_feature[layer -
    1],pearlite_train_feature[layer - 1])),
                      list(network_train_label) + list(pearlite_train_label))
print(network_pear_opt_clf.score(network_test_feature[4],network_test_label))
print(network_pear_opt_clf.score(pearlite_test_feature[4],pearlite_test_label))
print(1 - network_pear_opt_clf.score(np.concatenate((network_test_feature[layer - 1],
                                    pearlite_test_feature[layer - 1])),
                        list(network_test_label) + list(pearlite_test_label)))
print(np.concatenate((network_test_feature[4],pearlite_test_feature[4])).shape)
print("-----")

net_sw_opt_clf = SVC()
net_sw_opt_clf.fit(np.concatenate((network_train_feature[layer - 1],
                              spheroidite_widmanstatten_train_feature[layer - 1])),
                list(network_train_label) + list(spheroidite_widmanstatten_train_label))
print(net_sw_opt_clf.score(network_test_feature[4],network_test_label))
print(net_sw_opt_clf.score(spheroidite_widmanstatten_test_feature[4],spheroidite_widmanstatten_test_label))
print(1 - net_sw_opt_clf.score(np.concatenate((network_test_feature[layer - 1],
                                    spheroidite_widmanstatten_test_feature[layer -
                                        1])),
                        list(network_test_label) +
                            list(spheroidite_widmanstatten_test_label)))
print(np.concatenate((network_test_feature[4],spheroidite_widmanstatten_test_feature[4])).shape)
```

```
print("-----")
pear_sw_opt_clf = SVC()
pear_sw_opt_clf.fit(np.concatenate((pearlite_train_feature[layer - 1],
                              spheroidite_widmanstatten_train_feature[layer - 1])),
                list(pearlite_train_label) + list(spheroidite_widmanstatten_train_label))
print(pear_sw_opt_clf.score(pearlite_test_feature[4],pearlite_test_label))
print(pear_sw_opt_clf.score(spheroidite_widmanstatten_test_feature[4],spheroidite_widmanstatten_test_label))
print(1 - pear_sw_opt_clf.score(np.concatenate((pearlite_test_feature[layer - 1],
                                       spheroidite_widmanstatten_test_feature[layer -
                                           1])),
                         list(pearlite_test_label) +
                             list(spheroidite_widmanstatten_test_label)))
print(np.concatenate((pearlite_test_feature[4],spheroidite_widmanstatten_test_feature[4])).shape)




# (b) calculate test set error for multi-label one vs one classifier
multilabel_train_feature =
    list([spheroidite_train_feature[4],network_train_feature[4],pearlite_train_feature[4],
spheroidite_widmanstatten_train_feature[4]])
multilabel_train_label =
    list([spheroidite_train_label,network_train_label,pearlite_train_label,spheroidite_widmanstatten_train_la

np.concatenate(multilabel_train_feature)

np.concatenate(multilabel_train_label).shape

multilabel_test_feature =
    list([spheroidite_test_feature[4],network_test_feature[4],pearlite_test_feature[4],
spheroidite_widmanstatten_test_feature[4]])
multilabel_test_label =
    list([spheroidite_test_label,network_test_label,pearlite_test_label,spheroidite_widmanstatten_test_label]

np.concatenate(multilabel_test_feature).shape

np.concatenate(multilabel_test_label).shape

multi_label_clf = ovoclf(SVC())
multi_label_clf.fit(np.concatenate(multilabel_train_feature),np.concatenate(multilabel_train_label))

print(1 -
    multi_label_clf.score(np.concatenate(multilabel_test_feature),np.concatenate(multilabel_test_label)))
print(np.concatenate(multilabel_test_feature).shape)

# (c)

pearlite_spheroidite_test_label = path.iloc[pearlite_spheroidite_test_idx,1]
pearlite_widmanstatten_test_label = path.iloc[pearlite_widmanstatten_test_idx,1]

c_predict_all =
    multi_label_clf.predict(np.concatenate((pearlite_spheroidite_test_feature[4],pearlite_widmanstatten_test_

c_label_all =
    np.concatenate((pearlite_spheroidite_test_label,pearlite_widmanstatten_test_label))

c_result_all = pd.DataFrame(np.array((c_predict,c_label)).T)
c_result_all;

c_label_all =
    np.concatenate((pearlite_spheroidite_test_label,pearlite_widmanstatten_test_label))

c_result_all = pd.DataFrame(np.array((c_predict,c_label)).T)
```

```
c_result_all

c_predict_ps = multi_label_clf.predict(pearlite_spheroidite_test_feature[4])
c_result_ps = pd.DataFrame(np.array((c_predict_ps,pearlite_spheroidite_test_label)).T)
c_result_ps
c_result_ps[0].value_counts()

c_predict_pw = multi_label_clf.predict(pearlite_widmanstatten_test_feature[4])
c_result_pw = pd.DataFrame(np.array((c_predict_pw,pearlite_widmanstatten_test_label)).T)
c_result_pw[0].value_counts()

# d
d_predict = sp_pear_opt_clf.predict(pearlite_spheroidite_test_feature[4])

d_label = path.iloc[pearlite_spheroidite_test_idx,1]

d_result = pd.DataFrame(np.array((d_predict,d_label)).T)
d_result[0].value_counts()

#e

e_predict = multi_label_clf.predict(martensite_test_feature[4])
e_label = path.iloc[martensite_test_idx,1]
e_result = pd.DataFrame(np.array((e_predict,e_label)).T)
e_result[0].value_counts()
```