Objective of the Analysis:

The goal for this project is to improve on the previous linear regression model trained by introducing dimensionality reduction to predict the final price of a house given a set of features. In the model trained in our earlier project, a polynomial regression with degree three and above was taking a long time to converge for our dataset with 294 features. The same time issue was also persistent with the lasso regression where it was taking a lot of iterations (thus, more time) to converge on the optimal values. By introducing dimensionality reduction, will aim to reduce the number of features in our dataset and at the same time match our scores from earlier models if not better them, thus, reducing the time it takes to train the model on our dataset.

About the Data:

The Ames Housing Dataset is used compiled by Dean DE Cock which was available with this course and on Kaggle. This dataset has a total of 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa. The “SalePrice” variable is our target variable in the Ames Housing Dataset. Sample view of the dataset:

	1stFlrSF	2ndFlrSF	3SsnPorch	Alley	BedroomAbvGr	BldgType	BsmtCond	BsmtExposure	BsmtFinSF1	BsmtFinSF2	...	ScreenPorch	Street
0	856.0	854.0	0.0	None	3	1Fam	TA	No	706.0	0.0	...	0.0	Pave
1	1262.0	0.0	0.0	None	3	1Fam	TA	Gd	978.0	0.0	...	0.0	Pave
2	920.0	866.0	0.0	None	3	1Fam	TA	Mn	486.0	0.0	...	0.0	Pave
3	961.0	756.0	0.0	None	3	1Fam	Gd	No	216.0	0.0	...	0.0	Pave
4	1145.0	1053.0	0.0	None	4	1Fam	TA	Av	655.0	0.0	...	0.0	Pave

Data Exploration:

The dataset comprises of 1379 observations. There are lot of columns with categorical values in the dataset which needed to hot encoded so that they can be used in the linear regression model using OneHotEncoder function. After hot encoding, a total of 215 new features were created and in total we have 295 features.

```
data_ohc.dtypes.value_counts()
```

```
float64    279
int64       16
dtype: int64
```

The range of values for the numerical columns were very varied, hence, they were scaled using the StandardScaler() function. Some of the highly correlated features in the dataset:

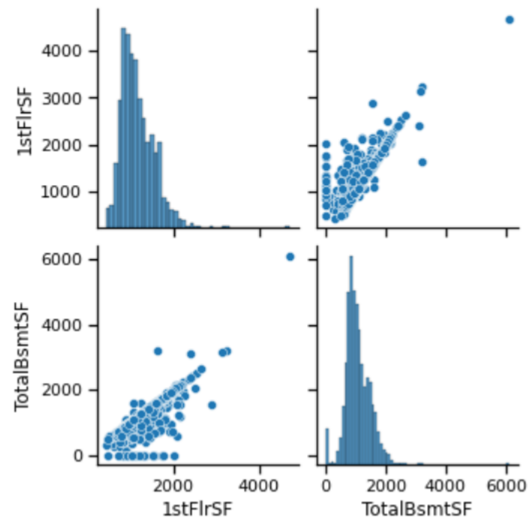
```
# Pairwise maximal correlations
corr_mat.abs().idxmax()
```

```
1stFlrSF            TotalBsmtSF
2ndFlrSF           HouseStyle_5
3SsnPorch          Foundation_5
BedroomAbvGr        TotRmsAbvGrd
BsmtFinSF1        BsmtFinType1_4
```

The above correlation is quite evident from the scatterplot between “1stFlrSF” and “TotalBsmtSF”:

```
sns.set_context('notebook')
sns.pairplot(data_ohc[['1stFlrSF','TotalBsmtSF']])
```

```
<seaborn.axisgrid.PairGrid at 0x18c82414a30>
```



<u>Summary of the Models:</u>

From the previous models, we had three models with the below R2 scores

	Vanilla Linear Regression	Ridge Regression	Lasso Regression
R2_Score	0.852526	0.852254	0.833682

Now dimensionality reduction is introduced for each of these three models in order to reduce the number of features while at the same time trying to keep the R2 scores similar if not better and also, to improve the time taken to train a model. First up, is the Principal Component Analysis (PCA) algorithm with below compiled data associated with it. From the examining the data we can see that the R2 scores for these models for the number of features between 50 to 200 is similar to the earlier trained models. In fact, for lasso we can see an improvement in the R2 scores with lesser features.

PCA Components	Vanilla Linear Regression	Ridge Regression	Lasso Regression
10	0.776064	0.775136	0.782150
20	0.810326	0.809765	0.808390
50	0.828226	0.829748	0.829263
100	0.831131	0.825717	0.824464
150	0.835824	0.834188	0.833651
200	0.844269	0.850232	0.846562
250	0.859923	0.859913	0.861872

R2 Scores

Other dimensionality reduction techniques were also employed for the three models namely Kernel PCA (to account for the non-linearity on our dataset) and Multi-Dimensional Scaling (MDS). Below is the summary of the R2 scores for each of them.

Kernel_PCA Components	Vanilla Linear Regression	Ridge Regression	Lasso Regression
10	0.779268	0.780575	0.781032
20	0.808067	0.810062	0.804942
50	0.828672	0.830221	0.831606
100	0.825936	0.830295	0.822153
150	0.835055	0.832702	0.830045
200	0.848816	0.844229	0.849283
250	0.860169	0.862829	0.863916

R2 Scores

MDS Components	Vanilla Linear Regression	Ridge Regression	Lasso Regression
10	0.780362	0.781218	0.776274
20	0.811535	0.803730	0.807987
50	0.830183	0.834932	0.832127
100	0.827381	0.828042	0.830848
150	0.828866	0.837266	0.835766
200	0.848715	0.849079	0.848108
250	0.862241	0.858613	0.861975

R2 Scores

Comparing the R2 scores across all the dimensionality reduction techniques, the Kernel PCA gives a more consistent and higher R2 scores at lower dimension for all the three models mainly because of its ability to account for the non-linearity in the dataset.

Also, we have increased the degree of the polynomial regression while employing the Kernel PCA to keep the overall time required to train the model low, since, lesser the number of features(less bias), less time to train the model. Below is the summary of the R2 scores for the same.

KernelPCA Components(Poly Degree=10)	Vanilla Linear Regression	Ridge Regression	Lasso Regression
10	0.776336	0.777637	0.782271
20	0.808678	0.810475	0.803051
50	0.822827	0.824773	0.832903
100	0.825182	0.828006	0.830524
150	0.837740	0.833111	0.836450
200	0.847071	0.847637	0.845473
250	0.859036	0.858797	0.858880

R2 Scores

KernelPCA Components(Poly Degree=7)	Vanilla Linear Regression	Ridge Regression	Lasso Regression
10	0.777810	0.780624	0.779670
20	0.806935	0.805864	0.805289
50	0.825603	0.828123	0.833140
100	0.826611	0.825340	0.827079
150	0.834817	0.833373	0.834536
200	0.847661	0.851185	0.843175
250	0.860482	0.861138	0.860581

R2 Scores

Final Model:

The objective was to predict the optimal sale price of a house given a set of features; hence, I have chosen the model based on Ridge regression for the purpose with the number components in the Kernel PCA equal to 150. This model gives a good R2 score while also reducing the number of features in our dataset.

Summary Key Findings and Insights:

The top 5 factors most influencing the sale price of the houses were

- GrLivArea (+25311.35) - Above grade (ground) living area square feet
- RoofMatl (-17350.74) - Roof Material
- OverallQual (+9358.69) - Rates the overall material and finish of the house
- Condition2 (-8856.07) - Proximity to various conditions (if more than one is present)
- YearBuilt (+8569.32) - Original construction date

The top features that added value to the price of a house were living area, materials used and finish of the house and the year it was built. As per the findings, per unit increase in the living area led to an increase of 25k (in US dollars) valuation in the price of the house. Older houses command lesser price as compared to the newer ones. However, roof materials and proximity to rail networks (Condition2) led to a decrease in the valuation of the houses.

Also, many of the features were zeroed out by the lasso’s feature selection process namely type of sale, availability of central air conditioning, pool quality, height of the basement, etc. which showed that these feature do not have a bearing on the overall price of a house.

Next Steps:

In order to further improve the model to better predict the final sale price of the house, would like to train the model on a bigger dataset and reduce the multicollinearity present in the dataset.

In [111]:

```
import pandas as pd
import numpy as np
from pathlib import Path
from sklearn.model_selection import KFold,train_test_split,cross_val_predict
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.pipeline import Pipeline
import seaborn as sns

# Import the data using the file path
filepath = Path('C:\\')/'Users'/'sranjanbehera'/'Documents'/'ML Algorithms'/'Linear Regression'/'Ames_Housing_Sales.csv'
#filepath = os.sep.join(data_path + ['Ames_Housing_Sales.csv'])

data = pd.read_csv(filepath)

print(data.shape)
```

(1379, 80)

In [101]:

```
data.head()
```

Out[101]:

	1stFlrSF	2ndFlrSF	3SsnPorch	Alley	BedroomAbvGr	BldgType	BsmtCond	BsmtExposure
0	856.0	854.0	0.0	None	3	1Fam	TA	No
1	1262.0	0.0	0.0	None	3	1Fam	TA	Gd
2	920.0	866.0	0.0	None	3	1Fam	TA	Mn
3	961.0	756.0	0.0	None	3	1Fam	Gd	No
4	1145.0	1053.0	0.0	None	4	1Fam	TA	Av

5 rows × 80 columns

In [2]:
```
# Select the object (string) columns
mask = data.dtypes == object
categorical_cols = data.columns[mask]

# Determine how many extra columns would be created
num_ohc_cols = (data[categorical_cols]
                .apply(lambda x: x.nunique())
                .sort_values(ascending=False))


# No need to encode if there is only one value
small_num_ohc_cols = num_ohc_cols.loc[num_ohc_cols>1]

# Number of one-hot columns is one less than the number of categories
small_num_ohc_cols -= 1

# This is 215 columns, assuming the original ones are dropped.
# This is quite a few extra columns!
small_num_ohc_cols.sum()
```

Out[2]: 215

In [3]:
```
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# Copy of the data
data_ohc = data.copy()

# The encoders
le = LabelEncoder()
ohc = OneHotEncoder()

for col in num_ohc_cols.index:

    # Integer encode the string categories
    dat = le.fit_transform(data_ohc[col]).astype(int)

    # Remove the original column from the dataframe
    data_ohc = data_ohc.drop(col, axis=1)

    # One hot encode the data--this returns a sparse array
    new_dat = ohc.fit_transform(dat.reshape(-1,1))

    # Create unique column names
    n_cols = new_dat.shape[1]
    col_names = ['_'.join([col, str(x)]) for x in range(n_cols)]

    # Create the new dataframe
    new_df = pd.DataFrame(new_dat.toarray(),
                          index=data_ohc.index,
                          columns=col_names)

    # Append the new data to the dataframe
    data_ohc = pd.concat([data_ohc, new_df], axis=1)
```

In [4]:
```
data_ohc.dtypes.value_counts()
```

Out[4]:
```
float64    279
int64       16
dtype: int64
```

In [107]:
```
float_columns = [x for x in data_ohc.columns if x not in ['SalePrice']]

# The correlation matrix
corr_mat = data_ohc[float_columns].corr()

# Strip out the diagonal values for the next step
for x in range(len(float_columns)):
    corr_mat.iloc[x,x] = 0.0

corr_mat
```

Out[107]:

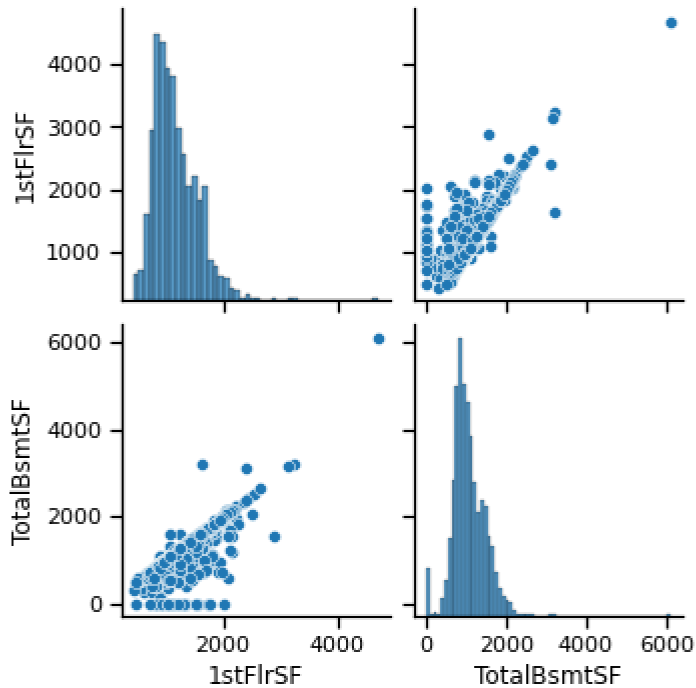
	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BsmtFinSF1	BsmtFinSF2	Bsr
1stFlrSF	0.000000	-0.223710	0.053200	0.104870	0.446596	0.094006	
2ndFlrSF	-0.223710	0.000000	-0.026654	0.507574	-0.142969	-0.106641	
3SsnPorch	0.053200	-0.026654	0.000000	-0.026018	0.023903	-0.031442	
BedroomAbvGr	0.104870	0.507574	-0.026018	0.000000	-0.118036	-0.007734	
BsmtFinSF1	0.446596	-0.142969	0.023903	-0.118036	0.000000	-0.054966	
...	...	...	...	...	...	...	
CentralAir_1	0.127758	-0.028869	0.027484	-0.001302	0.138010	0.024623	
Street_0	0.001851	-0.043936	-0.007224	-0.035803	0.006146	0.045426	
Street_1	-0.001851	0.043936	0.007224	0.035803	-0.006146	-0.045426	
Utilities_0	-0.011619	0.021668	0.003226	-0.004636	0.020200	-0.050166	
Utilities_1	0.011619	-0.021668	-0.003226	0.004636	-0.020200	0.050166	

294 rows × 294 columns

In [108]:
```
# Pairwise maximal correlations
corr_mat.abs().idxmax()
```

Out[108]:
```
1stFlrSF          TotalBsmtSF
2ndFlrSF         HouseStyle_5
3SsnPorch        Foundation_5
BedroomAbvGr     TotRmsAbvGrd
BsmtFinSF1      BsmtFinType1_4
                     ...
CentralAir_1     CentralAir_0
Street_0             Street_1
Street_1             Street_0
Utilities_0       Utilities_1
Utilities_1       Utilities_0
Length: 294, dtype: object
```

In [119]:
```
sns.set_context('notebook')
sns.pairplot(data_ohc[['1stFlrSF','TotalBsmtSF']]);
```



In [6]:
```
X=data_ohc.drop('SalePrice',axis=1)
y=data_ohc['SalePrice']
```

In [7]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.3,random_sta
te=10987)
X_test.shape
```

Out[7]: (414, 294)

In [74]:
```
from sklearn.decomposition import PCA,KernelPCA
from sklearn.model_selection import StratifiedShuffleSplit
```

In [120]:
```
def predict_price(model):
    pipe=[('poly',PolynomialFeatures(degree=2,include_bias=False)),
          ('sc',StandardScaler()),('training_model',model)]
    pipe=Pipeline(pipe)

    pipe.fit(X_train, y_train)

    y_pred=pipe.predict(X_test)

    return r2_score(y_test,y_pred)
```

In [122]:
```
#Linear Regression
predict_price(LR)
```

Out[122]: 0.8525263289929896

In [123]:
```
#Ridge Regression
predict_price(RR)
```

Out[123]: 0.8522541275415596

In [124]:
```
#Lasso Regression
predict_price(LassoR)
```

```
C:\Users\sranjanbehera\Anaconda3\lib\site-packages\sklearn\linear_model\_coor
dinate_descent.py:530: ConvergenceWarning: Objective did not converge. You mi
ght want to increase the number of iterations. Duality gap: 5149176151.01836
9, tolerance: 629732213.4051085
  model = cd_fast.enet_coordinate_descent(
```

Out[124]: 0.8336817743833744

In [129]:
```
r2_scores_previous=list()

for lab,mod in zip(coeff_labels, coeff_models):
    r2_sco=predict_price(mod)
    r2_scores_previous.append(pd.Series({'R2_Score':r2_sco},name=lab))

r2_scores_previous = pd.concat(r2_scores_previous, axis=1)
r2_scores_previous
```

```
C:\Users\sranjanbehera\Anaconda3\lib\site-packages\sklearn\linear_model\_coor
dinate_descent.py:530: ConvergenceWarning: Objective did not converge. You mi
ght want to increase the number of iterations. Duality gap: 5149176151.01836
9, tolerance: 629732213.4051085
  model = cd_fast.enet_coordinate_descent(
```

Out[129]:

	Vanilla Linear Regression	Ridge Regression	Lasso Regression
R2_Score	0.852526	0.852254	0.833682

In [13]:
```
def pca_predict_price(n,model):
    pipe=[('poly',PolynomialFeatures(degree=2,include_bias=False)),
          ('sc',StandardScaler()),('pca',PCA(n_components=n)),('training_mode
l',model)]
    pipe=Pipeline(pipe)

    pipe.fit(X_train, y_train)

    y_pred=pipe.predict(X_test)

    return r2_score(y_test,y_pred)
```

In [14]:
```
pca_predict_price(100,LinearRegression(n_jobs=-1))
```

Out[14]: 0.8314079874553768

In [15]:
```
scores=[]
ns=[10, 20, 50, 100, 150, 200, 250]
scores=[pca_predict_price(n,LinearRegression(n_jobs=-1)) for n in ns]
scores
```

Out[15]:
```
[0.7799491305687045,
 0.8152067503632504,
 0.8241645416568837,
 0.8219599809307636,
 0.8356890244334294,
 0.8497792894237196,
 0.8626415423539533]
```

In [61]:
```
scores=[]
scores_data={}
ns=[10, 20, 50, 100, 150, 200, 250]

#Vanilla Linear Regression
LR=LinearRegression(n_jobs=-1)

#Ridge Regression
RR=Ridge(alpha=0.03792690190732246)

#Lasso Regression
LassoR=Lasso(alpha=50.68421052631579)


coeff_labels = ['Vanilla Linear Regression' , 'Ridge Regression' , 'Lasso Regr
ession']
coeff_models = [LR, RR, LassoR]

for lab,mod in zip(coeff_labels, coeff_models):
    r2_scores=list()
    for x in ns:
        r2_sco=pca_predict_price(x,mod)
        r2_scores.append(r2_sco)
    scores_data[lab]=r2_scores

r2_scores_df=pd.DataFrame(scores_data,columns=coeff_labels)

#r2_scores_df=pd.concat(r2_scores_df, axis=1)
```

In [ ]:
```
r2_scores_df['PCA Components']=ns
```

In [66]:
```
r2_scores_df.set_index('PCA Components',inplace=True)
r2_scores_df
```

Out[66]:

PCA Components	Vanilla Linear Regression	Ridge Regression	Lasso Regression
10	0.776064	0.775136	0.782150
20	0.810326	0.809765	0.808390
50	0.828226	0.829748	0.829263
100	0.831131	0.825717	0.824464
150	0.835824	0.834188	0.833651
200	0.844269	0.850232	0.846562
250	0.859923	0.859913	0.861872

In [68]:
```
def kernelpca_predict_price(n,model):
    pipe=[('poly',PolynomialFeatures(degree=2,include_bias=False)),
          ('sc',StandardScaler()),('kernel_pca',KernelPCA(n_components=n,kerne
l='rbf',gamma=0.01,n_job=-1)),
          ('training_model',model)]
    pipe=Pipeline(pipe)

    pipe.fit(X_train, y_train)

    y_pred=pipe.predict(X_test)

    return r2_score(y_test,y_pred)
```

In [69]:
```
scores=[]
scores_data={}
ns=[10, 20, 50, 100, 150, 200, 250]

for lab,mod in zip(coeff_labels, coeff_models):
    r2_scores=list()
    for x in ns:
        r2_sco=pca_predict_price(x,mod)
        r2_scores.append(r2_sco)
    scores_data[lab]=r2_scores

r2_scores_df2=pd.DataFrame(scores_data,columns=coeff_labels)

#r2_scores_df=pd.concat(r2_scores_df, axis=1)
```

In [71]:
```
r2_scores_df2['Kernel_PCA Components']=ns
r2_scores_df2.set_index('Kernel_PCA Components',inplace=True)
```

In [72]:
```
r2_scores_df2
```

Out[72]:

Kernel_PCA Components	Vanilla Linear Regression	Ridge Regression	Lasso Regression
10	0.779268	0.780575	0.781032
20	0.808067	0.810062	0.804942
50	0.828672	0.830221	0.831606
100	0.825936	0.830295	0.822153
150	0.835055	0.832702	0.830045
200	0.848816	0.844229	0.849283
250	0.860169	0.862829	0.863916

In [76]:
```
from sklearn.manifold import MDS
```

In [77]:
```
def mds_predict_price(n,model):
    pipe=[('poly',PolynomialFeatures(degree=2,include_bias=False)),
          ('sc',StandardScaler()),('mds',MDS(n_components=n,n_job=-1)),
          ('training_model',model)]
    pipe=Pipeline(pipe)

    pipe.fit(X_train, y_train)

    y_pred=pipe.predict(X_test)

    return r2_score(y_test,y_pred)
```

In [78]:
```
scores=[]
scores_data={}
ns=[10, 20, 50, 100, 150, 200, 250]

for lab,mod in zip(coeff_labels, coeff_models):
    r2_scores=list()
    for x in ns:
        r2_sco=pca_predict_price(x,mod)
        r2_scores.append(r2_sco)
    scores_data[lab]=r2_scores

r2_scores_df3=pd.DataFrame(scores_data,columns=coeff_labels)

#r2_scores_df=pd.concat(r2_scores_df, axis=1)
```

In [81]:
```
r2_scores_df3['MDS Components']=ns
r2_scores_df3.set_index('MDS Components',inplace=True)
```

In [82]:
```
r2_scores_df3
```

Out[82]:

	Vanilla Linear Regression	Ridge Regression	Lasso Regression
MDS Components			
10	0.780362	0.781218	0.776274
20	0.811535	0.803730	0.807987
50	0.830183	0.834932	0.832127
100	0.827381	0.828042	0.830848
150	0.828866	0.837266	0.835766
200	0.848715	0.849079	0.848108
250	0.862241	0.858613	0.861975

In [83]:
```
import datetime

curr=datetime.datetime.now()

for i in range(10000):
    x=1

print(datetime.datetime.now()-curr)
```

```
0:00:00.000996
```

In [102]:
```
def kernelpca_3deg_predict_price(n,model):
    pipe=[('poly',PolynomialFeatures(degree=10,include_bias=False)),
          ('sc',StandardScaler()),('kernel_pca',KernelPCA(n_components=n,kerne
l='rbf',gamma=0.01,n_job=-1)),
          ('training_model',model)]
    pipe=Pipeline(pipe)

    pipe.fit(X_train, y_train)

    y_pred=pipe.predict(X_test)

    return r2_score(y_test,y_pred)
```

In [103]:
```
scores=[]
scores_data={}
ns=[10, 20, 50, 100, 150, 200, 250]

for lab,mod in zip(coeff_labels, coeff_models):
    r2_scores=list()
    for x in ns:
        r2_sco=pca_predict_price(x,mod)
        r2_scores.append(r2_sco)
    scores_data[lab]=r2_scores

r2_scores_df4=pd.DataFrame(scores_data,columns=coeff_labels)

#r2_scores_df=pd.concat(r2_scores_df, axis=1)
```

In [104]:
```
r2_scores_df4['KernelPCA Components(Poly Degree=10)']=ns
r2_scores_df4.set_index('KernelPCA Components(Poly Degree=10)',inplace=True)
```

In [105]:
```
r2_scores_df4
```

Out[105]:

KernelPCA Components(Poly Degree=10)	Vanilla Linear Regression	Ridge Regression	Lasso Regression
10	0.776336	0.777637	0.782271
20	0.808678	0.810475	0.803051
50	0.822827	0.824773	0.832903
100	0.825182	0.828006	0.830524
150	0.837740	0.833111	0.836450
200	0.847071	0.847637	0.845473
250	0.859036	0.858797	0.858880

In [99]:
```
r2_scores_df4
```

Out[99]:

KernelPCA Components(Poly Degree=7)	Vanilla Linear Regression	Ridge Regression	Lasso Regression
10	0.777810	0.780624	0.779670
20	0.806935	0.805864	0.805289
50	0.825603	0.828123	0.833140
100	0.826611	0.825340	0.827079
150	0.834817	0.833373	0.834536
200	0.847661	0.851185	0.843175
250	0.860482	0.861138	0.860581

In [ ]: