Karthik Konath (kk28699), Kyle Polansky (kpp446)

EE 379K-DS Lab 4

Problem 1: PCA

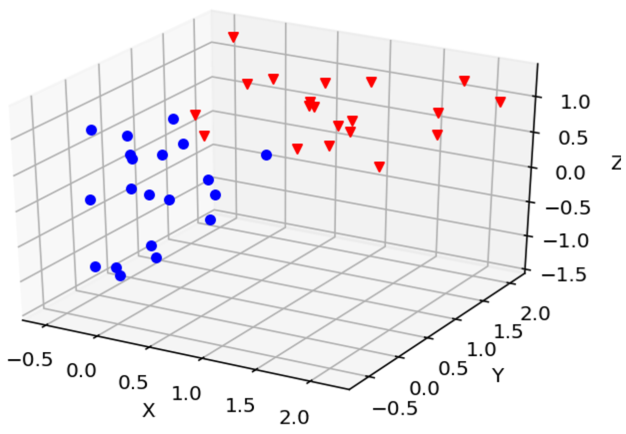
1.1 Generate 20 random points in d = 3 ... Create a 3D plot of the clouds of data points labeled with the two labels.

In [27]:
```
import numpy as np
mean=[0,0,0]
cov=[[0.5,0,0],[0,0.5,0],[0,0,0.7]]
rp1 = np.random.multivariate_normal(mean,cov,20)
print(rp1)

mean=[1,1,1]
cov=[[0.5,0,0],[0,0.5,0],[0,0,0.01]]
rp2 = np.random.multivariate_normal(mean,cov,20)
print(rp2)
```

```
[[-0.42260112 -0.18857163 -0.05253193]
 [ 0.11628831 -0.37790379  0.76360518]
 [ 0.15722341  0.33027791  0.64822288]
 [ 0.08903789 -0.35663238  0.80306633]
 [ 0.3579244   0.48690477 -0.07935387]
 [ 0.95946245  0.28995485  0.73498273]
 [-0.17256526 -0.16903128 -1.08299561]
 [-0.37109988  0.32000121  0.62628776]
 [-0.4682006  -0.06096256  0.86188894]
 [-0.5329082   0.33269546 -1.33697486]
 [ 0.38086277  0.34317545  0.20047105]
 [ 0.02509578  0.21701678  0.50904164]
 [-0.10626909  0.21482227 -0.08809754]
 [-0.40119341  0.76174155 -1.37159145]
 [ 0.72607062 -0.20579679  0.00323382]
 [ 0.11236199 -0.39860736  0.35315411]
 [ 0.42585862 -0.62435974 -0.23709363]
 [-0.05080734  0.51783564  0.85789056]
 [-0.2104911  -0.48329103 -0.794805  ]
 [ 0.51805064 -0.47970205  0.35762114]]
[[ 1.34346874  0.77088406  1.01227939]
 [ 0.45576646  0.16960397  0.91430263]
 [-0.37627463  2.03947519  1.25751951]
 [ 0.84795237  1.17396943  0.98213079]
 [ 0.94218752  1.09788336  1.0143513 ]
 [ 1.98229883  0.28239581  0.84829847]
 [ 0.85119155  1.18106058  1.03056432]
 [ 1.50042139 -0.16790258  1.17488469]
 [ 0.01907715  1.60491563  0.87957619]
 [ 2.15784468  0.91065656  1.03672731]
 [ 1.35113848  0.99472252  0.97574405]
 [ 1.53455758  0.27886123  1.0127976 ]
 [ 0.74879365  1.61564965  1.06923499]
 [ 1.02393671  1.90713192  1.03253789]
 [ 1.79349808  2.08417812  1.16917152]
 [ 2.14141094  2.06189532  0.97454375]
 [ 0.3083775   1.5246335   1.06111294]
 [ 1.55646106  0.58968324  1.07042976]
 [ 0.10928665  0.60316281  0.91950185]
 [ 1.73572541  1.74739295  0.8541466 ]]
```

1.2 What do the points look like?

```
In [28]: from mpl_toolkits.mplot3d import Axes3D
         import matplotlib.pyplot as plt
         n=0
         fig = plt.figure()
         ax = fig.add_subplot(111, projection='3d')
         while(n<len(rp1)):
             ax.scatter(rp1[n][0],rp1[n][1],rp1[n][2],c='b',marker='o')
             ax.scatter(rp2[n][0],rp2[n][1],rp2[n][2],c='r',marker='v')
             n+=1
         ax.set_xlabel('X')
         ax.set_ylabel('Y')
         ax.set_zlabel('Z')
         plt.show()
```



1.3 Concatenate all the points and ignore the labels. Find the covariance of this dataset.

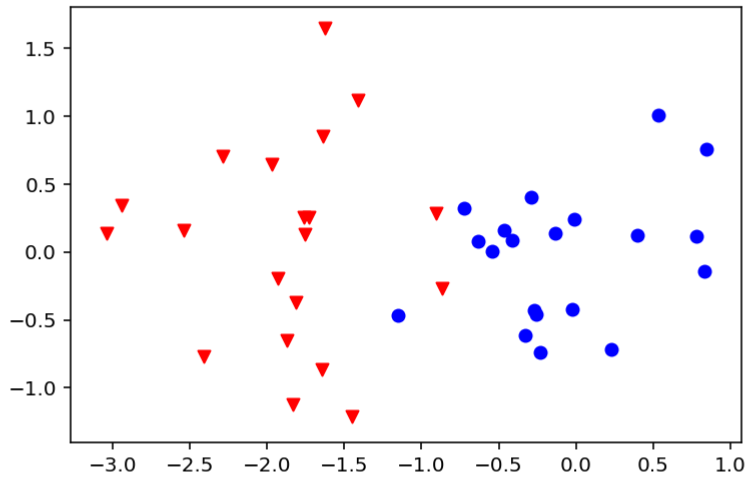
In [29]:
```
rp = np.concatenate([rp1,rp2])
#Manually calculate mean
rpSum = np.sum(rp, axis=0)
rpMean = np.divide(rpSum,rp.shape[0])

#Manually calculate covariance matrix
rpdiff = np.subtract(rp,rpMean)
rpdiffmultxy = np.multiply(rpdiff[:,0],rpdiff[:,1])
rpdiffmultsumxy = np.sum(rpdiffmultxy)
rpdiffmultyz = np.multiply(rpdiff[:,1],rpdiff[:,2])
rpdiffmultsumyz = np.sum(rpdiffmultyz)
rpdiffmultxz = np.multiply(rpdiff[:,0],rpdiff[:,2])
rpdiffmultsumxz = np.sum(rpdiffmultxz)
covarxy = np.divide(rpdiffmultsumxy,rp.shape[0]-1)
covaryz = np.divide(rpdiffmultsumyz,rp.shape[0]-1)
covarxz = np.divide(rpdiffmultsumxz,rp.shape[0]-1)
varx = np.divide(np.sum(np.square(rp[:,0]-rpMean[0])),rp.shape[0]-1)
vary = np.divide(np.sum(np.square(rp[:,1]-rpMean[1])),rp.shape[0]-1)
varz = np.divide(np.sum(np.square(rp[:,2]-rpMean[2])),rp.shape[0]-1)
COV = [[varx,covarxy,covarxz],[covarxy,vary,covaryz],[covarxz,covaryz,varz]]
print("COV: ", COV)
print("Real Cov", np.cov(rp,rowvar=False))
```

```
COV:  [[0.6242789891771622, 0.24158701472784883, 0.2965057056917436], [0.2415
8701472784883, 0.6208390999162904, 0.2555187595234695], [0.2965057056917436,
0.2555187595234695, 0.4844114164850418]]
Real Cov [[0.62427899 0.24158701 0.29650571]
 [0.24158701 0.6208391  0.25551876]
 [0.29650571 0.25551876 0.48441142]]
```

1.4 Find the two eigenvectors of the covariance matrix with the largest eigenvalues. Project the data points on these two vectors and show the two dimensional plot with the clouds of points. Also show the labels of the points. Did PCA make it easier to distinguish the two labels in two dimensions?

In [30]:
```
w,v = np.linalg.eig(COV);
print("Eigenvalues: ", w)
print("Eigenvectors: ",v)
min = 0
if(abs(w[1])<abs(w[min])):
    min=1
if(abs(w[2])<abs(w[min])):
    min=2
w = np.delete(w,min)
v= np.asarray(np.delete(v,min,1))
print("Max Eigenvalues: " , str(w[0]) , str(w[1]))
print("Corresponding Eigenvectors: ", v)

rp1 = rp1.dot(v)
rp2 = rp2.dot(v)

print(rp1)

n=0
while(n<len(rp1)):
    plt.scatter(rp1[n][0],rp1[n][1],c='b',marker='o')
    plt.scatter(rp2[n][0],rp2[n][1],c='r',marker='v')
    n+=1
plt.show()
```

```
Eigenvalues:  [1.10873635 0.3844608  0.23633236]
Eigenvectors:  [[-0.61542574 -0.62317164 -0.4826057 ]
 [-0.58270214  0.77203039 -0.2538253 ]
 [-0.530763   -0.12500475  0.83825083]]
Max Eigenvalues:  1.1087363467204001 0.3844607964138418
Corresponding Eigenvectors:  [[-0.61542574 -0.62317164]
 [-0.58270214  0.77203039]
 [-0.530763   -0.12500475]]
[[ 0.3978427   0.12433674]
 [-0.25665485 -0.45967506]
 [-0.6332657   0.07597648]
 [-0.27322365 -0.43120403]
 [-0.46187824  0.16277656]
 [-1.14953684 -0.46593216]
 [ 0.77951     0.11242009]
 [-0.29049135  0.40002063]
 [-0.13379304  0.13696417]
 [ 0.84371986  0.75607249]
 [-0.54076444  0.00253917]
 [-0.4120812   0.08827195]
 [-0.01301775  0.24308582]
 [ 0.53102632  1.00955543]
 [-0.32864072 -0.61175223]
 [-0.02432223 -0.42190374]
 [ 0.22757192 -0.71776988]
 [-0.72581236  0.32420615]
 [ 0.83300945 -0.14258888]
 [-0.22911036 -0.73788337]]
```



Yes, now it is very easy to draw a cluster around the two separate clusters

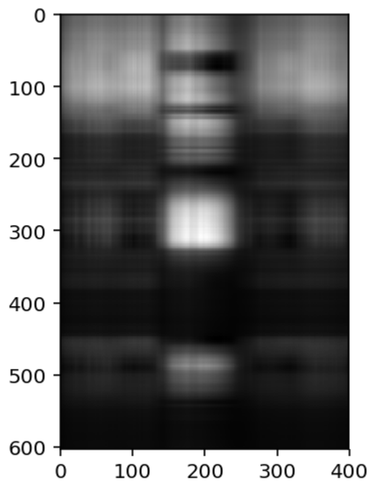
Problem 2: Low rank approximation of Mona Lisa

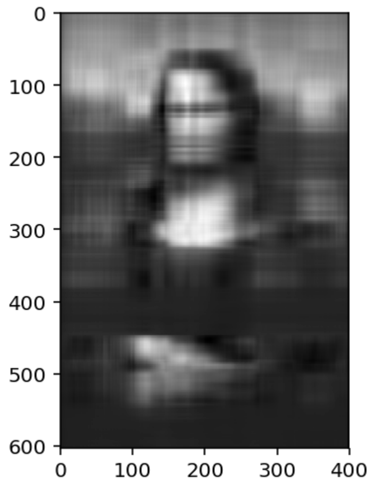
2.1 Load the Mona Lisa image (in grayscale) and treat it as a matrix M. Perform a singular value decomposition on this matrix usinglinalg.svd. You can perform a low-rank approximationby zeroing out singular values and keeping only the top k. Show the best rank k= 2, k= 5 and k= 10 approximation to Mona Lisa.

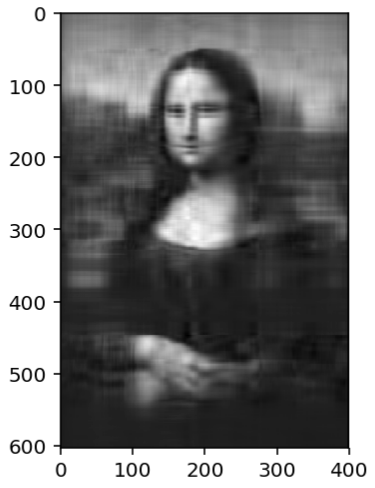
In [31]:
```
from scipy.misc import imread

#Read Image
image = imread('mona_lisa.png', flatten=True)
plt.gray()

#SVG
u, s, vh = np.linalg.svd(image, full_matrices='true')

#Zero out singular values
def approx(rank):
    u_k = np.array([ u[:, i] for i in range(rank)]).T
    vh_k = np.array([vh[j] for j in range(rank)])
    s_k = np.diag(s)[:rank, :rank]
    return np.dot(np.dot(u_k, s_k), vh_k)
```

```
<matplotlib.figure.Figure at 0x166f7c7db38>
```

In [32]:
```
ranks = [2, 5, 10]
for k in ranks:
    plt.imshow(approx(k))
    plt.show()
```







2.2 If each pixel is represented by two bytes, how many bits is your compressed Mona Lisa foreach of those k rank approximations?

The formula for calculating image size is: 16[(hk) + k + (wk)]

Where the formula is the size of the SVD results multiplied by the size per pixel
h = height of image
w = width of image
k = kth rank
16 = 2 bytes in bits

In [33]:
```
h = image.shape[0]
w = image.shape[1]

for k in ranks:
    print("Size of rank(%s): %s" % (k, 16*((h*k) + k + (w*k))))
```

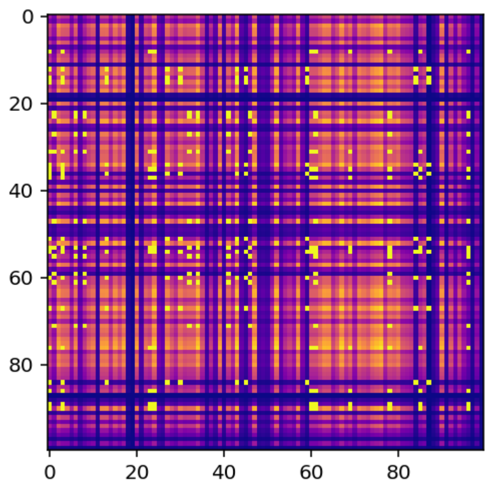
```
Size of rank(2): 32128
Size of rank(5): 80320
Size of rank(10): 160640
```

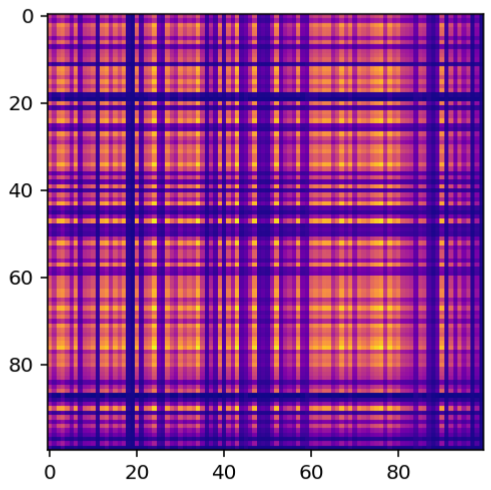
Problem 3: Using Low Rank Structure for Corrupted Entries

Download filesCorrMat1.csv and CorrMat3.csv from Canvas. These are each 100 by 100 matrices. Look at the data and find which entries are corrupted. Then try to correct these corrupted entries. Explain your approach. (Hint: The corrupted entries have values that are completely out of the range of the others. This should help you identify which are the corrupted ones. For completing them, the hint is that we have been talking about PCA, low rank matrices and low-rankapproximations.)

```
In [34]: import heapq
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

plt.rcParams['image.cmap'] = 'plasma'

#Import data
df1 = pd.read_csv("CorrMat1.csv", header = None)
df3 = pd.read_csv("CorrMat3.csv", header = None)

def filter(df):
    #Show data
    plt.imshow(df)
    plt.show()

    #PCA decomposition
    pca = PCA(n_components=1)
    pca.fit(df)

    #Show filtered through PCA
    df_reduced = pca.fit_transform(df)
    df_filtered = pca.inverse_transform(df_reduced)
    plt.imshow(df_filtered)
    plt.show()

print("DF1 Before/After:")
filter(df1)
```

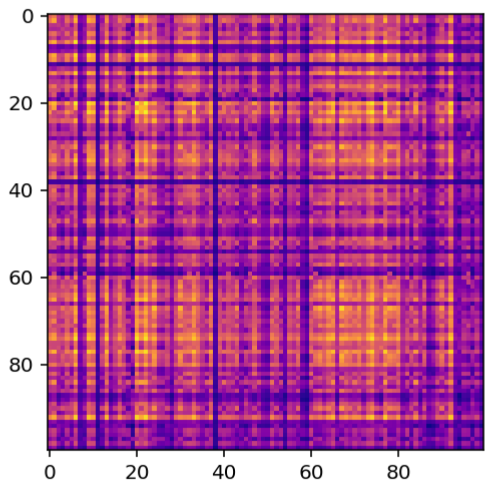
```
DF1 Before/After:
```

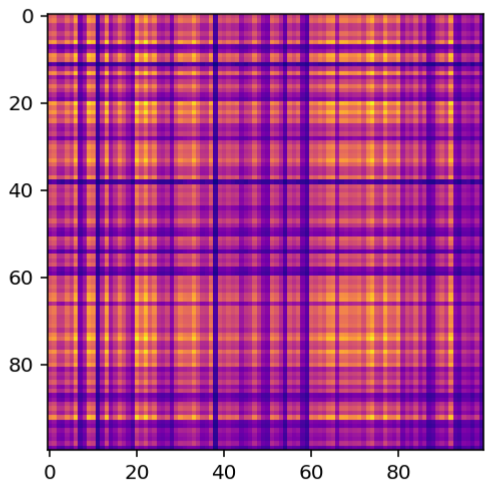




In [35]:
```
print("DF3 Before/After:")
filter(df3)
```

```
DF3 Before/After:
```





Problem 4: Getting more into Kaggle

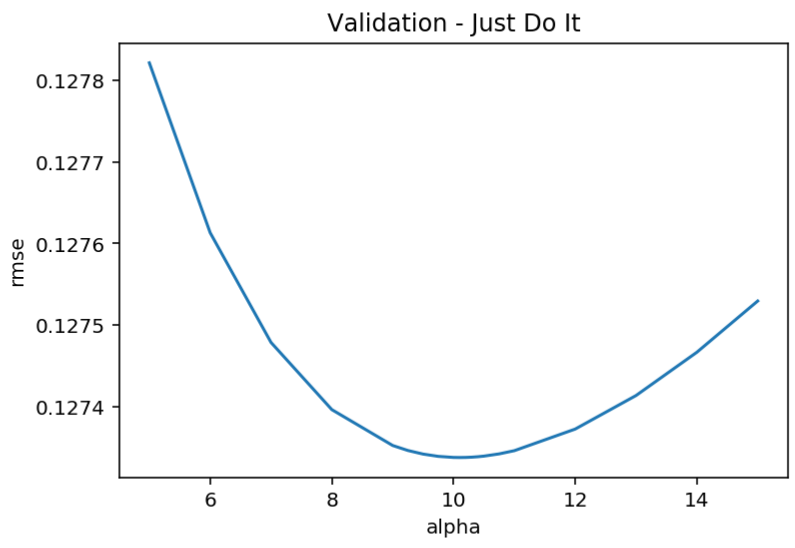
4.2 Train a ridge regression and a lasso regression model. Optimize the alphas using cross validation. What is the best score you can get from a single ridge regression model and from asingle lasso model?

In [36]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib

from scipy.stats import skew
from scipy.stats.stats import pearsonr


%config InlineBackend.figure_format = 'retina' #set 'png' here when working
 on notebook
%matplotlib inline

train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                      test.loc[:,'MSSubClass':'SaleCondition']))

#log transform the target:
train["SalePrice"] = np.log1p(train["SalePrice"])

#log transform skewed numeric features:
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna())) #compu
te skewness
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index

all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

all_data = pd.get_dummies(all_data)

#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())

#creating matrices for sklearn:
X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y = train.SalePrice
```

In [37]:
```
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV, LassoLar
sCV, LinearRegression, LogisticRegression
from sklearn.model_selection import cross_val_score

def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, y, scoring="neg_mean_square
d_error", cv = 5))
    return(rmse)
```

```
In [38]: #Ridge
         model_ridge = Ridge()

         alphas = [5, 6, 7, 8, 9, 9.25, 9.5, 9.75, 10, 10.1, 10.2, 10.3, 10.4, 10.5, 1
         0.75, 11, 12, 13, 14, 15]
         cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean()
                     for alpha in alphas]

         cv_ridge = pd.Series(cv_ridge, index = alphas)

         cv_ridge.plot(title = "Validation - Just Do It")
         plt.xlabel("alpha")
         plt.ylabel("rmse")

         print(cv_ridge)
         print("Ridge min error: %s" % str(cv_ridge.min()))
```

```
5.00     0.127822
6.00     0.127613
7.00     0.127479
8.00     0.127396
9.00     0.127352
9.25     0.127346
9.50     0.127341
9.75     0.127339
10.00    0.127337
10.10    0.127337
10.20    0.127337
10.30    0.127338
10.40    0.127338
10.50    0.127339
10.75    0.127342
11.00    0.127346
12.00    0.127372
13.00    0.127413
14.00    0.127466
15.00    0.127529
dtype: float64
Ridge min error: 0.12733723279715922
```



In [39]:
```
model_ridge = Ridge(alpha = 10.1).fit(X_train, y)
print("Ridge min error: %s" % str(rmse_cv(model_ridge).mean()))
ridge_coef = pd.Series(model_ridge.coef_, index = X_train.columns)
```

```
Ridge min error: 0.12733723279715922
```

In [40]:
```
#Lasso
model_lasso = LassoCV(alphas = [0.1, 0.01, 0.005, 0.001, 0.0005]).fit(X_train,
 y)

print("Lasso min error: %s" % str(rmse_cv(model_lasso).mean()))
```

```
Lasso min error: 0.12314421090977441
```

4.3 Plot the l_0 norm (number of nonzeros) of the coefficients that lasso produces as you vary alpha.

In [41]:
```
lasso_coef = pd.Series(model_lasso.coef_, index = X_train.columns)
print(lasso_coef)
print("Lasso non-zero coefficients: %s" % sum(lasso_coef != 0))
```

```
MSSubClass              -0.007480
LotFrontage              0.000000
LotArea                  0.071826
OverallQual              0.053160
OverallCond              0.043027
YearBuilt                0.001777
YearRemodAdd             0.000722
MasVnrArea              -0.000194
BsmtFinSF1               0.009292
BsmtFinSF2              -0.001385
BsmtUnfSF               -0.003975
TotalBsmtSF              0.019208
1stFlrSF                 0.030350
2ndFlrSF                -0.003396
LowQualFinSF            -0.003880
GrLivArea                0.400009
BsmtFullBath             0.025674
BsmtHalfBath             0.000000
FullBath                 0.021307
HalfBath                 0.013793
BedroomAbvGr            -0.001004
KitchenAbvGr            -0.009937
TotRmsAbvGrd             0.005079
Fireplaces               0.022117
GarageYrBlt              0.000029
GarageCars               0.038789
GarageArea               0.000051
WoodDeckSF               0.004409
OpenPorchSF              0.001631
EnclosedPorch            0.003123
                           ...
GarageCond_TA            0.000000
PavedDrive_N            -0.000000
PavedDrive_P            -0.000000
PavedDrive_Y             0.005346
PoolQC_Ex                0.000000
PoolQC_Fa               -0.000000
PoolQC_Gd               -0.000000
Fence_GdPrv              0.000000
Fence_GdWo              -0.016323
Fence_MnPrv              0.000000
Fence_MnWw              -0.000000
MiscFeature_Gar2        -0.000000
MiscFeature_Othr        -0.000000
MiscFeature_Shed         0.000000
MiscFeature_TenC        -0.000000
SaleType_COD            -0.011456
SaleType_CWD             0.000000
SaleType_Con             0.000000
SaleType_ConLD           0.000000
SaleType_ConLI          -0.000000
SaleType_ConLw          -0.000000
SaleType_New             0.021575
SaleType_Oth             0.000000
SaleType_WD             -0.030656
SaleCondition_Abnorml   -0.047116
SaleCondition_AdjLand    0.000000
```

```
SaleCondition_Alloca    -0.000000
SaleCondition_Family    -0.007925
SaleCondition_Normal     0.019666
SaleCondition_Partial    0.000000
Length: 288, dtype: float64
Lasso non-zero coefficients: 111
```

4.4 Add the outputs of your models as features and train a ridge regression on all the features plus the model outputs (This is called Ensembling and Stacking). Be careful not to overfit. What score can you get?

In [42]:
```
cols = X_train.columns.values
feature_dataframe = pd.DataFrame({'Ridge': ridge_coef, 'Lasso': lasso_coef})
```

In [43]:
```
print(feature_dataframe)
```

```
                         Lasso     Ridge
MSSubClass           -0.007480 -0.012809
LotFrontage           0.000000 -0.001896
LotArea               0.071826  0.075529
OverallQual           0.053160  0.052296
OverallCond           0.043027  0.039278
YearBuilt             0.001777  0.001311
YearRemodAdd          0.000722  0.000696
MasVnrArea           -0.000194 -0.001044
BsmtFinSF1            0.009292  0.010452
BsmtFinSF2           -0.001385 -0.002433
BsmtUnfSF            -0.003975 -0.004552
TotalBsmtSF           0.019208  0.030520
1stFlrSF              0.030350  0.122219
2ndFlrSF             -0.003396  0.007740
LowQualFinSF         -0.003880  0.000195
GrLivArea             0.400009  0.205824
BsmtFullBath          0.025674  0.025758
BsmtHalfBath          0.000000  0.001273
FullBath              0.021307  0.036846
HalfBath              0.013793  0.026862
BedroomAbvGr         -0.001004  0.004908
KitchenAbvGr         -0.009937 -0.033588
TotRmsAbvGrd          0.005079  0.011173
Fireplaces            0.022117  0.016189
GarageYrBlt           0.000029 -0.000159
GarageCars            0.038789  0.047964
GarageArea            0.000051  0.000021
WoodDeckSF            0.004409  0.003906
OpenPorchSF           0.001631  0.001975
EnclosedPorch         0.003123  0.003551
...                        ...       ...
GarageCond_TA         0.000000  0.010323
PavedDrive_N         -0.000000 -0.009723
PavedDrive_P         -0.000000 -0.001374
PavedDrive_Y          0.005346  0.011097
PoolQC_Ex             0.000000  0.042653
PoolQC_Fa            -0.000000  0.003069
PoolQC_Gd            -0.000000 -0.050238
Fence_GdPrv           0.000000  0.009466
Fence_GdWo           -0.016323 -0.024821
Fence_MnPrv           0.000000  0.003894
Fence_MnWw           -0.000000 -0.015554
MiscFeature_Gar2     -0.000000  0.001163
MiscFeature_Othr     -0.000000 -0.009281
MiscFeature_Shed      0.000000  0.004719
MiscFeature_TenC     -0.000000 -0.001548
SaleType_COD         -0.011456 -0.027306
SaleType_CWD          0.000000  0.010239
SaleType_Con          0.000000  0.011474
SaleType_ConLD        0.000000  0.031708
SaleType_ConLI       -0.000000 -0.014207
SaleType_ConLw       -0.000000 -0.004734
SaleType_New          0.021575  0.020842
SaleType_Oth          0.000000  0.010256
SaleType_WD          -0.030656 -0.038273
SaleCondition_Abnorml -0.047116 -0.040499
```

```
SaleCondition_AdjLand  0.000000  0.020553
SaleCondition_Alloca  -0.000000  0.012122
SaleCondition_Family  -0.007925 -0.023986
SaleCondition_Normal   0.019666  0.029915
SaleCondition_Partial  0.000000  0.001895

[288 rows x 2 columns]
```

In [44]:
```
from mlxtend.classifier import StackingClassifier
stacking_classifier = LinearRegression()
model_stacked = StackingClassifier(classifiers=[model_lasso, model_ridge],
                                   meta_classifier=stacking_classifier).fit(X_
train, y)

print("Stacked min error: %s" % rmse_cv(model_stacked).mean())
```

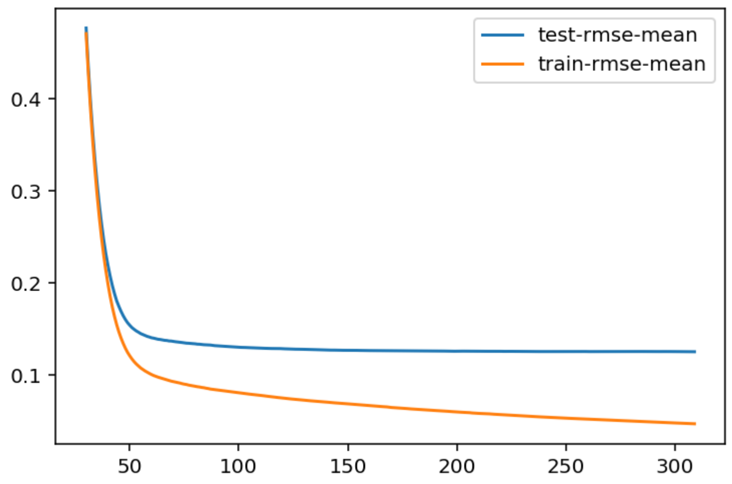
```
Stacked min error: 0.12518695399451502
```

4.5 Install XGBoost (Gradient Boosting) and train a gradient boosting regression. What score can you get just from a single XGB? (you will need to optimize over its parameters).

In [48]:
```
import xgboost as xgb

dtrain = xgb.DMatrix(X_train, label = y)
dtest = xgb.DMatrix(X_test)

params = {"max_depth":3, "eta":0.1}
model = xgb.cv(params, dtrain,  num_boost_round=1000, early_stopping_rounds=10
0)
np.set_printoptions(threshold=np.nan)
print(model["test-rmse-mean"].min())
model.loc[30:,["test-rmse-mean", "train-rmse-mean"]].plot()
```

```
0.12553933333333334
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x166816d0c88>
```



In [46]:
```
model_xgb = xgb.XGBRegressor(n_estimators=1000, max_depth=3, learning_rate=0.1
) #the params were tuned using xgb.cv
model_xgb.fit(X_train, y)

rmse_cv(model_xgb).mean()
```

Out[46]:
```
0.12380131448881475
```

4.9 You will be graded based on your public score (include that in your report) and also on the creativity of your solution. In your report (that you will submit as a pdf file), explain what worked and what did not work.

In [47]:
```
stacked_preds = np.expm1(model_stacked.predict(X_test))
solution = pd.DataFrame({"id":test.Id, "SalePrice":stacked_preds})
solution.to_csv("stacked_sol.csv", index = False)
```

Things that didn't work: stacking with the XGBoost model, using a non-linear stacking classifier. What we found to work: stacking simple models with the linear classifier.

Our final Kaggle score was: 0.12067