Homework 4 - Berkeley STAT 157

Name: Luyu Chen, SID 3034306110, teammates Zhiming Xu, Liang Huang, Zhaorui Zeng

In [1]:

```
%matplotlib inline
import d2l
from mxnet import autograd, gluon, init, nd
from mxnet.gluon import data as gdata, loss as gloss, nn, utils
import numpy as np
import pandas as pd


import seaborn as sns
from scipy import stats
from scipy.stats import skew
from scipy.stats import norm
import matplotlib
import matplotlib.pyplot as plt
```

In [2]:

```
# 这只是为了一下显示100列，不会被折叠，忽略这段代码
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('max_colwidth',100)
```

In [3]:

```
utils.download('https://github.com/d2l-ai/d2l-en/raw/master/data/kaggle_house_pred_train.
utils.download('https://github.com/d2l-ai/d2l-en/raw/master/data/kaggle_house_pred_test.c
train_data = pd.read_csv('kaggle_house_pred_train.csv')
test_data = pd.read_csv('kaggle_house_pred_test.csv')
```

Data Preprocessing

remove outlier

In [4]:

```
train_data.drop(train_data[(train_data['OverallQual']<5) & (train_data['SalePrice']>20000
train_data.drop(train_data[(train_data['GrLivArea']>4000) & (train_data['SalePrice']<2000
train_data.drop(train_data[(train_data['YearBuilt']<1900) & (train_data['SalePrice']>4000
train_data.drop(train_data[(train_data['TotalBsmtSF']>6000) & (train_data['SalePrice']<20
```

In [5]:

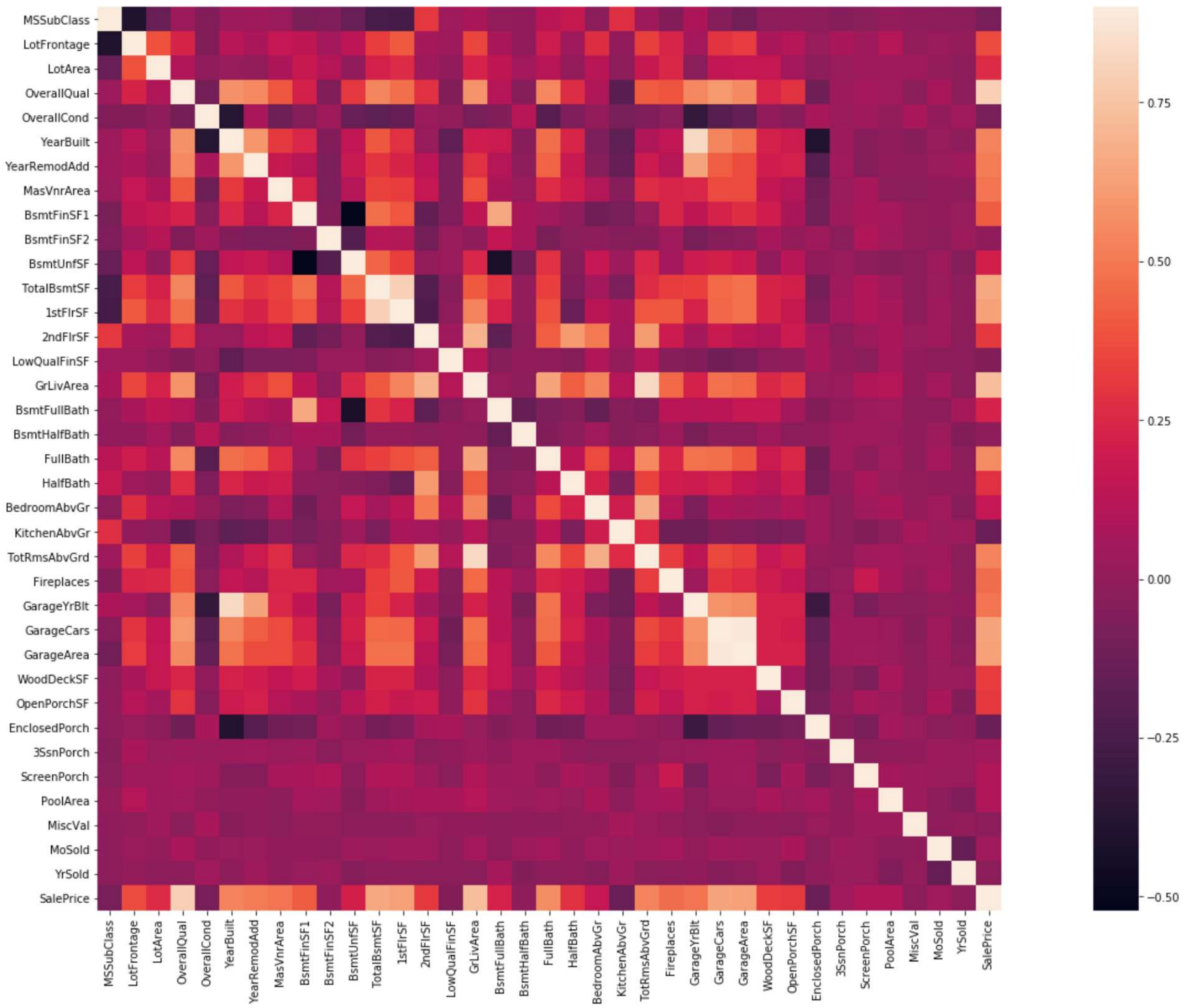
```
print(train_data.shape)
print(test_data.shape)
```

```
(1456, 81)
(1459, 80)
```

Analyze Correlation

In [7]:

```
corrmat = train_data.iloc[:, 1:].corr()
plt.subplots(figsize=(30,15))
sns.heatmap(corrmat, vmax=0.9, square=True)
plt.show()
```



In [8]:

```
all_features = pd.concat((train_data.iloc[:, 1:-1], test_data.iloc[:, 1:]))
all_features = all_features.drop(['TotRmsAbvGrd', 'GarageCars', 'TotalBsmtSF','GarageYrBl
all_features = all_features.drop(['LowQualFinSF','MiscVal', 'BsmtHalfBath', 'BsmtFinSF2',
all_features = all_features.reset_index(drop=True)
print(all_features.shape)
all_features.head(5)
```

```
(2915, 69)
```

Out[8]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	I
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	

In [ ]:

Do something to years: add YearsSinceRemodel

In [9]:

```
all_features['YearsSinceRemodel'] = all_features['YrSold']- all_features['YearRemodAdd']
all_features = all_features.drop(['YrSold','MoSold','YearBuilt','YearRemodAdd'], 1)
print(all_features.shape)
all_features.head(5)
```

```
(2915, 66)
```

Out[9]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	I
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	

Missing value & Feature Selection

In [11]:

```
total = all_features.isnull().sum().sort_values(ascending=False)
percent = (all_features.isnull().sum()/all_features.isnull().count()).sort_values(ascendin
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(30)
```

Out[11]:

	Total	Percent
PoolQC	2906	0.996913
MiscFeature	2810	0.963979
Alley	2717	0.932075
Fence	2345	0.804460
FireplaceQu	1420	0.487136
LotFrontage	485	0.166381
GarageFinish	159	0.054545
GarageCond	159	0.054545
GarageQual	159	0.054545
GarageType	157	0.053859
BsmtExposure	82	0.028130
BsmtCond	82	0.028130
BsmtQual	81	0.027787
BsmtFinType2	80	0.027444
BsmtFinType1	79	0.027101
MasVnrType	24	0.008233
MasVnrArea	23	0.007890
MSZoning	4	0.001372
BsmtFullBath	2	0.000686
Utilities	2	0.000686
Functional	2	0.000686
KitchenQual	1	0.000343
Electrical	1	0.000343
Exterior2nd	1	0.000343
Exterior1st	1	0.000343
SaleType	1	0.000343
BsmtFinSF1	1	0.000343
GarageArea	1	0.000343
BsmtUnfSF	1	0.000343
BldgType	0	0.000000

In [64]:

```
def fill_missings(res):
    #res['GarageType'] = res['GarageType'].fillna('missing')
    #res['GarageFinish'] = res['GarageFinish'].fillna('missing')
    #res['GarageQual'] = res['GarageQual'].fillna('missing')
    #res['GarageCond'] = res['GarageCond'].fillna('missing')

    #res['BsmtQual'] = res['BsmtQual'].fillna('missing')
    #res['BsmtCond'] = res['BsmtCond'].fillna('missing')
    #res['BsmtExposure'] = res['BsmtExposure'].fillna('missing')
    #res['BsmtFinType1'] = res['BsmtFinType1'].fillna('missing')
    #res['BsmtFinType2'] = res['BsmtFinType2'].fillna('missing')

    res['MSZoning'] = res['MSZoning'].fillna('missing')
    res['BsmtFullBath'] = res['BsmtFullBath'].fillna(res['BsmtFullBath'].mode()[0])
    res['Utilities'] = res['Utilities'].fillna('missing')
    res['Exterior1st'] = res['Exterior1st'].fillna(res['Exterior1st'].mode()[0])
    res['Exterior2nd'] = res['Exterior2nd'].fillna(res['Exterior2nd'].mode()[0])
    res['BsmtFinSF1'] = res['BsmtFinSF1'].fillna(0)
    res['BsmtUnfSF'] = res['BsmtUnfSF'].fillna(res['BsmtUnfSF'].mode()[0])

    res['Exterior1st'] = res['Exterior1st'].fillna(res['Exterior1st'].mode()[0])
    res['Exterior2nd'] = res['Exterior2nd'].fillna(res['Exterior2nd'].mode()[0])
    res["Functional"] = res["Functional"].fillna("Typ")
    res["Electrical"] = res["Electrical"].fillna(res["Electrical"].mode()[0])

    res['KitchenQual'] = res['KitchenQual'].fillna(res['KitchenQual'].mode()[0])
    res['SaleType'] = res['SaleType'].fillna(res['SaleType'].mode()[0])
    res['GarageArea'] = res['GarageArea'].fillna(0)

    #res['MasVnrArea'] = res['MasVnrArea'].apply(lambda x: np.exp(4) if x <= 0.0 else x)
    res['MasVnrArea'] = res['MasVnrArea'].fillna(0)
    res['MasVnrType'] = res['MasVnrType'].fillna('missing')



    return res
```

In [73]:

```
def map_values(full):

    full["oMSZoning"] = full.MSZoning.map({'missing': 0, 'C (all)':1, 'RH':2, 'RM':2, 'RL'

    full["oNeighborhood"] = full.Neighborhood.map({'MeadowV':1,
                                               'IDOTRR':2, 'BrDale':2,
                                               'OldTown':3, 'Edwards':3, 'BrkSide':3,
                                               'Sawyer':4, 'Blueste':4, 'SWISU':4, 'NAmes'
                                               'NPkVill':5, 'Mitchel':5,
                                               'SawyerW':6, 'Gilbert':6, 'NWAmes':6,
                                               'Blmngtn':7, 'CollgCr':7, 'ClearCr':7, 'Cra
                                               'Veenker':8, 'Somerst':8, 'Timber':8,
                                               'StoneBr':9,
                                               'NoRidge':10, 'NridgHt':10})

    full["oCondition1"] = full.Condition1.map({'Artery':1,
                                           'Feedr':2, 'RRAe':2,
                                           'Norm':3, 'RRAn':3,
                                           'PosN':4, 'RRNe':4,
                                           'PosA':5 ,'RRNn':5})

    full["oBldgType"] = full.BldgType.map({'2fmCon':1, 'Duplex':1, 'Twnhs':1, '1Fam':2, '

    full["oHouseStyle"] = full.HouseStyle.map({'1.5Unf':1,
                                           '1.5Fin':2, '2.5Unf':2, 'SFoyer':2,
                                           '1Story':3, 'SLvl':3,
                                           '2Story':4, '2.5Fin':4})

    full["oExterior1st"] = full.Exterior1st.map({'BrkComm':1,
                                             'AsphShn':2, 'CBlock':2, 'AsbShng':2,
                                             'WdShing':3, 'Wd Sdng':3, 'MetalSd':3, 'Stucc
                                             'BrkFace':4, 'Plywood':4,
                                             'VinylSd':5,
                                             'CemntBd':6,
                                             'Stone':7, 'ImStucc':7})

    full["oMasVnrType"] = full.MasVnrType.map({'missing':0, 'BrkCmn':1, 'None':1, 'BrkFace

    full["oExterQual"] = full.ExterQual.map({'Fa':1, 'TA':2, 'Gd':3, 'Ex':4})

    full["oFoundation"] = full.Foundation.map({'Slab':1,
                                           'BrkTil':2, 'CBlock':2, 'Stone':2,
                                           'Wood':3, 'PConc':4})

    #full["oBsmtQual"] = full.BsmtQual.map({'Fa':2, 'missing':1, 'TA':3, 'Gd':4, 'Ex':5})

    #full["oBsmtExposure"] = full.BsmtExposure.map({'missing':1, 'No':2, 'Av':3, 'Mn':3,

    full["oHeating"] = full.Heating.map({'Floor':1, 'Grav':1, 'Wall':2, 'OthW':3, 'GasW':

    full["oHeatingQC"] = full.HeatingQC.map({'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})

    full["oKitchenQual"] = full.KitchenQual.map({'Fa':1, 'TA':2, 'Gd':3, 'Ex':4})

    full["oFunctional"] = full.Functional.map({'Maj2':1, 'Maj1':2, 'Min1':2, 'Min2':2, 'Mo
    #full["oGarageType"] = full.GarageType.map({'CarPort':1, 'missing':1,
                                           #'Detchd':2,
                                           #'2Types':3, 'Basment':3,
                                           #'Attchd':4, 'BuiltIn':5})

    #full["oGarageFinish"] = full.GarageFinish.map({'missing':1, 'Unf':2, 'RFn':3, 'Fin':

    full["oPavedDrive"] = full.PavedDrive.map({'N':1, 'P':2, 'Y':3})

    full["oSaleType"] = full.SaleType.map({'COD':1, 'ConLD':1, 'ConLI':1, 'ConLw':1, 'Oth'
                                       'CWD':2, 'Con':3, 'New':3})

    full["oSaleCondition"] = full.SaleCondition.map({'AdjLand':1, 'Abnorml':2, 'Alloca':2,

    return "Done!"
```

In [14]:

```
def transform(X):

        #all_features = all_features.drop(['TotRmsAbvGrd', 'GarageCars', 'TotalBsmtSF','G
        #all_features = all_features.drop(['LowQualFinSF','MiscVal', 'BsmtHalfBath', 'Bsm
        X["TotalHouse"] =  X["1stFlrSF"] + X["2ndFlrSF"]
        X["TotalArea"] =  X["1stFlrSF"] + X["2ndFlrSF"] + X["GarageArea"]

        X["+_TotalHouse_OverallQual"] = X["TotalHouse"] * X["OverallQual"]
        X["+_GrLivArea_OverallQual"] = X["GrLivArea"] * X["OverallQual"]
        X["+_oMSZoning_TotalHouse"] = X["oMSZoning"] * X["TotalHouse"]
        X["+_oMSZoning_OverallQual"] = X["oMSZoning"] + X["OverallQual"]
        X["+_oNeighborhood_TotalHouse"] = X["oNeighborhood"] * X["TotalHouse"]
        X["+_oNeighborhood_OverallQual"] = X["oNeighborhood"] + X["OverallQual"]
        X["+_BsmtFinSF1_OverallQual"] = X["BsmtFinSF1"] * X["OverallQual"]

        X["-_oFunctional_TotalHouse"] = X["oFunctional"] * X["TotalHouse"]
        X["-_oFunctional_OverallQual"] = X["oFunctional"] + X["OverallQual"]
        X["-_LotArea_OverallQual"] = X["LotArea"] * X["OverallQual"]
        X["-_TotalHouse_LotArea"] = X["TotalHouse"] + X["LotArea"]
        X["-_oCondition1_TotalHouse"] = X["oCondition1"] * X["TotalHouse"]
        X["-_oCondition1_OverallQual"] = X["oCondition1"] + X["OverallQual"]

        X["PorchArea"] = X["OpenPorchSF"]+X["EnclosedPorch"]+X["3SsnPorch"]+X["ScreenPorc
        X["TotalPlace"] = X['BsmtUnfSF']+X["1stFlrSF"] + X["2ndFlrSF"] + X["GarageArea"]


        return X
```

In [15]:

```
all_features['LotArea'] = np.log1p(all_features['LotArea'] + 1.0)
all_features['3SsnPorch'] = np.log1p(all_features['3SsnPorch'] + 1.0)
all_features['KitchenAbvGr'] = np.log1p(all_features['KitchenAbvGr'] + 1.0)
all_features['EnclosedPorch'] = np.log1p(all_features['EnclosedPorch'] + 1.0)
all_features['ScreenPorch'] = np.log1p(all_features['ScreenPorch'] + 1.0)
all_features['MasVnrArea'] = np.log1p(all_features['MasVnrArea'] + 1.0)
all_features['OpenPorchSF'] = np.log1p(all_features['OpenPorchSF'] + 1.0)
```

Normalize

In [16]:

```
numeric_features = all_features.dtypes[all_features.dtypes != 'object'].index
all_features[numeric_features] = all_features[numeric_features].apply(
    lambda x: (x - x.mean()) / (x.std()))
# after standardizing the data all means vanish, hence we can set missing values to 0
#all_features = all_features.fillna(0)
print(all_features.shape)
all_features.head(3)
```

```
(2915, 66)
```

Out[16]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
0	0.067175	RL	-0.183043	-0.100176	Pave	NaN	Reg	Lvl	AllPub
1	-0.873119	RL	0.475068	0.151592	Pave	NaN	Reg	Lvl	AllPub
2	0.067175	RL	-0.051421	0.464553	Pave	NaN	IR1	Lvl	AllPub

Apply All the Function Defined Before

In [58]:

```
all_features2 = all_features.drop((missing_data[missing_data['Total'] >= 485]).index,1)
all_features2.shape
```

Out[58]:

```
(2915, 60)
```

In [59]:

```
##dropIndex = [1915, 2216, 2250, 2904, 2120, 2188, 1945, 2473, 2151, 2489, 1379, 1555, 25
all_features3 = all_features2.drop(1375)
all_features3.shape
```

Out[59]:

```
(2914, 60)
```

In [60]:

```
all_features4 = fill_missings(all_features3)
all_features4.shape
```

Out[60]:

```
(2914, 60)
```

In [61]:

```
all_features4 = all_features4.drop(['GarageType', 'GarageFinish','GarageQual','GarageCond'
```

In [62]:

```
### check null value
all_features4.isnull().sum().max()
```

Out[62]:

```
0
```

In [74]:

```
map_values(all_features4)
#transform(all_features4)
all_features4.head()
print(all_features4.shape)
```

```
(2914, 67)
```

In [75]:

```
# Dummy_na=True refers to a missing value being a legal eigenvalue, and creates an indica
all_features4 = pd.get_dummies(all_features4, dummy_na=True)
all_features4.shape
```

Out[75]:

```
(2914, 260)
```

In [76]:

```
### PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=250)
all_features4 = pd.DataFrame(pca.fit_transform(all_features4))
```

SalePrice Log and Normalize

In [24]:

```
n_train = train_data.shape[0]
train_features = nd.array(all_features4[:(n_train-1)].values)
test_features = nd.array(all_features4[(n_train-1):].values)
train_data2 = train_data.drop([1375])
```

In [25]:

```
log = np.log(train_data2.SalePrice.values)
mean = np.mean(log)
std = np.std(log)

label = (log-mean)/std
#print(label)
train_labels = nd.array(label).reshape((-1, 1))
#print(train_labels)
```

Train and K-fold

In [100]:

```
def get_net(drop_prob1,drop_prob2):
    net = nn.Sequential()
    net.add(nn.Dense(1024, activation="relu"),
            # Add a dropout layer after the first fully connected layer
            nn.Dropout(drop_prob1),
            #nn.Dense(64, activation="relu"),
            # Add a dropout layer after the second fully connected layer
            #nn.Dropout(drop_prob2),
            nn.Dense(1)
           )
    net.initialize(init.Normal(sigma=0.01))
    return net
```

In [27]:

```
def loss(train, labels):
    num_train = train.shape[0]
    # To further stabilize the value when the logarithm is taken, set the value less than
    rmse = nd.sqrt((train -labels) ** 2/ num_train)
    return rmse

def log_rmse(net, features, labels):
    # To further stabilize the value when the logarithm is taken, set the value less than
    #clipped_preds = nd.clip(net(features), 1, float('inf'))
    rmse = nd.sqrt(2 * loss(net(features), labels).mean())
    return rmse.asscalar()
```

In [28]:

```
def train(net, train_features, train_labels, test_features, test_labels,
          num_epochs, learning_rate, weight_decay, batch_size):
    train_ls, test_ls = [], []
    train_iter = gdata.DataLoader(gdata.ArrayDataset(
        train_features, train_labels), batch_size, shuffle=True)
    # The Adam optimization algorithm is used here.
    trainer = gluon.Trainer(net.collect_params(), 'adam', {
        'learning_rate': learning_rate, 'wd': weight_decay})
    for epoch in range(num_epochs):
        for X, y in train_iter:
            with autograd.record():
                l = loss(net(X), y)
            l.backward()
            trainer.step(batch_size)
        train_ls.append(log_rmse(net, train_features, train_labels))
        #train_ls.append(loss(net(train_features), train_labels).mean().asscalar())
        if test_labels is not None:
            test_ls.append(log_rmse(net, test_features, test_labels))
            #test_ls.append(loss(net(test_features), test_labels).mean().asscalar())
    return train_ls, test_ls
```

In [29]:

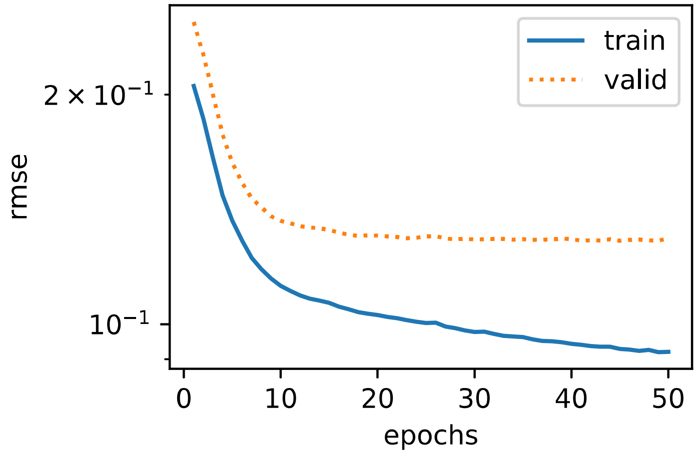
```
def get_k_fold_data(k, i, X, y):
    assert k > 1
    fold_size = X.shape[0] // k
    X_train, y_train = None, None
    for j in range(k):
        idx = slice(j * fold_size, (j + 1) * fold_size)
        X_part, y_part = X[idx, :], y[idx]
        if j == i:
            X_valid, y_valid = X_part, y_part
        elif X_train is None:
            X_train, y_train = X_part, y_part
        else:
            X_train = nd.concat(X_train, X_part, dim=0)
            y_train = nd.concat(y_train, y_part, dim=0)
    return X_train, y_train, X_valid, y_valid
```

In [30]:

```python
def k_fold(k, X_train, y_train, num_epochs,
           learning_rate, weight_decay, batch_size, drop_prob1,drop_prob2):
    train_l_sum, valid_l_sum = 0, 0
    for i in range(k):
        data = get_k_fold_data(k, i, X_train, y_train)
        net = get_net(drop_prob1,drop_prob2)
        train_ls, valid_ls = train(net, *data, num_epochs, learning_rate,
                                   weight_decay, batch_size)
        train_l_sum += train_ls[-1]
        valid_l_sum += valid_ls[-1]
        if i == 0:
            d2l.semilogy(range(1, num_epochs + 1), train_ls, 'epochs', 'rmse',
                         range(1, num_epochs + 1), valid_ls,
                         ['train', 'valid'])
        print('fold %d, train rmse: %f, valid rmse: %f' % (
            i, train_ls[-1], valid_ls[-1]))
    return train_l_sum / k, valid_l_sum / k
```

In [108]:

```python
k, num_epochs, lr, weight_decay, batch_size = 3, 50, 0.00012, 0.0005, 64
train_l, valid_l = k_fold(k, train_features, train_labels, num_epochs, lr,
                          weight_decay, batch_size, 0.7, 0)
print('%d-fold validation: avg train rmse: %f, avg valid rmse: %f'
              % (k, train_l, valid_l))
```



```
fold 0, train rmse: 0.091979, valid rmse: 0.129602
fold 1, train rmse: 0.091047, valid rmse: 0.131212
fold 2, train rmse: 0.089265, valid rmse: 0.136443
3-fold validation: avg train rmse: 0.090764, avg valid rmse: 0.132419
```

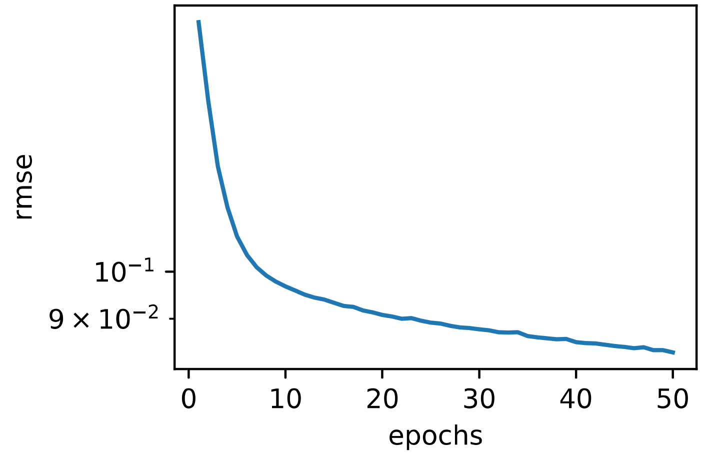
Predict and Submit

In [32]:

```
def train_and_pred(train_features, test_feature, train_labels, test_data,
                   num_epochs, lr, weight_decay, batch_size, drop_prob1,drop_prob2,mean,s
    net = get_net(drop_prob1,drop_prob2)
    train_ls, _ = train(net, train_features, train_labels, None, None,
                        num_epochs, lr, weight_decay, batch_size)
    d2l.semilogy(range(1, num_epochs + 1), train_ls, 'epochs', 'rmse')
    print('train rmse %f' % train_ls[-1])
    # apply the network to the test set
    preds = net(test_features).asnumpy()
    # reformat it for export to Kaggle
    test_data['SalePrice'] = np.expm1(pd.Series(preds.reshape(1, -1)[0])*std+mean)
    submission = pd.concat([test_data['Id'], test_data['SalePrice']], axis=1)
    submission.to_csv('submission.csv', index=False)
```

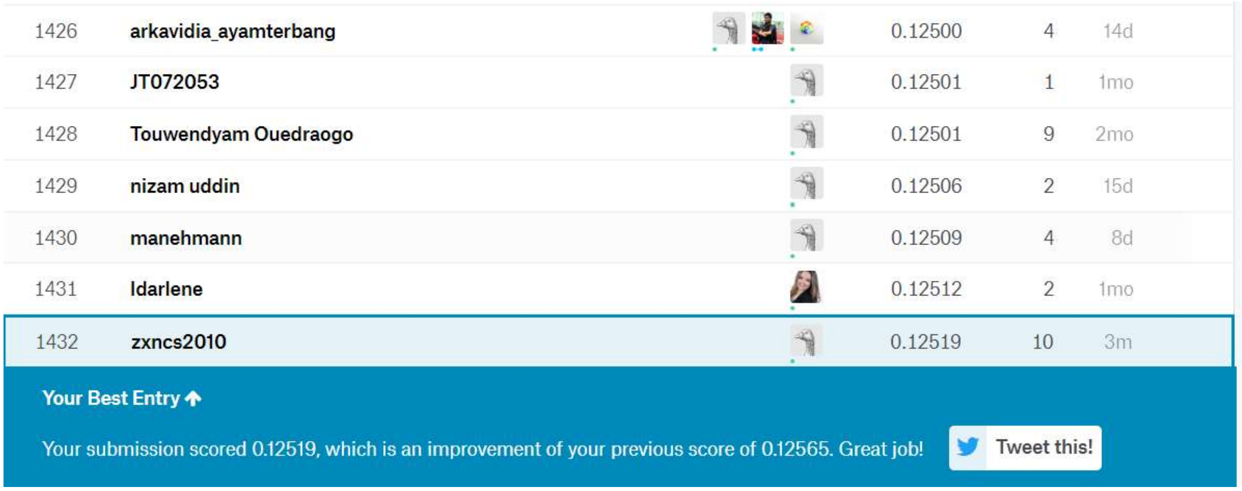
In [109]:

```
train_and_pred(train_features, test_features, train_labels, test_data,
               num_epochs, lr, weight_decay, batch_size,0.7,0,mean,std)
```



```
train rmse 0.083432
```

Note: I basically tried all the data preprocessing method I could think of. And tried every hyperparameters I could think of.
The error just stops at 0.125, which is frustrating...



1426	arkavidia_ayamterbang	0.12500	4	14d
1427	JT072053	0.12501	1	1mo
1428	Touwendyam Ouedraogo	0.12501	9	2mo
1429	nizam uddin	0.12506	2	15d
1430	manehmann	0.12509	4	8d
1431	Idarlene	0.12512	2	1mo
1432	zxncs2010	0.12519	10	3m

Your Best Entry ↑

Your submission scored 0.12519, which is an improvement of your previous score of 0.12565. Great job!

Tweet this!

In [ ]: