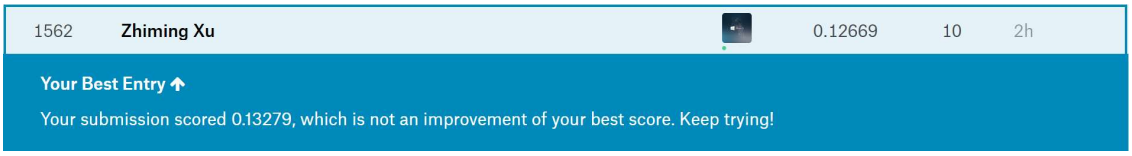
Homework 4 - Berkeley STAT 157

Your name: Zhiming Xu, SID 3034485754, teammates Luyu Chen, Liang Huang, Zhaorui Zeng, Brandon Huang (Please add your name, SID and teammates to ease Ryan and Rachel to grade.)

Handout 2/12/2019, due 2/19/2019 by 4pm in Git by committing to your repository.

In this homework, we will build a model based real house sale data from a Kaggle competition (https://www.kaggle.com/c/house-prices-advanced-regression-techniques). This notebook contains codes to download the dataset, build and train a baseline model, and save the results in the submission format. Your jobs are

1. Developing a better model to reduce the prediction error. You can find some hints on the last section.
2. Submitting your results into Kaggle and take a sceenshot of your score. Then replace the following image URL with your screenshot.



We have two suggestions for this homework:

1. Start as earlier as possible. Though we will cover this notebook on Thursday's lecture, tuning hyper-parameters takes time, and Kaggle limits #submissions per day.
2. Work with your project teammates. It's a good opportunity to get familiar with each other.

Your scores will depend your positions on Kaggle's Leaderboard. We will award the top-3 teams/individuals 500 AWS credits.

Accessing and Reading Data Sets

The competition data is separated into training and test sets. Each record includes the property values of the house and attributes such as street type, year of construction, roof type, basement condition. The data includes multiple datatypes, including integers (year of construction), discrete labels (roof type), floating point numbers, etc.; Some data is missing and is thus labeled 'na'. The price of each house, namely the label, is only included in the training data set (it's a competition after all). The 'Data' tab on the competition tab has links to download the data.

We will read and process the data using `pandas` , an efficient data analysis toolkit (http://pandas.pydata.org/pandas-docs/stable/). Make sure you have `pandas` installed for the experiments in this section.

```
# If pandas is not installed, please uncomment the following line:
# !pip install pandas

%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


#下面的包是我加的
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
```

```
# 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)
```

We downloaded the data into the current directory. To load the two CSV (Comma Separated Values) files containing training and test data respectively we use Pandas.

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

```
train_data.drop(train_data[(train_data['OverallQual']<5) & (train_dat
train_data.drop(train_data[(train_data['GrLivArea']>4000) & (train_da
train_data.drop(train_data[(train_data['YearBuilt']<1900) & (train_da
train_data.drop(train_data[(train_data['TotalBsmtSF']>6000) & (train_
```

The training data set includes 1,460 examples, 80 features, and 1 label., the test data contains 1,459 examples and 80 features.

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

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

Let's take a look at the first 4 and last 2 features as well as the label (SalePrice) from the first 4 examples:

```
train_data.iloc[0:4, [0, 1, 2, 3, -3, -2, -1]]
```

	Id	MSSubClass	MSZoning	LotFrontage	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	WD	Normal	208500
1	2	20	RL	80.0	WD	Normal	181500
2	3	60	RL	68.0	WD	Normal	223500
3	4	70	RL	60.0	WD	Abnorml	140000

Kaggle Tutorial (https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python)
知乎 (https://zhuanlan.zhihu.com/p/39429689)

```
corrmat['SalePrice'].sort_values()
```

```
KitchenAbvGr    -0.135958
EnclosedPorch   -0.128153
OverallCond     -0.086731
MSSubClass      -0.085251
LowQualFinSF    -0.058163
YrSold          -0.025644
MiscVal         -0.021029
BsmtHalfBath    -0.016193
BsmtFinSF2      -0.010554
3SsnPorch        0.045167
MoSold           0.047491
ScreenPorch      0.095740
PoolArea         0.100196
BedroomAbvGr     0.166414
BsmtUnfSF        0.213042
BsmtFullBath     0.231127
LotArea          0.265330
HalfBath         0.283586
OpenPorchSF      0.314867
2ndFlrSF         0.316650
WoodDeckSF       0.327820
LotFrontage      0.370063
BsmtFinSF1       0.413509
Fireplaces       0.466798
MasVnrArea       0.487148
GarageYrBlt      0.489000
YearRemodAdd     0.509039
TotRmsAbvGrd     0.534152
YearBuilt        0.534649
FullBath         0.563970
GarageArea       0.628454
1stFlrSF         0.631967
GarageCars       0.640480
TotalBsmtSF      0.653743
GrLivArea        0.732257
OverallQual      0.796374
SalePrice        1.000000
Name: SalePrice, dtype: float64
```

相关性分析结论 - 明显线性相关的：

变量描述看这里： https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data (https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data)

- YearBuilt - Garage Year Built

- GrLivArea: Above grade (ground) living area square feet - TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
- Garage Cars - Garage Area
- TotalBsmtSF: Total square feet of basement area（基地面积） - 1stFlrSF: First Floor square feet（一层面积）

- 因为相互之间高相关性，去掉了all_features.drop(['TotRmsAbvGrd', 'GarageCars', 'TotalBsmtSF','GarageYrBlt', 'OverallCond'], axis=1)（非常淡的）
- 又因为和salesprice的相关性太低，去掉了'LowQualFinSF','MiscVal', 'BsmtHalfBath', 'BsmtFinSF2', '3SsnPorch', 'PoolArea'

We can see that in each example, the first feature is the ID. This helps the model identify each training example. While this is convenient, it doesn't carry any information for prediction purposes. Hence we remove it from the dataset before feeding the data into the network.

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

```
(2915, 69)
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
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

年份相减，新建变量为YearsSinceRemodel，指房子的年龄（然后把'YrSold','MoSold','YearBuilt','YearRemodAdd'变量删掉了）

```
#先在这里处理一下年份
all_features['YearsSinceRemodel'] = all_features['YrSold']- all_featu
all_features = all_features.drop(['YrSold','MoSold','YearBuilt','Year
print(all_features.shape)
all_features.head(5)
```

```
(2915, 66)
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
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

Data Preprocessing

As stated above, we have a wide variety of datatypes. Before we feed it into a deep network we need to perform some amount of processing. Let's start with the numerical features. We begin by replacing missing values with the mean. This is a reasonable strategy if features are missing at random. To adjust them to a common scale we rescale them to zero mean and unit variance. This is accomplished as follows:

$$x \leftarrow \frac{x - \mu}{\sigma}$$

To check that this transforms x to data with zero mean and unit variance simply calculate $\mathbf{E}[(x - \mu)/\sigma] = (\mu - \mu)/\sigma = 0$. To check the variance we use $\mathbf{E}[(x - \mu)^2] = \sigma^2$ and thus the transformed variable has unit variance. The reason for 'normalizing' the data is that it brings all features to the same order of magnitude. After all, we do not know *a priori* which features are likely to be relevant. Hence it makes sense to treat them equally.

缺失值:

PoolQC, MiscFeature, Alley, Fence, FireplaceQu, LotFrontage不应该被包含进模型里（或者加一个变量missing）。

- Garage同一批数据缺失，GarageYearBuilt不需要包括。
- Basement同一批数据缺失
- Masonry（查了字典这个词是石工？？家里的大理石？？？）同一批数据缺失
- 所有那些只有一条数据缺失的解决办法：
  1）取众数/上一值/下一值（方法一），
  2）把这个observation删掉（如果只有一个observation）？？？

下面找出了缺失值较多的变量

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

	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

删掉了前6个feature，对Garage和Basement这些feature（应该是同一组数据，159和82这些）新建一个missing class

对于缺失只有1，或者2个observation的feature，我删掉了train data里的observation（index 1379），补全了test data里的（补了missing和mode，这一步值得推敲，到底对test data的缺失值补什么）

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

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

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

    #对Area用mean填充
    #res['MasVnrArea'] = res['MasVnrArea'].apply(lambda x: np.exp(4)
    res['MasVnrArea'] = res['MasVnrArea'].fillna(0)
    res['MasVnrType'] = res['MasVnrType'].fillna('missing')



    return res
#mydata=fill_missings(my_data)
```

抄来一段特征工程的代码：

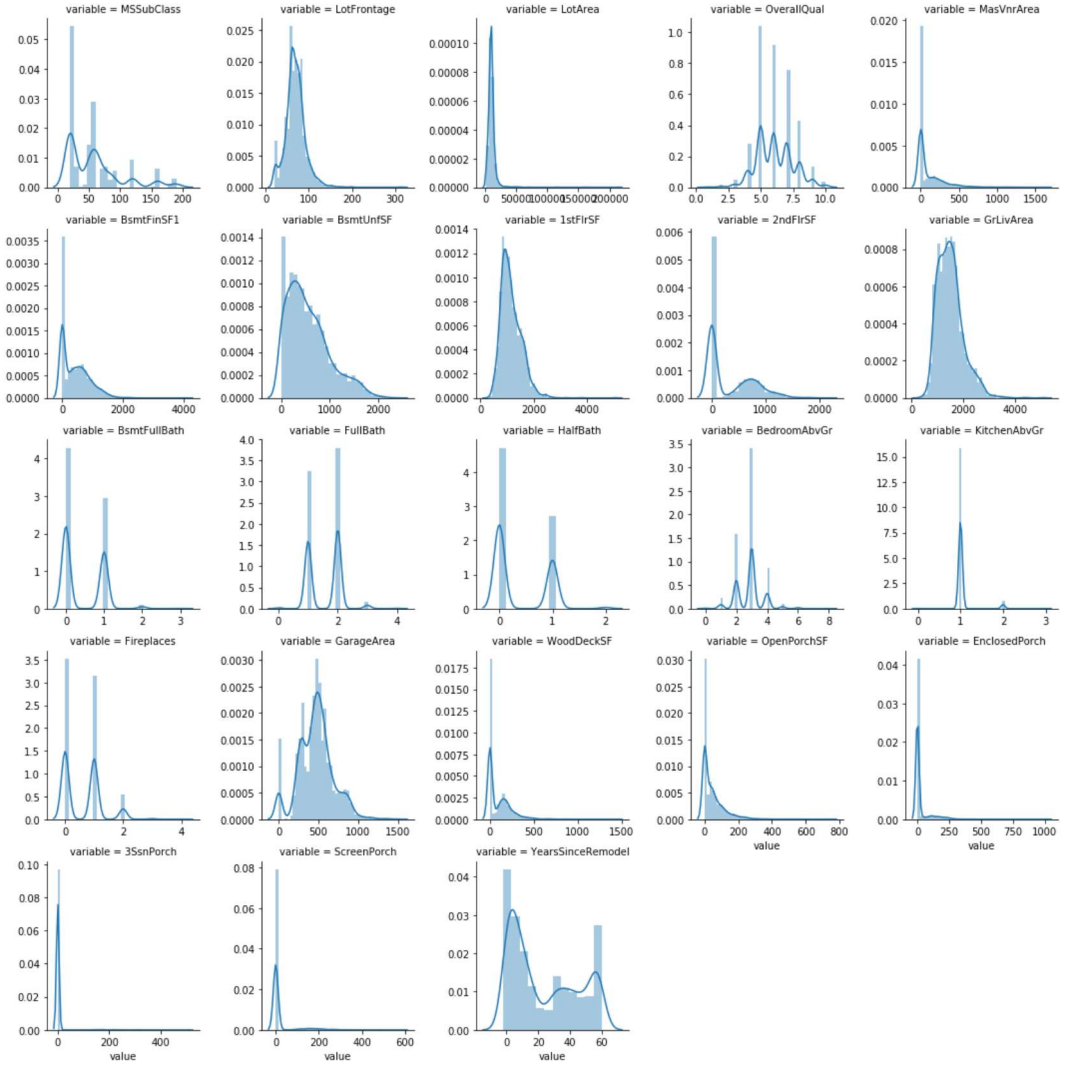
```
def map_values(full):

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

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

    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, 'T

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

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

    full["oMasVnrType"] = full.MasVnrType.map({'missing':0, 'BrkCmn':

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

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

```
    full["oBsmtQual"] = full.BsmtQual.map({'Fa':2, 'missing':1, 'TA':

    full["oBsmtExposure"] = full.BsmtExposure.map({'missing':1, 'No':

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

    full["oHeatingQC"] = full.HeatingQC.map({'Po':1, 'Fa':2, 'TA':3,

    full["oKitchenQual"] = full.KitchenQual.map({'Fa':1, 'TA':2, 'Gd'

    full["oFunctional"] = full.Functional.map({'Maj2':1, 'Maj1':2, 'M


    full["oGarageType"] = full.GarageType.map({'CarPort':1, 'missing'
                                               'Detchd':2,
                                               '2Types':3, 'Basment':3,
                                               'Attchd':4, 'BuiltIn':5})

    full["oGarageFinish"] = full.GarageFinish.map({'missing':1, 'Unf'

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

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

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

    return "Done!"
```

```
def transform(X):

        #all_features = all_features.drop(['TotRmsAbvGrd', 'GarageCar
        #all_features = all_features.drop(['LowQualFinSF','MiscVal',
        X["TotalHouse"] =  X["1stFlrSF"] + X["2ndFlrSF"]
        X["TotalArea"] =  X["1stFlrSF"] + X["2ndFlrSF"] + X["GarageAr

        X["+_TotalHouse_OverallQual"] = X["TotalHouse"] * X["OverallQ
        X["+_GrLivArea_OverallQual"] = X["GrLivArea"] * X["OverallQua
        X["+_oMSZoning_TotalHouse"] = X["oMSZoning"] * X["TotalHouse"
        X["+_oMSZoning_OverallQual"] = X["oMSZoning"] + X["OverallQua
        X["+_oNeighborhood_TotalHouse"] = X["oNeighborhood"] * X["Tot
        X["+_oNeighborhood_OverallQual"] = X["oNeighborhood"] + X["Ov
        X["+_BsmtFinSF1_OverallQual"] = X["BsmtFinSF1"] * X["OverallQ

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

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


        return X
```

试图分析各个变量的skewness，看看有没有变量要取log的变量（比如偏度>0.15）

```
#有哪些要取log的吗……？？？
quantitative = [f for f in all_features.columns if all_features.dtype
skewed_feats = all_features[quantitative].apply(lambda x: skew(x.drop
skewness = pd.DataFrame({'Skew' :skewed_feats})
f = pd.melt(all_features, value_vars=quantitative)
g = sns.FacetGrid(f, col="variable",  col_wrap=5, sharex=False, share
g = g.map(sns.distplot, "value")
plt.show()
```



```
skewness.head(20)
```

	Skew
LotArea	13.279061
3SsnPorch	11.368094
KitchenAbvGr	4.298845
EnclosedPorch	4.000796
ScreenPorch	3.931624
MasVnrArea	2.608281
OpenPorchSF	2.536807
WoodDeckSF	1.845749
MSSubClass	1.375060
1stFlrSF	1.259793
LotFrontage	1.122483
GrLivArea	1.056656
BsmtFinSF1	0.981281
BsmtUnfSF	0.920366
2ndFlrSF	0.859878
Fireplaces	0.725958
HalfBath	0.697173
BsmtFullBath	0.621727
YearsSinceRemodel	0.448405
BedroomAbvGr	0.326792

```
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'] 
all_features['EnclosedPorch'] = np.log1p(all_features['EnclosedPorch'
all_features['ScreenPorch'] = np.log1p(all_features['ScreenPorch'] + 
all_features['MasVnrArea'] = np.log1p(all_features['MasVnrArea'] + 1.
all_features['OpenPorchSF'] = np.log1p(all_features['OpenPorchSF'] + 
```

normalize

```
numeric_features = all_features.dtypes[all_features.dtypes != 'object
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 mis
#all_features = all_features.fillna(0)
print(all_features.shape)
all_features.head(5)
```

```
(2915, 66)
```

	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
3	0.302249	RL	-0.402414	0.141288	Pave	NaN	IR1	Lvl	AllPub
4	0.067175	RL	0.650564	0.932398	Pave	NaN	IR1	Lvl	AllPub

删掉missing data多的几个feature(注意这里命名成了all_features2)

```
all_features2 = all_features.drop((missing_data[missing_data['Total']
all_features2.shape
```

```
(2915, 60)
```

删掉缺失个数小于4的

人工把index找出来了……这里1379属于train，其他index属于test

```python
for index in (missing_data[missing_data['Total'] <5].index):
    print(all_features2.loc[all_features2[index].isnull()].index)
```

```
Int64Index([1911, 2212, 2246, 2900], dtype='int64')
Int64Index([2116, 2184], dtype='int64')
Int64Index([1911, 1941], dtype='int64')
Int64Index([2212, 2469], dtype='int64')
Int64Index([1551], dtype='int64')
Int64Index([1375], dtype='int64')
Int64Index([2147], dtype='int64')
Int64Index([2147], dtype='int64')
Int64Index([2485], dtype='int64')
Int64Index([2116], dtype='int64')
Int64Index([2572], dtype='int64')
Int64Index([2116], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
Int64Index([], dtype='int64')
```

```python
#dropIndex = [1915, 2216, 2250, 2904, 2120, 2188, 1945, 2473, 2151, 2
#属于train的有：1个，
#属于test的有：12个
#for index in dropIndex:
    #all_features3.iloc[index,:] = all_features3.iloc[index,:].fillna
```

把1379drop掉，这里新建了all_features3

```python
all_features3 = all_features2.drop(1375)
all_features3.shape
```

```
(2914, 60)
```

调用之前定义的函数把缺失值补全，新建all_feature4

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

```
(2914, 60)
```

检查缺失值是不是没有了

```
all_features4.isnull().sum().max()
```

```
0
```

Next we deal with discrete values. This includes variables such as 'MSZoning'. We replace them by a one-hot encoding in the same manner as how we transformed multiclass classification data into a vector of 0 and 1. For instance, 'MSZoning' assumes the values 'RL' and 'RM'. They map into vectors $(1, 0)$ and $(0, 1)$ respectively. Pandas does this automatically for us.

```
map_values(all_features4)

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

```
(2914, 97)
```

```
# Dummy_na=True refers to a missing value being a legal eigenvalue, a
# mli 是 (2919, 354)
all_features4 = pd.get_dummies(all_features4, dummy_na=True)
all_features4.shape
```

```
(2914, 342)
```

```
all_features4.head(5)
```

	MSSubClass	LotArea	OverallQual	MasVnrArea	BsmtFinSF1	BsmtUnfSF	1stFlrSF	2ndFlrSF	
0	0.067175	-0.100176	0.650801	1.209366	0.601388	-0.933965	-0.782389	1.212243	0
1	-0.873119	0.151592	-0.061051	-0.793521	1.213886	-0.629175	0.271541	-0.784666	-
2	0.067175	0.464553	0.650801	1.127247	0.105985	-0.287993	-0.616252	1.240302	0
3	0.302249	0.141288	0.650801	-0.793521	-0.502009	-0.046891	-0.509821	0.983089	0
4	0.067175	0.932398	1.362653	1.460151	0.486544	-0.160619	-0.032177	1.677564	1

```
#from sklearn.decomposition import PCA
#pca = PCA(n_components=295)
#all_features4 = pd.DataFrame(pca.fit_transform(all_features4))
```

```
all_features4.shape
```

```
(2914, 342)
```

这边Log SalesPrice了.

You can see that this conversion increases the number of features from 79 to 331. Finally, via the `values` attribute we can extract the NumPy format from the Pandas dataframe and convert it into MXNet's native representation - NDArray for training.

train_data2 = train_data.drop([1379]) 新建train_data2是为了drop了label 1379

```
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])
#test_data2 = test_data.drop([455, 756, 790, 1444,660, 728,485,1013,6
```

SalePrice取log

```
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)
```

temp = nd.array(train_data2.SalePrice.values).reshape((-1, 1))
#print(temp)
#train_label log化 train_labels = temp.log() train_labels -= train_labels.mean()

Training

To get started we train a linear model with squared loss. This will obviously not lead to a competition winning submission but it provides a sanity check to see whether there's meaningful information in the data. It also amounts to a minimum baseline of how well we should expect any 'fancy' model to work.

```
#loss = gloss.L2Loss()
def get_net(drop_prob1,drop_prob2):
    net = nn.Sequential()
    net.add(#nn.Dense(16, activation="relu"),
            # Add a dropout layer after the first fully connected lay
            #nn.Dropout(drop_prob1),
            #nn.Dense(32, activation="relu"),
            # Add a dropout layer after the second fully connected la
            #nn.Dropout(drop_prob2),
            nn.Dense(1)
           )
    net.initialize(init.Normal(sigma=0.01))
    return net
```

loss = gloss.L2Loss()

def get_net(drop_prob1,drop_prob2): net = nn.Sequential() net.add(nn.Dense(1)) net.initialize() return net

House prices, like shares, are relative. That is, we probably care more about the relative error $\frac{y-\hat{y}}{y}$ than about the absolute error. For instance, getting a house price wrong by USD 100,000 is terrible in Rural Ohio, where the value of the house is USD 125,000. On the other hand, if we err by this amount in Los Altos Hills, California, we can be proud of the accuracy of our model (the median house price there exceeds 4 million).

One way to address this problem is to measure the discrepancy in the logarithm of the price estimates. In fact, this is also the error that is being used to measure the quality in this competition. After all, a small value δ of $\log y - \log \hat{y}$ translates into $e^{-\delta} \leq \frac{\hat{y}}{y} \leq e^{\delta}$. This leads to the following loss function:

$$L = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log y_i - \log \hat{y}_i)^2}$$

def log_rmse(net, features, labels):

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

```
#log之后
def loss(train, labels):
    num_train = train.shape[0]
    # To further stabilize the value when the logarithm is taken, set
    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
    #clipped_preds = nd.clip(net(features), 1, float('inf'))
    rmse = nd.sqrt(2 * loss(net(features), labels).mean())
    return rmse.asscalar()
```

Unlike in the previous sections, the following training functions use the Adam optimization algorithm. Compared to the previously used mini-batch stochastic gradient descent, the Adam optimization algorithm is relatively less sensitive to learning rates. This will be covered in further detail later on when we discuss the details on Optimization Algorithms (../chapter_optimization/index.md) in a separate chapter.

```
def train(net, train_features, train_labels, test_features, test_labe
          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
        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).mea
    return train_ls, test_ls
```

k-Fold Cross-Validation

The k-fold cross-validation was introduced in the section where we discussed how to deal with "Model Selection, Underfitting and Overfitting" (underfit-overfit.md). We will put this to good use to select the model design and to adjust the hyperparameters. We first need a function that returns the i-th fold of the data in a k-fold cros-validation procedure. It proceeds by slicing out the i-th segment as validation data and returning the rest as

training data. Note - this is not the most efficient way of handling data and we would use something much smarter if the amount of data was considerably larger. But this would obscure the function of the code considerably and we thus omit it.

```
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
```

The training and verification error averages are returned when we train k times in the k-fold cross-validation.

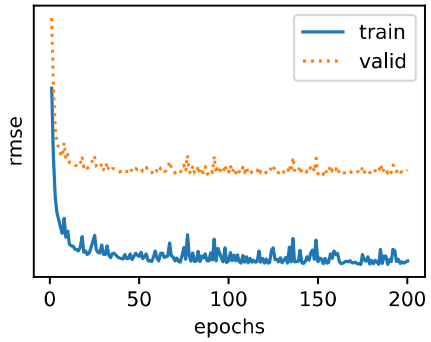
```
def k_fold(k, X_train, y_train, num_epochs,
           learning_rate, weight_decay, batch_size, drop_prob1,drop_p
    train_l_sum, valid_l_sum = 0, 0
    for i in range(k):
        data = get_k_fold_data(k, i, X_train, y_train)
        net = get_net(drop_prob1,drop_prob2)
        train_ls, valid_ls = train(net, *data, num_epochs, learning_r
                                   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'
                         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
```

Model Selection

We pick a rather un-tuned set of hyperparameters and leave it up to the reader to improve the model considerably. Finding a good choice can take quite some time, depending on how many things one wants to optimize over. Within reason the k-fold crossvalidation approach is resilient against multiple testing. However, if we were to try out an unreasonably large number of options it might fail since we might just get lucky on the validation split with a particular set of hyperparameters.

```
#全部的price都log过
print(train_features.shape)
k, num_epochs, lr, weight_decay, batch_size = 3, 200, 0.001, 0.01, 64
train_l, valid_l = k_fold(k, train_features, train_labels, num_epochs
                          weight_decay, batch_size, 0.2, 0)
print('%d-fold validation: avg train rmse: %f, avg valid rmse: %f'
              % (k, train_l, valid_l))
```

```
(1455, 342)
```




```
fold 0, train rmse: 0.107974, valid rmse: 0.133974
fold 1, train rmse: 0.107908, valid rmse: 0.137533
fold 2, train rmse: 0.106641, valid rmse: 0.142346
3-fold validation: avg train rmse: 0.107508, avg valid rmse: 0.137951
```

You will notice that sometimes the number of training errors for a set of hyper-parameters can be very low, while the number of errors for the K-fold cross validation may be higher. This is most likely a consequence of overfitting. Therefore, when we reduce the amount of training errors, we need to check whether the amount of errors in the k-fold cross-validation have also been reduced accordingly.

Predict and Submit

Now that we know what a good choice of hyperparameters should be, we might as well use all the data to train on it (rather than just $1 - 1/k$ of the data that is used in the crossvalidation slices). The model that we obtain in this way can then be applied to the test set. Saving the estimates in a CSV file will simplify uploading the results to Kaggle.

test_data['SalePrice'] = nd.exp(pd.Series(preds.reshape(1, -1)[0]))`我又exp回来了

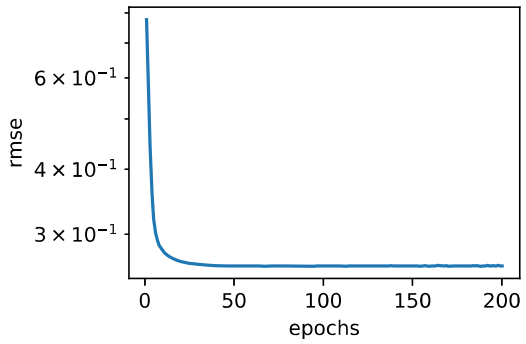
def train_and_pred(train_features, test_feature, train_labels, test_data, num_epochs, lr, weight_decay, batch_size, drop_prob1,drop_prob2): net = get_net(drop_prob1,drop_prob2) train_ls, _ = train(net, train_features, train_labels, None, None, num_epochs, lr, weight_decay, batch_size) d2l.semilogy(range(1, 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'] = pd.Series(preds.reshape(1, -1)
[0])
submission = pd.concat([test_data['Id'], test_data['SaleP
rice']], axis=1)
submission.to_csv('submission.csv', index=False)
```

```
#如果log了用这个
def train_and_pred(train_features, test_feature, train_labels, test_d
                   num_epochs, lr, weight_decay, batch_size, drop_pro
    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)[
    submission = pd.concat([test_data['Id'], test_data['SalePrice']],
    submission.to_csv('submission.csv', index=False)
```

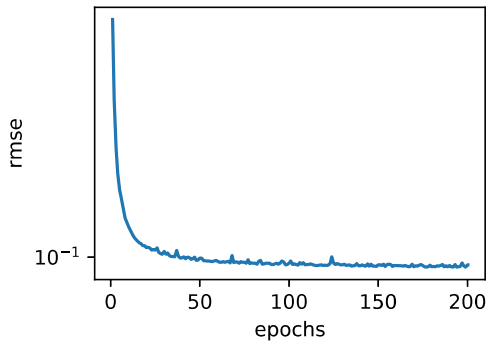
Let's invoke the model. A good sanity check is to see whether the predictions on the test set resemble those of the k-fold crossvalication process. If they do, it's time to upload them to Kaggle.

```
train_and_pred(train_features, test_features, train_labels, test_data
               num_epochs, 0.001, weight_decay, batch_size,0.4,0,mean
```




```
train rmse 0.260672
```

```
# .2047 256-relu, no dropout
# .2189 256-relu, .2 dropout
# .2373 256-relu, .4 dropout
# .2388 32-relu, .2 dropout
# .2549 16-relu, .2 dropout
train_and_pred(train_features, test_features, train_labels, test_data
               num_epochs, 0.0003, weight_decay, batch_size, 0, 0, me
```




```
train rmse 0.098138
```

```
train_and_pred(train_features, test_features, train_labels, test_data
               num_epochs, lr, weight_decay, batch_size,0.3,0)
```

A file, `submission.csv` will be generated by the code above (CSV is one of the file formats accepted by Kaggle). Next, we can submit our predictions on Kaggle and compare them to the actual house price (label) on the testing data set, checking for errors. The steps are quite simple:

- Log in to the Kaggle website and visit the House Price Prediction Competition page.
- Click the "Submit Predictions" or "Late Submission" button on the right.
- Click the "Upload Submission File" button in the dashed box at the bottom of the page and select the prediction file you wish to upload.
- Click the "Make Submission" button at the bottom of the page to view your results.

Hints

1. Can you improve your model by minimizing the log-price directly? What happens if you try to predict the log price rather than the price?
2. Is it always a good idea to replace missing values by their mean? Hint - can you construct a situation where the values are not missing at random?
3. Find a better representation to deal with missing values. Hint - What happens if you add an indicator variable?
4. Improve the score on Kaggle by tuning the hyperparameters through k-fold crossvalidation.
5. Improve the score by improving the model (layers, regularization, dropout).
6. What happens if we do not standardize the continuous numerical features like we have done in this section?

Note for converting this notebook into PDF. If you use 'File -> Download as -> PDF', you may get the error that svg cannot converted because inkscape is not installed and cannot find PNG images. The easiest way is printing this notebook as a PDF in your browser. Or, you can install inkscape to convert SVG (On macOS, you may `brew cask install xquartz inkscape` , on Ubuntu, you may `sudo apt-get install inkscape`) and change the image URL to local filenames.