part 1 : Machine Learning

**##split the numeric and categorical variables**

num\_columns = []

cate\_columns = []

for column in test.columns:

if train.dtypes[column] != 'object':

num\_columns.append(column)

else:

cate\_columns.append(column)

#a easier method

cat\_train=train.select\_dtypes(['object'])

num\_train=train.select\_dtypes(exclude=['object'])

**##evaluate the accuracy of ragression model**

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_squared\_log\_error

**##split train set and test set**

from sklearn.model\_selection import train\_test\_split

train\_X,test\_X,train\_y,test\_y=train\_test\_split(X,y) #train\_size (default:0.75f),random\_state

**##estimate the mean absolute error of DecisionTree with different max nodes**

def get\_mae(max\_nodes,train\_X,train\_y,test\_X,test\_y):

model=DecisionTreeRegressor(max\_nodes=max\_nodes,random\_state=0)

model.fit(train\_X,train\_y)

pre=model.predict(test\_X)

mae=mean\_absolute\_error(pre,test\_y)

return mae

**##random forset model**

from sklearn.ensemble import RandomForestRegressor

**##deal with missing data**

1.delete col with nan in this col (which is not a good solution to deal with missing data)

nan\_col=[col for col in train.columns if train[col].isnull().any()]

reduced\_date=train.drop(nan\_col,axis=1)

2.a better way

from sklearn.preprocessing import Imputer

my\_imputer=Imputer()

data\_with\_imputed\_values=my\_imputer.fit\_transform(train)

we can change the fill method by change the para strategy,its default value is 'mean'.the optinal value contain 'median','most\_frequent'

3.get score from impution with extra columns showing what wad imputed

miss\_col=[col for col in train.columns if train[col].isnull().any()]

for col in miss\_col:

train[col+"\_miss"]=train[col].isnull()

my\_imputer=Imputer()

train=my\_imputer.fit\_transform(train)

**##one hot encoding**

one hot encoding generally won't be used for variables taking more than 15 values.

from this point, we need to get a col with less than 15 distinct values

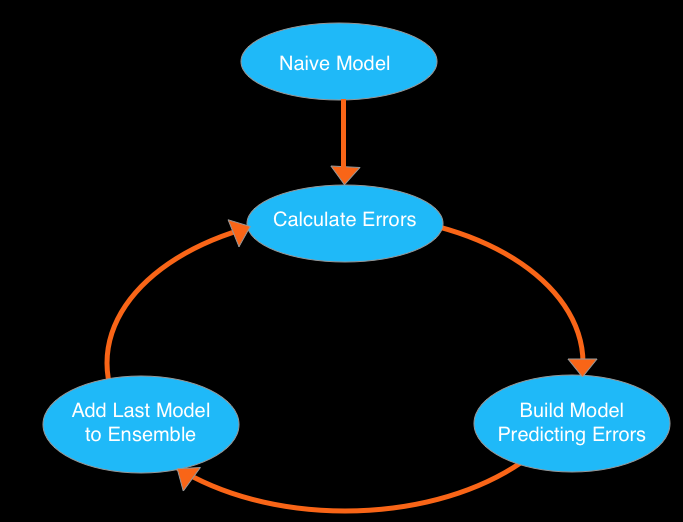
cat\_cols=[col for col in train.columns if train[col].dtypes=='object' and train[col].nunique()<15]

note:train[col].unique() returns an array, while nunique() returns a int

**##delete rows which has nan values of several columns**

train.dropna(axis=0,subset=['SalePrice'],inplace=True)

**##xgboost**



how to tune?

parameters:

n\_estimators:specifies how many times to go through the model cycle described above

if its num is too big, it may bring the problem of overfitting,typical values

range from 100 to 1000

early\_stopping\_rounds:cause the model to stop iterating when the validation

score stops moving, even if we are not at the hard stop for n\_estimators,

5 is a reasonable value

learning\_rate:instead of getting predictions by simply adding up the predictions

from each component model, we will multiply the predictions from each model

by a small number before adding them in.this means each tree we add to this

ensemble helps less,this reduces propensity to overfit

**##partial dependency plot**

partial dependency plot how each variable or predictor affects the model's predictions

from sklearn.ensemble.partial\_dependence import partial\_dependence,plot\_partial\_dependence

model=GradientBoostingRegressor()

cols\_to\_use = ['Distance', 'Landsize', 'BuildingArea']

model.fit(X,y) #**here, we must fit before plot**

plots=plot\_partial\_dependence(model,

features=[0,2],

X=X,

feature\_names=cols\_to\_use,

grid\_resolution=10)

there are some negative values, it doesn't mean it's really negative value.

**##understanding pipeline**

Pipelines are a simple way to keep your data processing and modeling code organized. Specifically, a pipeline bundles preprocessing and modeling steps so you can use the whole bundle as if it were a single step.

Most scikit-learn objects are either transformers or models.

Transformers are for pre-processing before modeling. The Imputer class (for filling in missing values) is an example of a transformer.

Models are used to make predictions. You will usually preprocess your data (with transformers) before putting it in a model.

example:

from sklearn.pipeline import make\_pipeline

my\_pipeline=make\_pipeline(Imputer(),RandomForestRegressor())

my\_pipeline.fit(train\_X,train\_y) #first it execute imputer, then train data with model

pre=my\_pipeline.predict(test\_X)

##cross-validation

train-test-split: suitable for small data set

ccross validation: suitable for bigger data set

example:

from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import Imputer

from sklearn.pipeline import make\_pipeline

from sklearn.model\_selection import cross\_val\_score

my\_pip=make\_pipeline(Imputer(),RandomForestRegressor())

scores=cross\_val\_score(my\_pip,X,y,scoring='neg\_mean\_absolute\_error') #para 'cv' default value:3

print scores

#result:[-322195.39079754 -303433.56326314 -283034.51985773]

different scoring function:

<http://scikit-learn.org/stable/modules/model_evaluation.html>

##leakage

two main types of leakage: Leaky Predictors and a Leaky Validation Strategies.

1.leaky predictors

This occurs when your predictors include data that will not be available at the time you make predictions.

solution:

To prevent this type of data leakage, any variable updated (or created) after the target value is realized should be excluded. Because when we use this model to make new predictions, that data won't be available to the model.

remember:

To screen for possible leaky predictors, look for columns that are statistically correlated to your target.

If you build a model and find it extremely accurate, you likely have a leakage problem.

2.leaky validation strategies

This occurs when you aren't careful distinguishing training data from validation data.

For example, this happens if you run preprocessing (like fitting the Imputer for missing values) before calling train\_test\_split.

solution:

If your validation is based on a simple train-test split, exclude the validation data from any type of fitting, including the fitting of preprocessing steps.