447final

December 4, 2018

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import sys
from html.parser import HTMLParser
from html.entities import name2codepoint
sns.set(color_codes=True)
import warnings
warnings.filterwarnings("ignore")
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
LA = np.linalg

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from collections import defaultdict
from gensim.models.word2vec import Word2Vec
import re
%matplotlib inline
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/chd415/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [2]:
```
# load training data text
def load_data(filename):
    load_file = pd.read_csv(filename,delimiter=',', header=0,
                            dtype={'name':str, 'lvl1':str, 'lvl2':str, 'lvl3':str, 'descrp
    load_file.columns = ['id', 'name','lvl1','lvl2','lvl3','descrption','price','type']
    load_file.duplicated(subset=None, keep='first')
```

```
        load_file.set_index('id', inplace = True)
        load_file.head()
        return load_file
    #print(len(train_file))
    def load_label(filename):
        load_label = pd.read_csv(filename,delimiter=',', header=0)
        load_label.columns = ['id', 'score']
        load_label.duplicated(subset=None, keep='first')
        load_label.set_index('id', inplace = True)
        return load_label
```

In [3]:
```
    '''
    def map_mathod(column):
        values = []
        indexs = []
        mapping = {}
        index = 0
        for count in range(len(train_file)):
            value = train_file.get_value(count+1,column)
            if value in values and value != np.nan:
                continue
            values.append(value)
            indexs.append(len(values))
        for j in range(len(indexs)):
            mapping[values[j]] = indexs[j]
        mapping[np.nan] = 0.0
        return mapping
    #train_file['lvl3'] = train_file['lvl3'].str.lower().replace('[^\'\w]+',' ',regex=True
    #mapping_lvl3 = map_mathod('lvl3')
    #print(mapping_lvl3)
    '''

    class MultiColumnLabelEncoder:
        def __init__(self,columns = None):
            self.columns = columns # array of column names to encode

        def fit(self,X,y=None):
            return self # not relevant here

        def transform(self,X):
            '''
            Transforms columns of X specified in self.columns using
            LabelEncoder(). If no columns specified, transforms all
            columns in X.
            '''
            output = X.copy()
            if self.columns is not None:
                for col in self.columns:
```

```
                    output[col] = LabelEncoder().fit_transform(output[col])
            else:
                for colname,col in output.iteritems():
                    output[colname] = LabelEncoder().fit_transform(col)
            return output

        def fit_transform(self,X,y=None):
            return self.fit(X,y).transform(X)

In [4]: def text_embedding(column):
        temp_X = column.astype(str)
        stop = set(stopwords.words('english'))
        temp =[]
        snow = nltk.stem.SnowballStemmer('english')

        for sentence in temp_X:
            words = [snow.stem(word) for word in sentence.split(' ') if word not in stopwo
            temp.append(sentence)

        count_vect = CountVectorizer(max_features=10000)
        bow_data = count_vect.fit_transform(temp)

        final_tf = temp
        tf_idf = TfidfVectorizer(max_features=10000)
        tf_data = tf_idf.fit_transform(final_tf)
        w2v_data = temp
        splitted = []
        for row in w2v_data:
            splitted.append([word for word in row.split()])     #splitting words

        train_w2v = Word2Vec(splitted,min_count=1,size=50, workers=4)
        avg_data = []
        for row in splitted:
            vec = np.zeros(50, dtype=float)
            count = 0
            for word in row:
                try:
                    vec += train_w2v[word]
                    count += 1
                except:
                    pass
            if (count == 0):
                avg_data.append(vec)
            else:
                avg_data.append(vec/count)
    #         avg_data.append(vec)

        return avg_data
```

In [ ]:
```
'''
#test cell
temp = load_data('train_data.csv')
temp['descrption'] = temp['descrption'].str.lower()
description_X = temp.descrption.str.lower().replace('<.*?>','',regex=True).replace('[^
'''
```

In [ ]:
```
'''
#test cell
def test_embedding(column):
    temp_X = column.astype(str)
    stop = set(stopwords.words('english'))
    temp =[]
    snow = nltk.stem.SnowballStemmer('english')

    for sentence in temp_X:
        words = [snow.stem(word) for word in sentence.split(' ') if word not in stopwo
        temp.append(sentence)

    count_vect = CountVectorizer(max_features=5000)
    bow_data = count_vect.fit_transform(temp)

    final_tf = temp
    tf_idf = TfidfVectorizer(ngram_range=(1,2),stop_words = 'english',max_features=500
    tf_data = tf_idf.fit_transform(final_tf)
    w2v_data = temp
    splitted = []
    for row in w2v_data:
        splitted.append([word for word in row.split()])     #splitting words

    train_w2v = Word2Vec(splitted,min_count=5,size=50, workers=4)
    avg_data = []
    for row in splitted:
        vec = np.zeros(50, dtype=float)
        count = 0
        for word in row:
            try:
                vec += train_w2v[word]
                count += 1
            except:
                pass
        if (count == 0):
            avg_data.append(vec)
        else:
            avg_data.append(vec/count)
#         avg_data.append(vec)

    tf_w_data = []
```

```
            tf_data = tf_data.toarray()
            i = 0
            for row in splitted:
                vec = [0 for i in range(50)]

                temp_tfidf = []
                for val in tf_data[i]:
                    if val != 0:
                        temp_tfidf.append(val)

                count = 0
                tf_idf_sum = 0
                for word in row:
                    try:
                        count += 1
                        tf_idf_sum = tf_idf_sum + temp_tfidf[count-1]
                        vec += (temp_tfidf[count-1] * train_w2v[word])
                    except:
                        pass
                if (tf_idf_sum == 0):
                    tf_w_data.append(vec)
                else:
                    tf_w_data.append(vec/tf_idf_sum)
        #             vec = float(1/tf_idf_sum) * vec
        #         tf_w_data.append(vec)
                i = i + 1

            return tf_w_data
        '''
```

In [ ]:
```
        '''
        #test cell
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.feature_extraction.text import TfidfVectorizer

        temp['descrption'] = test_embedding(description_X)
        temp.descrption.head()
        #tfidf_n = TfidfVectorizer(ngram_range=(1,2),stop_words = 'english')
        '''
```

In [5]:
```
def clean_data(filename):

    filename['lvl1'] = filename['lvl1'].str.lower().replace('[^a-zA-Z]+',' ',regex=True
    filename['lvl2'] = filename['lvl2'].str.lower().replace('[^\'\w]+',' ',regex=True)
    filename['lvl3'] = filename['lvl3'].str.lower().replace('[^\'\w]+',' ',regex=True)
    filename['descrption'] = filename['descrption'].str.lower()
    filename['name'] = filename['name'].str.lower()
```

```
'''

mapping_lvl1 = map_mathod('lvl1')
mapping_lvl2 = map_mathod('lvl2')
mapping_lvl3 = map_mathod('lvl3')

filename['lvl1'] = filename['lvl1'].map(mapping_lvl1)
filename['lvl2'] = filename['lvl2'].map(mapping_lvl2)
filename['lvl3'] = filename['lvl3'].map(mapping_lvl3)

'''
#clean up data for lvl 1&2&3
temp =  filename.drop(['price', 'descrption','name','type'], axis=1)
outfile = MultiColumnLabelEncoder(columns = ['lvl1','lvl2','lvl3']).fit_transform(t
enc = preprocessing.OneHotEncoder()
enc.fit(outfile)
outfile = enc.transform(outfile).toarray()

#normalize price
maxp = filename.price.max()
valuethred = 1000.
filename['price'] = filename['price'].clip(lower=0.,upper=valuethred)
#filename['price'] = filename['price'].clip(lower=0.,upper=valuethred).div(valueth
#hist = train_file['price'].hist(bins=10)
#maxp
ld = filename.price.as_matrix(columns=None).tolist()
outfile = np.column_stack((outfile,ld))

#clean up type
mapping_type = {'international':1.,'local':2., np.nan:0.}
filename['type'] = filename['type'].map(mapping_type)
le = filename.type.as_matrix(columns=None).tolist()
outfile = np.column_stack((outfile,le))

#clean up text
description_X = filename.descrption.str.lower().replace('<li>','final ',regex=True)
count_descrption = description_X.str.count('final').fillna(0).tolist()
outfile = np.column_stack((outfile,count_descrption))

description_X = filename.descrption.str.lower().replace('final ','',regex=True)
filename['descrption'] = text_embedding(description_X)

lg = filename.descrption.as_matrix(columns=None).tolist()
lg_array = np.vstack( lg )

U, s, Vh = LA.svd(lg_array, full_matrices=False)
assert np.allclose(lg_array, np.dot(U, np.dot(np.diag(s), Vh)))
s_new = s[:8]
```

```
        U_new = U[:, :8]
        new_lg = np.dot(U_new, np.diag(s_new))
        lg_min = np.min(new_lg)
        lg_out = new_lg  - (lg_min-2)*np.ones_like(new_lg.size)
        outfile = np.column_stack((outfile,lg_out))


        name_X = filename.name.str.lower().replace('<.*?>','',regex=True).replace('[^\w\s]+
        filename['name'] = text_embedding(name_X)

        lf = filename.name.as_matrix(columns=None).tolist()
        lf_array = np.vstack( lf )

        Uf, sf, Vhf = LA.svd(lf_array, full_matrices=False)
        assert np.allclose(lf_array, np.dot(Uf, np.dot(np.diag(sf), Vhf)))
        sf_new = sf[:8]
        Uf_new = Uf[:, :8]
        new_lf = np.dot(Uf_new, np.diag(sf_new))
        lf_min = np.min(new_lf)
        lf_out = new_lf  - (lf_min-2)*np.ones_like(new_lf.size)
        outfile = np.column_stack((outfile,lf_out))


        return outfile
```

In [ ]:
```
'''
train_file = pd.read_csv('train_data.csv',delimiter=',', header=0, nrows=10)
train_file.columns = ['id', 'name','lvl1','lvl2','lvl3','descrption','price','type']
description_X = train_file.descrption.str.lower().replace('<li>','final ',regex=True).
count_descrption = description_X.str.count('final')
description_X = train_file.descrption.str.lower().replace('final ','',regex=True)
print(np.shape(count_descrption))

'''
```

In [6]:
```
train_file = load_data('train_data.csv')
test_file = load_data('test_data.csv')
combined_file = pd.concat([train_file,test_file])
cleaned_train = clean_data(combined_file)
train_score = load_label('train_label.csv')
np.shape(cleaned_train)
```

Out[6]:
```
(36283, 270)
```

In [ ]:
```
'''
# test cell
cleaned_train['descrption'] = temp['descrption']
'''
```

In [7]:
```
cleaned_train
#print(mapping_lvl1)
```

Out[7]:
```
array([[ 0.        ,  0.        ,  0.        , ..., 10.13324131,
        11.99554897, 11.35688418],
       [ 0.        ,  1.        ,  0.        , ..., 11.9036486 ,
         9.82575646, 10.58128699],
       [ 0.        ,  0.        ,  0.        , ..., 11.73378475,
        11.62101979, 11.51830028],
       ...,
       [ 0.        ,  0.        ,  0.        , ..., 12.33374908,
        11.35996205, 11.5136934 ],
       [ 0.        ,  0.        ,  0.        , ..., 11.17770531,
        11.97945511, 11.61931454],
       [ 0.        ,  0.        ,  0.        , ..., 10.76861269,
        11.21852384, 11.04587675]])
```

In [8]:
```
'''

def rearrange(cleaned_data):
    la = cleaned_data.lvl1.as_matrix(columns=None).tolist()
    lb = cleaned_data.lvl2.as_matrix(columns=None).tolist()
    lc = cleaned_data.lvl3.as_matrix(columns=None).tolist()

    X = la
    X = np.column_stack((X,lb))
    X = np.column_stack((X,lc))

    enc = preprocessing.OneHotEncoder()
    enc.fit(X)
    X = enc.transform(X).toarray()

    ld = cleaned_data.price.as_matrix(columns=None).tolist()
    le = cleaned_data.type.as_matrix(columns=None).tolist()

    X = np.column_stack((X,ld))
    X = np.column_stack((X,le))


    lf = cleaned_data.name.as_matrix(columns=None).tolist()
    lg = cleaned_data.descrption.as_matrix(columns=None).tolist()
    lg_array = np.vstack( lg )
    lf_array = np.vstack( lf )

    U, s, Vh = LA.svd(lg_array, full_matrices=False)
    assert np.allclose(lg_array, np.dot(U, np.dot(np.diag(s), Vh)))

# only use U \cdot s
```

```
        #    s[8:] = 0.
        #    new_lg = np.dot(U, np.dot(np.diag(s), Vh))
            s_new = s[:5]
            U_new = U[:, :5]
            new_lg = np.dot(U_new, np.diag(s_new))
            lg_min = np.min(new_lg)
            lg_out = new_lg  - (lg_min-2)*np.ones_like(new_lg.size)

            Uf, sf, Vhf = LA.svd(lf_array, full_matrices=False)
            assert np.allclose(lf_array, np.dot(Uf, np.dot(np.diag(sf), Vhf)))

        #    sf[5:] = 0.
        #    new_lf = np.dot(Uf, np.dot(np.diag(sf), Vhf))
            sf_new = sf[:5]
            Uf_new = Uf[:, :5]
            new_lf = np.dot(Uf_new, np.diag(sf_new))
            lf_min = np.min(new_lf)
            lf_out = new_lf  - (lf_min-2)*np.ones_like(new_lf.size)


            X = np.column_stack((X,lf_out))
            X = np.column_stack((X,lg_out))
            X = X.tolist()

            return X,lf_min,lg_min
        '''

            #print(len(X))
        X = cleaned_train#rearrange(cleaned_train)
        print(np.shape(np.array(X)))
        Y = train_score.score.as_matrix(columns=None).tolist()
        X[137,253]
```

```
(36283, 270)
```

Out[8]: 0.0

```
In [9]: X = cleaned_train[:18141]
        XX = cleaned_train[18141:]
        Y = train_score.score.as_matrix(columns=None).tolist()
        print(np.size(Y))
        print(np.shape(X))
        print(np.shape(XX))
```

```
18141
(18141, 270)
```

```
(18142, 270)
```

```
In [10]: from sklearn.utils import shuffle
         X, Y = shuffle(X, Y)
         from sklearn.model_selection import train_test_split
         X_train, X_validation, y_train, y_validation = train_test_split(X, Y, test_size=0.20,

         from keras.preprocessing import sequence
         maxlen = 270
         X_train = sequence.pad_sequences(X_train, maxlen=maxlen, dtype='float32')
         X_validation = sequence.pad_sequences(X_validation, maxlen=maxlen, dtype='float32')
         #X_train = np.any(np.isnan(X_train))
         #X_train = np.all(np.isfinite(X_train))
         print(X_train[1400].size)
```

```
Using TensorFlow backend.
```

```
270
```

```
In [11]: import os
         import time
         from sklearn import metrics
         from sklearn import preprocessing
         import numpy as np
         import pandas as pd
         import random
         import math
         import warnings
         warnings.filterwarnings("ignore")

         from matplotlib import pyplot as plt

         from sklearn import preprocessing
         from sklearn.ensemble import ExtraTreesClassifier
         from sklearn.feature_selection import SelectFromModel
         from sklearn.feature_selection import SelectPercentile
         from sklearn.feature_selection import chi2

         from sklearn import linear_model
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.linear_model import LogisticRegression
         from sklearn.ensemble import RandomForestClassifier
         from sklearn import tree
         from sklearn.ensemble import GradientBoostingClassifier
         from sklearn.svm import SVC
```

```
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.model_selection import cross_val_score
```

In [12]:
```
# Linear Regression Classifier
def linear_regression_classifier(train_x, train_y):
    model = linear_model.LinearRegression()
    model.fit(train_x, train_y)
    return model

# Multinomial Naive Bayes Classifier
def naive_bayes_classifier(train_x, train_y):
    model = MultinomialNB()

    param_grid = {'alpha': [math.pow(10,-i) for i in range(11)]}
    grid_search = GridSearchCV(model, param_grid, n_jobs = 1, verbose=1)
    grid_search.fit(train_x, train_y)
    best_parameters = grid_search.best_estimator_.get_params()

    model = MultinomialNB(alpha = best_parameters['alpha'])
    model.fit(train_x, train_y)
    return model


# KNN Classifier
def knn_classifier(train_x, train_y):
    model = KNeighborsClassifier()

    param_grid = {'n_neighbors': list(range(1,21))}
    grid_search = GridSearchCV(model, param_grid, n_jobs = 1, verbose=1)
    grid_search.fit(train_x, train_y)
    best_parameters = grid_search.best_estimator_.get_params()

    model = KNeighborsClassifier(n_neighbors = best_parameters['n_neighbors'], algori

    bagging = BaggingClassifier(model, max_samples=0.5, max_features=1 )
    bagging.fit(train_x, train_y)
    return bagging


# Logistic Regression Classifier
def logistic_regression_classifier(train_x, train_y):
    model = LogisticRegression(penalty='l2')
    model.fit(train_x, train_y)
    return model
```

```
# Random Forest Classifier
def random_forest_classifier(train_x, train_y):
    model = RandomForestClassifier()

    param_grid = {'n_estimators': list(range(1,21))}
    grid_search = GridSearchCV(model, param_grid, n_jobs = 1, verbose=1)
    grid_search.fit(train_x, train_y)
    best_parameters = grid_search.best_estimator_.get_params()

    model = RandomForestClassifier(n_estimators = best_parameters['n_estimators'])

    model.fit(train_x, train_y)
    return model


# Decision Tree Classifier
def decision_tree_classifier(train_x, train_y):
    model = tree.DecisionTreeClassifier()
    model.fit(train_x, train_y)

    bagging = BaggingClassifier(model, max_samples=0.5, max_features=1 )
    bagging.fit(train_x, train_y)
    return bagging


# GBDT(Gradient Boosting Decision Tree) Classifier
def gradient_boosting_classifier(train_x, train_y):
    model = GradientBoostingClassifier()

    model = RandomForestClassifier()

    param_grid = {'n_estimators': list(range(100,300,10))}
    grid_search = GridSearchCV(model, param_grid, n_jobs = 1, verbose=1)
    grid_search.fit(train_x, train_y)
    best_parameters = grid_search.best_estimator_.get_params()

    model = RandomForestClassifier(n_estimators = best_parameters['n_estimators'])

    model.fit(train_x, train_y)
    return model

# SVM Classifier
def svm_classifier(train_x, train_y):
    model = SVC(kernel='linear', probability=True)
    model.fit(train_x, train_y)
    return model
```

```
        # SVM Classifier using cross validation
        def svm_cross_validation(train_x, train_y):
            model = SVC(kernel='linear', probability=True)
            param_grid = {'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000], 'gamma': [0.001, 0.0001]}
            grid_search = GridSearchCV(model, param_grid, n_jobs = 1, verbose=1)
            grid_search.fit(train_x, train_y)
            best_parameters = grid_search.best_estimator_.get_params()
            #for para, val in best_parameters.items():
                #print para, val
            model = SVC(kernel='rbf', C=best_parameters['C'], gamma=best_parameters['gamma'],
            model.fit(train_x, train_y)
            return model

        def feature_select(x,y):
            clf = ExtraTreesClassifier()
            clf = clf.fit(x, y)
            model = SelectFromModel(clf, prefit=True)
            return model
```

In [13]:
```
        # just for my own record


        if __name__ == '__main__':
            thresh = 0.5
        #    model_save_file = "/home/jason/datamining/model/models"
        #    model_save = {}
        #    result_save_file = '/home/jason/datamining/result/results'

            test_classifiers = ['KNN','LR','RF','DT','GBC']
            classifiers = {
                           'KNN':knn_classifier,
                            'LR':logistic_regression_classifier,
                            'RF':random_forest_classifier,
                            'DT':decision_tree_classifier,
                            'GBC':gradient_boosting_classifier
            }

            print('reading training and testing data...')
            #X_train, X_validation, y_train, y_validation
            select_model = feature_select(X_train, y_train)
            X_train = select_model.transform(X_train)
            X_validation = select_model.transform(X_validation)

            result = []

            for classifier in test_classifiers:
                print('****************** %s ********************' % classifier)
```

```
                    start_time = time.time()
                    model = classifiers[classifier](X_train, y_train)
                    print('training took %fs!' % (time.time() - start_time))
                    predict = model.predict(X_validation)

                    precision = metrics.precision_score(y_validation, predict)
                    recall = metrics.recall_score(y_validation, predict)
                    print('precision: %.2f%%, recall: %.2f%%' % (100 * precision, 100 * recall))
                    accuracy = metrics.accuracy_score(y_validation, predict)
                    print('accuracy: %.2f%%' % (100 * accuracy))

                    scores = cross_val_score(model, X_train, y_train)
                    print(scores)
```

```
reading training and testing data...
****************** KNN ********************
Fitting 3 folds for each of 20 candidates, totalling 60 fits


[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:   16.6s finished


training took 16.900737s!
precision: 69.16%, recall: 100.00%
accuracy: 69.17%
[0.6920215  0.69588588 0.69051065]
****************** LR ********************
training took 0.241711s!
precision: 77.36%, recall: 90.43%
accuracy: 75.09%
[0.72219926 0.7366136  0.73433947]
****************** RF ********************
Fitting 3 folds for each of 20 candidates, totalling 60 fits


[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:   19.7s finished


training took 21.534177s!
precision: 79.11%, recall: 89.36%
accuracy: 76.33%
[0.73770153 0.7506719  0.74260906]
****************** DT ********************
training took 0.650436s!
precision: 71.19%, recall: 92.07%
accuracy: 68.75%
[0.68561389 0.66694232 0.68658259]
****************** GBC ********************
Fitting 3 folds for each of 20 candidates, totalling 60 fits
```

```
[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:  6.4min finished


training took 410.052062s!
precision: 79.57%, recall: 90.35%
accuracy: 77.29%
[0.75010335 0.76431673 0.76121563]
```

```
In [14]: X_train
         #X_train, y_train = X,Y

Out[14]: array([[ 0.      ,  1.      ,  0.      , ..., 11.774629, 11.737314,
                  11.739768],
                [ 0.      ,  0.      ,  0.      , ..., 11.737541, 12.21775 ,
                  11.256157],
                [ 1.      ,  0.      ,  0.      , ..., 11.562332, 11.451252,
                  11.531164],
                ...,
                [ 0.      ,  0.      ,  1.      , ..., 11.248144, 11.919338,
                  12.642514],
                [ 0.      ,  1.      ,  0.      , ..., 11.222754, 11.798913,
                  11.439027],
                [ 0.      ,  1.      ,  0.      , ..., 11.568027, 11.701103,
                  11.861883]], dtype=float32)
```

```
In [15]: #test cnn model

         from __future__ import print_function

         from keras.preprocessing import sequence
         from keras.models import Sequential
         from keras.layers import Dense, Dropout, Activation
         from keras.layers import Embedding
         from keras.layers import LSTM
         from keras.layers import Conv1D, MaxPooling1D
         from keras.datasets import imdb

         # Embedding
         max_features = 15000
         maxlen = 270
         embedding_size = 128

         # Convolution
         kernel_size = 5
         filters = 64
         pool_size = 3

         # LSTM
```

```
lstm_output_size = 70

# Training
batch_size = 30
epochs = 5

'''
Note:
batch_size is highly sensitive.
Only 2 epochs are needed as the dataset is very small.
'''

print('Loading data...')
#X_train, X_validation, y_train, y_validation
#(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

#X_train = np.asarray(np.abs(X))
X_train = np.asarray(np.abs(X_train))
X_validation = np.asarray(np.abs(X_validation))
print(len(X_train), 'train sequences')
print(len(X_validation), 'test sequences')

print('Pad sequences (samples x time)')
X_train = sequence.pad_sequences(X_train, maxlen=maxlen, padding='post')
X_validation = sequence.pad_sequences(X_validation, maxlen=maxlen, padding='post')
print('x_train shape:', X_train.shape)
print('x_test shape:', X_validation.shape)

print('Build model...')

cnnmodel = Sequential()
cnnmodel.add(Embedding(max_features, embedding_size,input_length=maxlen))
#model.add(Dense(32, activation='relu', input_dim=100))
cnnmodel.add(Dropout(0.5))
cnnmodel.add(Conv1D(filters,
                 kernel_size,
                 padding='valid',
                 activation='relu',
                 strides=1))
cnnmodel.add(MaxPooling1D(pool_size=pool_size))
cnnmodel.add(LSTM(lstm_output_size))
cnnmodel.add(Dense(1))
cnnmodel.add(Activation('sigmoid'))

cnnmodel.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
Loading data...
14512 train sequences
3629 test sequences
Pad sequences (samples x time)
x_train shape: (14512, 270)
x_test shape: (3629, 270)
Build model...
```

```
In [16]: X_train = cleaned_train[:18141]
         X_test = cleaned_train[18141:]
         y_train = train_score.score.as_matrix(columns=None).tolist()
```

```
In [17]: if __name__ == '__main__':
             thresh = 0.5
         #    model_save_file = "/home/jason/datamining/model/models"
         #    model_save = {}
         #    result_save_file = '/home/jason/datamining/result/results'

             test_classifiers = ['LR','RF','GBC']
             classifiers = {
                           'LR':logistic_regression_classifier,
                           'RF':random_forest_classifier,
                           'GBC':gradient_boosting_classifier
             }

             print('reading training and testing data...')
             #X_train, X_validation, y_train, y_validation
         #    X_test = rearrange(Xt)
         #    X_test = Xt
             select_model = feature_select(X_train, y_train)
             X_train = select_model.transform(X_train)
             X_test = select_model.transform(X_test)

             result = []

             start_time = time.time()
             model = classifiers['LR'](X_train, y_train)
             print('training took %fs!' % (time.time() - start_time))
             Y_predict_lr = model.predict_proba(X_test)[:,1]
             print('predict finished')

             model = classifiers['RF'](X_train, y_train)
             print('training took %fs!' % (time.time() - start_time))
             Y_predict_rf = model.predict_proba(X_test)[:,1]
             print('predict finished')

             model = classifiers['GBC'](X_train, y_train)
```

```
            print('training took %fs!' % (time.time() - start_time))
            Y_predict_gbc = model.predict_proba(X_test)[:,1]
            print('predict finished')


        #    for classifier in test_classifiers:
        #        print('****************** %s ******************' % classifier)
        #        start_time = time.time()
        #        model = classifiers[classifier](X_train, y_train)
        #        print('training took %fs!' % (time.time() - start_time))
        #        Y_predict = model.predict_proba(X_test)[:,1]
        #        print('predict finished')
```

```
reading training and testing data...
training took 0.241209s!
predict finished
Fitting 3 folds for each of 20 candidates, totalling 60 fits


[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:   25.5s finished


training took 27.739618s!
predict finished
Fitting 3 folds for each of 20 candidates, totalling 60 fits


[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:  8.2min finished


training took 546.078951s!
predict finished
```

```
In [18]: #cnn test
         X_train = cleaned_train[:18141]
         X_test = cleaned_train[18141:]
         y_train = train_score.score.as_matrix(columns=None).tolist()
         X_train = np.asarray(X_train)
         X_train = sequence.pad_sequences(X_train, maxlen=maxlen, padding='post')
         #X_test = np.asarray(Xt)
         X_test = sequence.pad_sequences(X_test, maxlen=maxlen, padding='post')


         cnnmodel.fit(X_train, y_train, batch_size=128, epochs=5)

         Y_predict_cnn = cnnmodel.predict(X_test, verbose=0)
         Y_predict_cnn = np.squeeze(Y_predict_cnn, axis=1)
```

```
Epoch 1/5
18141/18141 [==============================] - 60s 3ms/step - loss: 0.5846 - acc: 0.6992
Epoch 2/5
18141/18141 [==============================] - 59s 3ms/step - loss: 0.5381 - acc: 0.7351
Epoch 3/5
18141/18141 [==============================] - 59s 3ms/step - loss: 0.5243 - acc: 0.7417
Epoch 4/5
18141/18141 [==============================] - 62s 3ms/step - loss: 0.5139 - acc: 0.7487
Epoch 5/5
18141/18141 [==============================] - 65s 4ms/step - loss: 0.5042 - acc: 0.7539
```

```
In [19]: #nn train
         X_train = cleaned_train[:18141]
         X_test = cleaned_train[18141:]
         y_train = train_score.score.as_matrix(columns=None).tolist()
         X_array = np.asarray(X_train)
         Y_array = np.asarray(y_train)
         Xtest_array = np.asarray(X_test)
         import warnings
         warnings.filterwarnings("ignore")
         # Create your first MLP in Keras
         from keras.models import Sequential
         from keras.layers import Dense


         # fix random seed for reproducibility
         np.random.seed(7)

         # split into input (X) and output (Y) variables
         X = X_array
         Y = Y_array
         # create model
         nnmodel = Sequential()
         nnmodel.add(Dense(100, input_dim=270, activation='relu'))
         nnmodel.add(Dense(100, activation='relu'))
         nnmodel.add(Dense(100, activation='relu'))
         nnmodel.add(Dense(100, activation='relu'))
         nnmodel.add(Dense(1, activation='sigmoid'))
         # Compile model
         nnmodel.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         # Fit the model
         nnmodel.fit(X, Y, epochs=25, batch_size=150)
         # evaluate the model
         scores = nnmodel.evaluate(X, Y)
         print("\n%s: %.2f%%" % (nnmodel.metrics_names[1], scores[1]*100))
         Y_predict_nn = nnmodel.predict(Xtest_array, verbose=0)
         Y_predict_nn = np.squeeze(Y_predict_nn, axis=1)
```

```
Epoch 1/25
18141/18141 [==============================] - 1s 59us/step - loss: 0.6737 - acc: 0.6873
Epoch 2/25
18141/18141 [==============================] - 0s 26us/step - loss: 0.5794 - acc: 0.7171
Epoch 3/25
18141/18141 [==============================] - 0s 22us/step - loss: 0.5597 - acc: 0.7253
Epoch 4/25
18141/18141 [==============================] - 0s 22us/step - loss: 0.5516 - acc: 0.7259
Epoch 5/25
18141/18141 [==============================] - 0s 23us/step - loss: 0.5565 - acc: 0.7264
Epoch 6/25
18141/18141 [==============================] - 0s 22us/step - loss: 0.5491 - acc: 0.7297
Epoch 7/25
18141/18141 [==============================] - 1s 29us/step - loss: 0.5310 - acc: 0.7361
Epoch 8/25
18141/18141 [==============================] - 0s 23us/step - loss: 0.5338 - acc: 0.7349
Epoch 9/25
18141/18141 [==============================] - 0s 26us/step - loss: 0.5247 - acc: 0.7377
Epoch 10/25
18141/18141 [==============================] - 0s 23us/step - loss: 0.5242 - acc: 0.7415
Epoch 11/25
18141/18141 [==============================] - 0s 23us/step - loss: 0.5275 - acc: 0.7386
Epoch 12/25
18141/18141 [==============================] - 1s 29us/step - loss: 0.5212 - acc: 0.7438
Epoch 13/25
18141/18141 [==============================] - 0s 24us/step - loss: 0.5219 - acc: 0.7452
Epoch 14/25
18141/18141 [==============================] - 0s 26us/step - loss: 0.5212 - acc: 0.7422
Epoch 15/25
18141/18141 [==============================] - 0s 22us/step - loss: 0.5218 - acc: 0.7409
Epoch 16/25
18141/18141 [==============================] - 0s 22us/step - loss: 0.5178 - acc: 0.7452
Epoch 17/25
18141/18141 [==============================] - 0s 23us/step - loss: 0.5130 - acc: 0.7465
Epoch 18/25
18141/18141 [==============================] - 1s 28us/step - loss: 0.5088 - acc: 0.7495
Epoch 19/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.5174 - acc: 0.7449
Epoch 20/25
18141/18141 [==============================] - 0s 23us/step - loss: 0.5054 - acc: 0.7480
Epoch 21/25
18141/18141 [==============================] - 0s 23us/step - loss: 0.5094 - acc: 0.7462
Epoch 22/25
18141/18141 [==============================] - 0s 24us/step - loss: 0.5045 - acc: 0.7511
Epoch 23/25
18141/18141 [==============================] - 0s 25us/step - loss: 0.4998 - acc: 0.7534
Epoch 24/25
18141/18141 [==============================] - 0s 25us/step - loss: 0.5001 - acc: 0.7516
```

```
Epoch 25/25
18141/18141 [==============================] - 0s 25us/step - loss: 0.4994 - acc: 0.7551
18141/18141 [==============================] - 1s 28us/step

acc: 75.64%
```

```
In [20]: Y_predict_nn
```

```
Out[20]: array([0.8579925 , 0.94041866, 0.8313853 , ..., 0.70595163, 0.50523454,
                0.61891323], dtype=float32)
```

```
In [21]: Y_predict_cnn
```

```
Out[21]: array([0.8295582 , 0.78621113, 0.6645141 , ..., 0.64757854, 0.41389206,
                0.569912  ], dtype=float32)
```

```
In [22]: Y_predict_lr
```

```
Out[22]: array([0.73480094, 0.67819865, 0.68934352, ..., 0.62148617, 0.41585488,
                0.49107215])
```

```
In [23]: Y_predict_rf
```

```
Out[23]: array([0.77777778, 0.83333333, 0.61111111, ..., 0.66666667, 0.27777778,
                0.72222222])
```

```
In [24]: Y_predict_gbc
```

```
Out[24]: array([0.68636364, 0.79545455, 0.71818182, ..., 0.68636364, 0.52272727,
                0.58636364])
```

```
In [25]: Y_predict = []
         for i in range(len(Y_predict_rf)):
             Y_predict.append(( Y_predict_rf[i] + Y_predict_gbc[i] + Y_predict_cnn[i] + Y_predi
```

```
In [31]: Y_predict
```

```
Out[31]: [0.7879230195223683,
          0.8388544181079576,
          0.7062980920377404,
          0.943190951750736,
          0.768477741034344,
          0.7789613797508105,
          0.49860520630773875,
          0.7843875995790115,
          0.7212234910690423,
          0.7845466396122267,
          0.8631969074709247,
          0.3807709932026237,
```

```
0.6003489641830174,
0.40475087120677483,
0.8834678248323575,
0.2311255061265194,
0.873183729010399,
0.6564867117489227,
0.8245094442307347,
0.5769025559527705,
0.9085451350970701,
0.8365589516632485,
0.8573541283005416,
0.8478584218506862,
0.9570992681113156,
0.6973043681997242,
0.7669518869633627,
0.9678706323558635,
0.8102602714239949,
0.9404498155670937,
0.8150021161394889,
0.7075678125926943,
0.8056072395558309,
0.6466429714602653,
0.720919211645319,
0.45648324354128406,
0.6798729751748268,
0.9523854328827425,
0.5096632492963714,
0.8012050654852029,
0.5585386865367793,
0.8966844571961297,
0.9577842205762863,
0.6449160896166406,
0.5598847478327125,
0.8654995611821762,
0.7667021310690678,
0.31053749369551437,
0.9412690380606988,
0.6922423969314556,
0.6682960966018715,
0.8878353593927442,
0.7403141237569577,
0.5666097553390446,
0.34242223821806184,
0.5745137103579261,
0.5056084916597665,
0.8443176256887841,
0.31077154433034887,
0.652228018490955,
```

```
0.7001406736747183,
0.5219168139377026,
0.40930632921782406,
0.656459810562206,
0.3910168477983186,
0.9005869375936912,
0.6207800335956342,
0.8155419999300831,
0.7215054362410247,
0.674287120471097,
0.4946316438792932,
0.44189750848695486,
0.38850753680624145,
0.670125709790172,
0.8459431837604503,
0.456926736356032,
0.649020043858374,
0.6364261954119711,
0.35317333382789534,
0.49661802042915365,
0.5756961399557614,
0.3289054932768899,
0.2922133699661553,
0.5969380459099105,
0.6982537371642662,
0.6672041084880781,
0.9109541324051944,
0.38655630711353184,
0.9205146239562468,
0.7611620288605642,
0.847835368550185,
0.7250494646002548,
0.6386729278347709,
0.8814101485591946,
0.9426997240143593,
0.8593327081865735,
0.8854973321009164,
0.761613302670344,
0.8395708713567618,
0.7394665759320211,
0.8789345564866307,
0.9001224914584497,
0.7608284189243509,
0.890749600379154,
0.8078777135020554,
0.8980867635120044,
0.6896643446852462,
0.9010148370506788,
```

```
0.5399896084359198,
0.9436106944626028,
0.7173233939240677,
0.5830297254853778,
0.910392844135111,
0.840965419947499,
0.7943909067698199,
0.6443230259900141,
0.2854903297592895,
0.7789211454114529,
0.5589566548063297,
0.47925539472789475,
0.7434961258461981,
0.8142665823601714,
0.6737935066524179,
0.32949335013676173,
0.7789133901848937,
0.8469757748974693,
0.8763380272520913,
0.5939704627099663,
0.534669560449894,
0.8890788649970836,
0.9061540941999416,
0.8587573018639979,
0.6894106894129455,
0.42708793359272407,
0.9092364098387535,
0.7932932148979168,
0.9141534354048546,
0.8778116136488288,
0.31902512615979317,
0.5606782776689289,
0.799687826181903,
0.5312301490945046,
0.8372125069300334,
0.6845006552910564,
0.7081792589389917,
0.42824005556076467,
0.8511270585084202,
0.8249551722196617,
0.8076245030068387,
0.9575832835652611,
0.8483461627755502,
0.6348126131928329,
0.4342086132728692,
0.910660274432163,
0.7441339467812066,
0.5487191449813169,
```

```
0.9428483106873252,
0.6458503601828007,
0.6478715577510872,
0.6748445736037361,
0.6227878210821537,
0.47721030353897753,
0.38173084201836827,
0.5624694144936523,
0.4064605542204597,
0.42283884106260355,
0.7142826438552201,
0.25735870097773245,
0.300363426979142,
0.8148475043701403,
0.7483688134737689,
0.18327714607420592,
0.187424812940034,
0.5018928777238335,
0.8774436646940732,
0.7461095380060601,
0.5821547227074402,
0.7574544534237698,
0.9182699016248337,
0.4044226799348388,
0.9030570583813118,
0.9534332047809254,
0.6053482233273862,
0.8338723847059288,
0.6746717732663107,
0.4710536087101156,
0.7165282924668958,
0.5253831539792244,
0.5434089610672961,
0.5374659979584241,
0.4923096076707647,
0.9242050391556036,
0.7847563390779977,
0.9297017475270262,
0.24365215433968437,
0.8239973632675228,
0.8743608000001521,
0.7637416302254706,
0.8781491487917273,
0.5730877148834141,
0.4451484339857342,
0.8534584199238305,
0.8504844195312924,
0.9633799940347672,
```

```
0.8727364252613048,
0.5896017647752858,
0.7880152231515056,
0.7866893572638733,
0.6372686437585138,
0.557448315635474,
0.5232809077308636,
0.6768979428392468,
0.41333027170463044,
0.6733820088885047,
0.3901863222471391,
0.37588769712231374,
0.6202792083073144,
0.6115061372819574,
0.9277016359146195,
0.7874259028470878,
0.5903055927970193,
0.674875012612102,
0.5501333484745989,
0.8095936993757884,
0.7760574921514048,
0.8920446934724094,
0.7165780002420599,
0.7187719467613432,
0.8814212735253151,
0.5807313810845818,
0.3672774839130315,
0.3771320255717846,
0.7178939505056902,
0.23993113250518688,
0.8084332549511785,
0.6688980896665593,
0.5472185995843676,
0.4059818581649751,
0.4196123341117242,
0.48654953126955514,
0.48960989256097814,
0.8181980276348615,
0.6550325908444145,
0.4042796959178616,
0.26801735141662636,
0.32224028688864875,
0.792819875689468,
0.5036875356929471,
0.41166821430728895,
0.9129034184446239,
0.6911988183103427,
0.682110450574846,
```

```
0.8094428915267039,
0.7789399079903208,
0.8764303402166174,
0.8697151045907627,
0.8709752200227795,
0.8973718220236325,
0.8913357615470886,
0.7886683804519249,
0.9222270407459953,
0.3868860996281258,
0.8841624048322139,
0.93529912693934,
0.7465166455567485,
0.42523248632146854,
0.9372259506974557,
0.9483359392243202,
0.8820688567077271,
0.7292473619935489,
0.7551258529376502,
0.6753964922644875,
0.866395427933847,
0.9497094652869484,
0.7964538020919067,
0.9300987685268576,
0.9003800843099151,
0.4278203835391035,
0.9267455072414995,
0.47436560932735,
0.8086494818781361,
0.7873489572845325,
0.8615791296116029,
0.7466026792923609,
0.689918143219418,
0.6607753328602723,
0.4456969882653217,
0.749690659268938,
0.7708888434099429,
0.734322137934993,
0.5580578808230583,
0.7534403522207279,
0.5241464212837845,
0.8705191810022701,
0.42413503350031495,
0.35287840304651646,
0.6580460749641813,
0.7663515042777014,
0.6625641684339504,
0.5929596339211319,
```

0.32459908910471985,
0.889376096924146,
0.9109267601762155,
0.9192031358829652,
0.8022899895304381,
0.9361279361476802,
0.7880824397007624,
0.5849414509053182,
0.8972358349898849,
0.8594265683130784,
0.8043501635091473,
0.7545945063684926,
0.9132316573099657,
0.9459113776683807,
0.4982894633454506,
0.9029795646366446,
0.8340057898350436,
0.44305764192884617,
0.8350959051137019,
0.8564420763591323,
0.8746259006285908,
0.7906884379760184,
0.8366959465874566,
0.5337753916780154,
0.8058989164203104,
0.6716750606442943,
0.809103346743969,
0.8884953575904924,
0.5002396085045555,
0.7149070091319807,
0.4350913967448051,
0.8789101107554003,
0.46671295701855364,
0.597493737183436,
0.7157944269252545,
0.30779319616279216,
0.31967834889738245,
0.3729706623036452,
0.32380542139513324,
0.4057966457924458,
0.4992691194168245,
0.47575669637834184,
0.5767367167906328,
0.2931530572699778,
0.8955279203677418,
0.384741752986053,
0.855808419621352,
0.42032031480110055,

```
0.9209269355643879,
0.6788103330616999,
0.47227998574574787,
0.4001841742884029,
0.5892985716462136,
0.9187423077195582,
0.6371719511169376,
0.8178846693099147,
0.7647785050399376,
0.8042988872889316,
0.757292119661967,
0.888607545573302,
0.8474724759658178,
0.9578534979711879,
0.4961389179783638,
0.7548413442240821,
0.4101055415892842,
0.6277019953486895,
0.5365940389157545,
0.3704177319551959,
0.5888153023792035,
0.5242236738554156,
0.9496873981422849,
0.8443300134605831,
0.6415797694463923,
0.9593277966434306,
0.6091647285403627,
0.8655457016795572,
0.7507855719689167,
0.7546051210225231,
0.7758263726728131,
0.3751059223907163,
0.3423158787416689,
0.8653505594441385,
0.7227092297992321,
0.43182052370875773,
0.8085773186852234,
0.6896097604373489,
0.6933330390188429,
0.6969068336817953,
0.3701540231855229,
0.9184647465294058,
0.13544006724700783,
0.9093469715479648,
0.5321388243424772,
0.4576946802964114,
0.727243339142414,
0.4438348039232119,
```

```
0.9637766320597042,
0.9689246118068695,
0.8169049958990078,
0.919564668668641,
0.20554686580794027,
0.6648080778242361,
0.5578568451633357,
0.6442159159918024,
0.4795824647551835,
0.743976118980032,
0.464428401655621,
0.32222290223445554,
0.7983155416719842,
0.7495141196732569,
0.49164667039206533,
0.8083746325789076,
0.7292885642762136,
0.6584220804048306,
0.3972718074917793,
0.1028292984447696,
0.05326840234073726,
0.7955834566944777,
0.7289211059760565,
0.7083548713212061,
0.8309462668919804,
0.8010857508339063,
0.9074121671493607,
0.7328024386757552,
0.7808678450608494,
0.6055613092400811,
0.8571539946276732,
0.4696546486832879,
0.28327577216456634,
0.9422403510170754,
0.7699714381586422,
0.8715417295092285,
0.9366658614440397,
0.9321278364670397,
0.6327915799437147,
0.9014507049863989,
0.6959251076585115,
0.595489101202199,
0.4356335046766984,
0.8150362147225274,
0.9268806893717159,
0.3257427946636171,
0.8197516848944655,
0.7197908465609406,
```

```
0.10365140590478074,
0.8157847303633738,
0.5447167629545385,
0.558878982217625,
0.9167099753413538,
0.7278378998089319,
0.7693455044067268,
0.3853273902427067,
0.734401288688785,
0.7713853182816746,
0.8355310353064778,
0.9070854442890244,
0.8348229946813198,
0.6997885297043155,
0.8324414808340747,
0.6628010556252316,
0.3103325757339145,
0.7936818902841722,
0.8793618994529802,
0.4081586702152936,
0.7434379329284032,
0.20183814443422085,
0.6864081867415496,
0.9084792191031004,
0.6237328369659607,
0.7192723036414446,
0.5657037662285747,
0.5498628740960901,
0.7007513774765862,
0.6705296686502418,
0.8905225707425011,
0.8143293269354888,
0.46385399267227967,
0.8660266120024401,
0.9071223375171122,
0.9121611044864462,
0.7801007724169529,
0.8090312107945934,
0.9398656336948125,
0.9264702871291324,
0.7470003828857884,
0.849475904305776,
0.8143595014858728,
0.9029330836101012,
0.9118548187041523,
0.8930209364252861,
0.8769379066698479,
0.8485732344966946,
```

```
0.4375465753102543,
0.8197648066763926,
0.9108869037844918,
0.6449862662288878,
0.9186910711755656,
0.658431378849829,
0.7887469777254144,
0.7775346564223068,
0.35678259522625894,
0.42267902166855453,
0.83986246649063,
0.7070821835838184,
0.894167574126311,
0.5859400462622595,
0.8905502291941884,
0.33554185865202335,
0.37886788098499025,
0.3440730490016215,
0.729179378621506,
0.6898022575510873,
0.2878511454345602,
0.7443106286453478,
0.2915862341571336,
0.8045872831585431,
0.35082990915486306,
0.7108257103146929,
0.6259022390902644,
0.4495347396291868,
0.741992192828294,
0.6463522977901227,
0.5807271829759232,
0.6020084450642268,
0.7261942840886838,
0.5173846330004509,
0.36835122415513705,
0.7860754937535585,
0.7891653306857505,
0.6723196595004111,
0.45558334614592366,
0.9599750445647673,
0.5629204741030028,
0.9503787873670309,
0.859867684768908,
0.8321789311640191,
0.9202518269269153,
0.8649194291745773,
0.7552853695069901,
0.7954299054061523,
```

0.8355315815017681,
0.9473099441540361,
0.7726764951992516,
0.8898822652872163,
0.6314235114388995,
0.3838155296715823,
0.5239775431276572,
0.8985324497776802,
0.8374498656903855,
0.7955200344926179,
0.7532314709039649,
0.8690880518971067,
0.8622094330161509,
0.8181074752952113,
0.8112900867305621,
0.8614388316568702,
0.38136518137021497,
0.48709615276317403,
0.5635666191878945,
0.42303621169894634,
0.6369227657715479,
0.6419386001247348,
0.24425826901287745,
0.7132078319185913,
0.8307586353836638,
0.7492448065317039,
0.6531701219503325,
0.6608176450386192,
0.49004978992120185,
0.5269612772597208,
0.5462466916050573,
0.9082844938593682,
0.6020829531731028,
0.7255329367488321,
0.7802642231035715,
0.7156682534952357,
0.6562530000251954,
0.9009111145530084,
0.9601631183515895,
0.5610611627047712,
0.9158598087050698,
0.41309251372862343,
0.4526897855930858,
0.4782074906007208,
0.43593470603227613,
0.6557854638256208,
0.30235017770619105,
0.32731815435067574,

0.38970013107314255,
0.6547851233169286,
0.5663105560974642,
0.26096761526182444,
0.6606358373706991,
0.8017281283934912,
0.47105137823206006,
0.5252332033232006,
0.3625335262128801,
0.47551747136043776,
0.45711930774980125,
0.05902095386250453,
0.787687869866689,
0.3052899274485882,
0.7819288205618811,
0.38294206514503015,
0.8429615298003862,
0.7520125616078425,
0.7890625485868166,
0.5200500780584836,
0.23488469391007616,
0.8324995105314736,
0.8896085299927778,
0.858211874118959,
0.8270981555033212,
0.4496875521811572,
0.9211600911737692,
0.8220193033567582,
0.78929760194186,
0.5177481925999275,
0.7844253030389247,
0.8484801931212647,
0.8001171421824079,
0.41480516333772677,
0.8618271171745627,
0.854580536454615,
0.8593842302910005,
0.8247756405310197,
0.9546056005087766,
0.8031388124131194,
0.9424343022433195,
0.8511320826080111,
0.827624569818227,
0.7866125786244267,
0.9148824332639425,
0.7902558701809006,
0.6598129751104297,
0.7823420165163097,

```
0.3218209906659945,
0.3871970508134726,
0.730774763045889,
0.5063675195129231,
0.7091709606575244,
0.8933122584615091,
0.8754273354706138,
0.31748977554867963,
0.7404434863064024,
0.5334188088624164,
0.564526966771092,
0.7394960466358397,
0.326452602551441,
0.7236637247632248,
0.5851711707283752,
0.6229759257550191,
0.8900834606452421,
0.9314379628258522,
0.8714868636143328,
0.6038995921611786,
0.8013794960999729,
0.7845880152902218,
0.40001558647914365,
0.5621602526367313,
0.8281177141148635,
0.43678953333033455,
0.7717404727983956,
0.3562752952328836,
0.4038624115815066,
0.34944907285348337,
0.5136231831077374,
0.9200076937675477,
0.9050677021645537,
0.8558018631104267,
0.9258658757113447,
0.9111968085019275,
0.8136884852792278,
0.5506491722031073,
0.7393131019792172,
0.3964356944868059,
0.9114408005066592,
0.9214841476895592,
0.7854384439160125,
0.6253934148285124,
0.9224556067977289,
0.8777651586014815,
0.936825275963003,
0.7993254716348166,
```

```
0.7260014947616693,
0.6477493209369255,
0.79232004673192,
0.7647254532936848,
0.9169420971111818,
0.8550469211255661,
0.864100040058897,
0.792993063336671,
0.8191558237328673,
0.2435109046297242,
0.7884945525665477,
0.847916167131578,
0.17336645784853685,
0.8055338214443187,
0.9100347671845946,
0.8663167132271661,
0.8811299311392236,
0.7913085504914775,
0.39716114316022755,
0.3771075626515379,
0.9341997802257538,
0.6458960713461193,
0.7908398753765857,
0.6274451381931401,
0.9022722799368579,
0.8700531182867108,
0.3597250626845793,
0.7462744343461413,
0.6671692872589284,
0.5943291255620995,
0.13912622679514114,
0.3686602446617502,
0.5297194616963166,
0.8720610153494459,
0.4793304104395587,
0.7536379430029128,
0.8493370020630384,
0.9153751092727738,
0.48565360783326506,
0.935922782258554,
0.9070489707017185,
0.7644820706109807,
0.4658168690975266,
0.7927372709669248,
0.9703926411542025,
0.7349012274934787,
0.4293954891387862,
0.9176871607098916,
```

```
0.5404468921549392,
0.9656298734925011,
0.5396649338831805,
0.6040047725643775,
0.44933132660208325,
0.527478818098704,
0.6002599843975269,
0.6500359291380102,
0.8344554177137337,
0.8135585780095572,
0.9083320323565993,
0.803929597350082,
0.8585041982055914,
0.8743599624344797,
0.6980289100697546,
0.7136686215497027,
0.6902662993079485,
0.5050955508995538,
0.8792023096120719,
0.7476489152872201,
0.9401367968983121,
0.7270474076873124,
0.9757437099109997,
0.5213362146206577,
0.6655629533709901,
0.5157555628906597,
0.8286846267454552,
0.6608413184230978,
0.8418127854966154,
0.33477355467550685,
0.4960052791568968,
0.3419775910901301,
0.2896699372415591,
0.3562328591190203,
0.48902776903576317,
0.7169425722324487,
0.8259694466085146,
0.5589403622379207,
0.9461627973751588,
0.5835951017309922,
0.589495322424354,
0.33038517508392384,
0.4673619196571485,
0.850788276544725,
0.6934460300688792,
0.4858258688841203,
0.7777762259497787,
0.976058808781884,
```

```
0.9675060776146975,
0.9027483421142655,
0.599894448572939,
0.7160207851968631,
0.44587824385274544,
0.8329481082432197,
0.9242391085082835,
0.7562096073771968,
0.6336719548762446,
0.7749986921597009,
0.9167070532386953,
0.8129135453941846,
0.8702816483348307,
0.10495512248440222,
0.033144038322974335,
0.5834954345768149,
0.7578388076538991,
0.893047513835358,
0.7188508684888031,
0.9511283324523405,
0.4256792176552493,
0.271090364584116,
0.44630487637989447,
0.7574264776827109,
0.6591650060180462,
0.7372200908684972,
0.9177856813777576,
0.2656013273003728,
0.7537123854413177,
0.6406325691276127,
0.6621558812531558,
0.8076169346920168,
0.30865597390767296,
0.842190174863796,
0.778767822336669,
0.273904467298828,
0.7715323046602384,
0.8602748858507233,
0.726620542130085,
0.6108839739753742,
0.7364188602777443,
0.6916003118259738,
0.863901778513735,
0.8597175928378346,
0.8432592916067201,
0.501890183714303,
0.21940075374688162,
0.3287073458234469,
```

0.9025375494451234,
0.9228275938467546,
0.6783093330384504,
0.7730840746200446,
0.736280726242547,
0.7294487932414719,
0.5917655890337145,
0.916839751842046,
0.4268235251608521,
0.21054683601615404,
0.42905028406718765,
0.6716274393327308,
0.7520833444113684,
0.5900950534777207,
0.520497067558645,
0.6018791823525622,
0.44106063686235986,
0.7724467981343317,
0.5778893949106486,
0.44469946661982873,
0.5417783844651598,
0.9042863611320052,
0.26785599322180553,
0.748206107243143,
0.4480677302437599,
0.31609608740216555,
0.8160831307521974,
0.9181353016032112,
0.12824774717893261,
0.8549439351366024,
0.629485568855748,
0.5954501293674864,
0.8088600411559597,
0.8499969636551057,
0.9267046536761101,
0.5099593178641917,
0.8970545065222364,
0.8929224230725356,
0.8085074072233354,
0.5513218885419344,
0.48961812626532836,
0.6030695152854679,
0.649836534984184,
0.8866916691414033,
0.8468568741974204,
0.7661621690398515,
0.7346226676545962,
0.3825571549397827,

0.4901130429873563,
0.43762050022681553,
0.5322285868302741,
0.651788144521039,
0.8478331111597293,
0.6329213119516469,
0.6350479079015328,
0.7124549667040507,
0.8463533117915645,
0.539778175965102,
0.4338415113362399,
0.618502124105439,
0.44856004728512333,
0.09647370129823685,
0.4791076360025791,
0.7272516473074152,
0.5179096732928296,
0.8310377510509106,
0.8222080749393713,
0.4570708885487884,
0.9518342633139003,
0.8716532677412033,
0.4343061544527911,
0.6532043250221194,
0.5621861278709739,
0.7148254750503433,
0.7891695371480903,
0.9193875507874922,
0.35620876559705444,
0.8274226472835348,
0.5733025091766106,
0.5985998908979724,
0.8921740058696631,
0.8444695000997697,
0.5334795136192833,
0.4432387126666127,
0.547464120674013,
0.8537881079045209,
0.7934244191706783,
0.41874417483505577,
0.5908398338791095,
0.9263035312144443,
0.898107785498253,
0.6112812922458456,
0.9010241892903742,
0.8553693280978636,
0.8964370949400795,
0.185333118836085,

```
0.534944370840535,
0.6686003845448446,
0.5835403515533968,
0.6973734078985272,
0.6942390414199444,
0.7148904488243237,
0.8220461128336011,
0.7831209435306414,
0.7586249789505294,
0.9199771710417488,
0.6223448736198021,
0.5099099472165107,
0.8874459596896412,
0.962796746600758,
0.3573729398124146,
0.387807725896739,
0.6822845463499878,
0.8120342117668402,
0.6132718926126307,
0.5349965954669799,
0.6653914597299364,
0.2882020725520572,
0.7743458351703605,
0.7350094733515171,
0.6838421782760908,
0.7569975134700235,
0.4411341812273469,
0.7142410216909466,
0.6375649080131993,
0.6912523763348357,
0.36259249076247213,
0.6160562602257488,
0.43535042019805525,
0.6808919955985715,
0.7052298425725012,
0.6327057378159628,
0.7820889674051843,
0.45044085926780797,
0.6856062317285875,
0.5227355614455059,
0.7840680001058964,
0.7814689537792494,
0.8521055931093717,
0.8143975616404504,
0.8166510367634321,
0.5592338114374815,
0.8636048674583435,
0.46555604450028354,
```

```
          0.8021451380818782,
          0.5838297293041692,
          0.9007595030948369,
          0.751490477088726,
          0.9144833569875871,
          0.5356153687142362,
          0.8782710076582552,
          0.3201451778110832,
          0.8058657756357481,
          0.6573308645775824,
          0.7995697505245305,
          0.9229980880262876,
          0.9121274190719681,
          0.8425156342561799,
          0.8987830066921735,
          0.4438310125861505,
          0.8963282037865032,
          0.7028695393993397,
          0.867933125929399,
          0.7304127310562616,
          0.578213436404864,
          0.3260228379733032,
          0.8932427215154725,
          0.9148329760390099,
          0.5005388915839821,
          0.8016761651544859,
          0.8241777198483246,
          0.8153525488846229,
          ...]
```

```
In [37]: Y_predict_temp = []
         for i in range(len(Y_predict_rf)):
             Y_predict_temp.append((Y_predict_rf[i] + Y_predict_gbc[i]) / 2.0)
```

```
In [40]: Y_predict_temp
```

```
In [35]: temp_score = load_label('submission.csv')
         submit_score  = temp_score
         submit_score['score'] = Y_predict
         submit_score.to_csv('predict_result_cnn.csv')
```

```
In [41]: temp_score = load_label('submission.csv')
         submit_score  = temp_score
         submit_score['score'] = Y_predict_temp
         submit_score.to_csv('predict_result_ncnn.csv')
```

```
In [ ]: table = pd.pivot_table(train_file,values = 'price', index=['lvl1','lvl2','lvl3'],colum
        table
```