447final

December 5, 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 scipy.sparse import csr_matrix
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
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, 'descrpt
        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 [ ]:
```
'''
def text_embedding(column,vecsize):
    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=vecsize, workers=4)

    avg_data = []
    for row in splitted:
        vec = np.zeros(vecsize, 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)


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

            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)
            i = i + 1

        return tf_w_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

    temp = pd.read_csv('train_data.csv',delimiter=',', header=0)
    description_X = temp.descrption.str.lower().replace('<.*?>','',regex=True).replace('[^
    #description_X.head()
    tfidf_n = TfidfVectorizer(max_features=1000,stop_words = 'english')
    tf_out = tfidf_n.fit_transform(description_X.astype('U')).toarray()
```

```
        #lg_array = csr_matrix(tf_out, dtype=np.int8).toarray()
        #lg_array = np.vstack( lg_array )

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

        print(tf_out.shape)
        '''
```

In [4]:
```
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 = 500.
#    filename['price'] = filename['price'].clip(lower=0.)
    filename['price'] = filename['price'].clip(lower=0.,upper=valuethred).div(valuethre
    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)
    tfidf_n = TfidfVectorizer(max_features=1000,stop_words = 'english')
    lg = tfidf_n.fit_transform(description_X.astype('U')).toarray()
    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[:5]
    U_new = U[:, :5]
    new_lg = np.dot(U_new, np.diag(s_new))
    outfile = np.column_stack((outfile,new_lg))


    name_X = filename.name.str.lower().replace('<.*?>','',regex=True).replace('[^\w\s]+
    tfidf_f = TfidfVectorizer(max_features=500,stop_words = 'english')
    lf = tfidf_f.fit_transform(name_X.astype('U')).toarray()
    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[:5]
    Uf_new = Uf[:, :5]
    new_lf = np.dot(Uf_new, np.diag(sf_new))
    outfile = np.column_stack((outfile,new_lf))


    return outfile
```

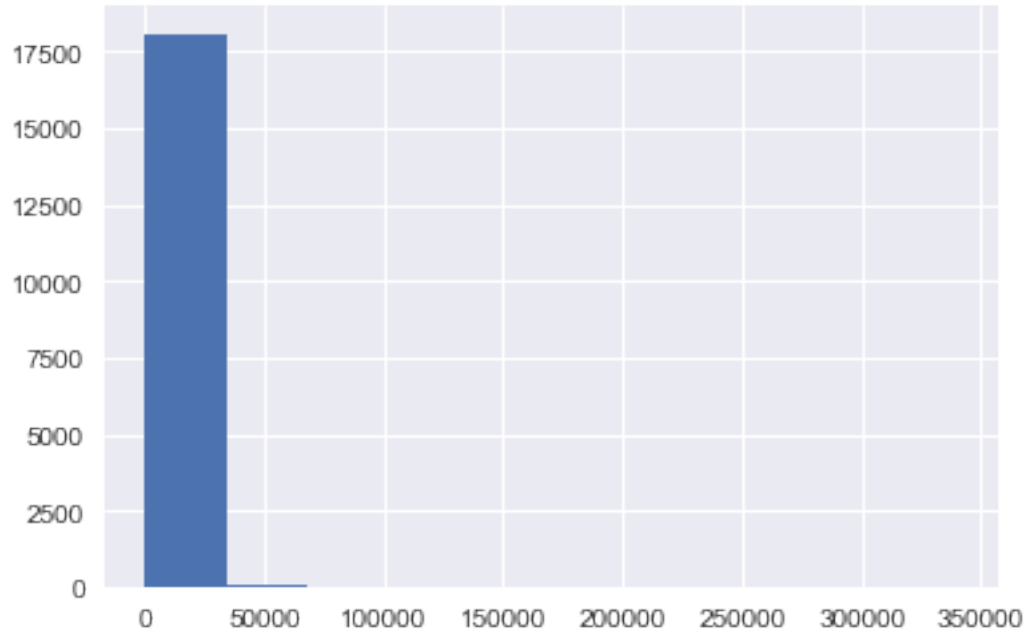
In [ ]:
```
'''
train_file = pd.read_csv('train_data.csv',delimiter=',', header=0, nrows=10)
train_file.columns = ['id', 'name','lvl1','lvl2','lvl3','descrption','price','type']
description_X = train_file.descrption.str.lower().replace('<li>','final ',regex=True).
count_descrption = description_X.str.count('final')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

'''
```

In [5]:
```
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[5]: (36283, 264)



In [6]:
```
'''
# test cell
cleaned_train['descrption'] = temp['descrption']
'''
cleaned_train
```

Out[6]:
```
array([[ 0.        ,  0.        ,  0.        , ...,  0.01040592,
         0.00706059, -0.01183207],
       [ 0.        ,  1.        ,  0.        , ..., -0.11074271,
         0.1532164 ,  0.04451235],
       [ 0.        ,  0.        ,  0.        , ..., -0.08138973,
        -0.07558503,  0.00685938],
       ...,
       [ 0.        ,  0.        ,  0.        , ..., -0.0272351 ,
         0.02885672,  0.0097179 ],
       [ 0.        ,  0.        ,  0.        , ...,  0.13301183,
        -0.05339164,  0.20656577],
       [ 0.        ,  0.        ,  0.        , ...,  0.09247052,
         0.02570214,  0.12365277]])
```

```
In [7]: pca = PCA(n_components=150, svd_solver='full')
        cleaned_train_temp = pca.fit_transform(cleaned_train)
        cleaned_train_min = np.min(cleaned_train_temp)
        cleaned_train_out = cleaned_train_temp - (cleaned_train_min)*np.ones_like(cleaned_trai
        print(cleaned_train_out)
        cleaned_train = cleaned_train_out
```

```
[[0.01532479 4.8106171  5.0433837  ... 5.04587077 5.18875357 4.8538586 ]
 [6.03212141 4.90830981 5.15302028 ... 5.05407926 5.04850023 5.05973097]
 [5.01505303 6.56541841 5.3675697  ... 5.05227194 5.05323505 5.05447177]
 ...
 [6.01793131 4.82661788 5.16436038 ... 5.05505595 5.04146352 5.06648967]
 [0.0149594  4.72873787 5.33765066 ... 5.05584353 5.03698275 5.06676065]
 [6.01991194 4.75962606 5.3783636  ... 5.0562746  5.00421698 5.09565824]]
```

```
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)
        w,b = np.shape(np.array(X))
        print(np.shape(np.array(X)))
        Y = train_score.score.as_matrix(columns=None).tolist()
        b
```

```
(36283, 150)
```

Out[8]: 150

```
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, 150)
```

```
(18142, 150)
```

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,


#maxlen = b
#X_train = sequence.pad_sequences(X_train, maxlen=maxlen, dtype='float32')
#X_validation = sequence.pad_sequences(X_validation, maxlen=maxlen, dtype='float32')
print(X_train[1400].size)
```

```
150
```

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.tree import DecisionTreeRegressor
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 = []

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

```
            regressor.fit(train_x, train_y)
            predict = model.predict(X_validation)
            '''


            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))
                logloss = metrics.log_loss(y_validation, predict)
                print('loss: %.2f' % (logloss))

                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:   18.2s finished


training took 18.351413s!
accuracy: 68.97%
loss: 10.72
******************** LR ********************
training took 0.309603s!
accuracy: 73.71%
loss: 9.08
******************** RF ********************
Fitting 3 folds for each of 20 candidates, totalling 60 fits


[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:   36.2s finished


training took 39.324011s!
accuracy: 72.53%
loss: 9.49
******************** DT ********************
```

```
training took 1.095098s!
accuracy: 64.70%
loss: 12.19
******************** GBC ********************
Fitting 3 folds for each of 20 candidates, totalling 60 fits


[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed: 11.0min finished


training took 699.363715s!
accuracy: 75.70%
loss: 8.39


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

Out[14]: array([[6.02446008, 6.13571744, 4.96386001, ..., 5.05651347, 5.0570877 ,
                 5.02440727],
                [3.02808194, 4.90267968, 5.23573949, ..., 5.06352585, 5.05271443,
                 5.07289584],
                [6.04280844, 4.76798545, 4.34700411, ..., 5.06156029, 5.04765451,
                 5.02266416],
                ...,
                [4.03428522, 4.71609443, 4.33737427, ..., 5.05837053, 5.05061787,
                 5.04283774],
                [8.02264955, 4.83231627, 5.14338912, ..., 5.09236285, 5.04149679,
                 5.05960442],
                [8.02509883, 4.81465775, 5.10575448, ..., 5.0520636 , 5.0539802 ,
                 5.05392005]])

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 = 5000
         maxlen = b
         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'])
```

```
Using TensorFlow backend.
```

```
Loading data...
14512 train sequences
3629 test sequences
Pad sequences (samples x time)
x_train shape: (14512, 150)
x_test shape: (3629, 150)
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','DT']
             classifiers = {
                           'LR':logistic_regression_classifier,
                           'RF':random_forest_classifier,
                           'GBC':gradient_boosting_classifier,
                           'DT':decision_tree_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')

            model = classifiers['DT'](X_train, y_train)
            print('training took %fs!' % (time.time() - start_time))
            regressor = DecisionTreeRegressor()
            regressor.fit(X_train, y_train)
            Y_predict_dt = regressor.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 0.399461s!
predict finished
Fitting 3 folds for each of 20 candidates, totalling 60 fits


[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:   42.2s finished


training took 46.672847s!
predict finished
Fitting 3 folds for each of 20 candidates, totalling 60 fits


[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed: 13.4min finished


training took 910.524079s!
predict finished
training took 912.883881s!
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-17-128d44ea6611> in <module>()
     43     regressor = DecisionTreeRegressor()
     44     regressor.fit(X_train, y_train)
---> 45     Y_predict_dt = regressor.predict_proba(X_test)[:,1]
     46     print('predict finished')
     47


AttributeError: 'DecisionTreeRegressor' object has no attribute 'predict_proba'
```

```
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=8)

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

```
Epoch 1/8
18141/18141 [==============================] - 32s 2ms/step - loss: 0.6222 - acc: 0.6852
Epoch 2/8
18141/18141 [==============================] - 30s 2ms/step - loss: 0.6133 - acc: 0.6882
Epoch 3/8
18141/18141 [==============================] - 32s 2ms/step - loss: 0.6063 - acc: 0.6882
Epoch 4/8
18141/18141 [==============================] - 32s 2ms/step - loss: 0.5964 - acc: 0.6883
Epoch 5/8
18141/18141 [==============================] - 29s 2ms/step - loss: 0.5769 - acc: 0.6891
Epoch 6/8
18141/18141 [==============================] - 30s 2ms/step - loss: 0.5598 - acc: 0.6954
Epoch 7/8
18141/18141 [==============================] - 30s 2ms/step - loss: 0.5487 - acc: 0.7194
Epoch 8/8
18141/18141 [==============================] - 31s 2ms/step - loss: 0.5453 - acc: 0.7223
```

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=b, activation='relu'))
nnmodel.add(Dense(100, activation='relu'))
nnmodel.add(Dropout(0.5))
nnmodel.add(Dense(50, activation='sigmoid'))
nnmodel.add(Dropout(0.5))
nnmodel.add(Dense(25, 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 72us/step - loss: 0.6433 - acc: 0.6755
Epoch 2/25
18141/18141 [==============================] - 1s 35us/step - loss: 0.6280 - acc: 0.6881
Epoch 3/25
18141/18141 [==============================] - 1s 32us/step - loss: 0.6264 - acc: 0.6878
Epoch 4/25
18141/18141 [==============================] - 1s 29us/step - loss: 0.6263 - acc: 0.6882
Epoch 5/25
```

```
18141/18141 [==============================] - 0s 27us/step - loss: 0.6242 - acc: 0.6881
Epoch 6/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6241 - acc: 0.6882
Epoch 7/25
18141/18141 [==============================] - 1s 29us/step - loss: 0.6236 - acc: 0.6881
Epoch 8/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6244 - acc: 0.6881
Epoch 9/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6236 - acc: 0.6882
Epoch 10/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6224 - acc: 0.6881
Epoch 11/25
18141/18141 [==============================] - 1s 35us/step - loss: 0.6218 - acc: 0.6882
Epoch 12/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6228 - acc: 0.6882
Epoch 13/25
18141/18141 [==============================] - 1s 30us/step - loss: 0.6223 - acc: 0.6882
Epoch 14/25
18141/18141 [==============================] - 0s 25us/step - loss: 0.6218 - acc: 0.6882
Epoch 15/25
18141/18141 [==============================] - 0s 26us/step - loss: 0.6222 - acc: 0.6882
Epoch 16/25
18141/18141 [==============================] - 0s 26us/step - loss: 0.6221 - acc: 0.6882
Epoch 17/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6220 - acc: 0.6882
Epoch 18/25
18141/18141 [==============================] - 1s 30us/step - loss: 0.6217 - acc: 0.6882
Epoch 19/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6218 - acc: 0.6882
Epoch 20/25
18141/18141 [==============================] - 1s 31us/step - loss: 0.6221 - acc: 0.6882
Epoch 21/25
18141/18141 [==============================] - 0s 26us/step - loss: 0.6224 - acc: 0.6882
Epoch 22/25
18141/18141 [==============================] - 0s 26us/step - loss: 0.6217 - acc: 0.6882
Epoch 23/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6217 - acc: 0.6882
Epoch 24/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6214 - acc: 0.6882
Epoch 25/25
18141/18141 [==============================] - 0s 27us/step - loss: 0.6218 - acc: 0.6882
18141/18141 [==============================] - 1s 33us/step

acc: 68.82%

In [20]: Y_predict_nn

Out[20]: array([0.6498375 , 0.64984405, 0.6498582 , ..., 0.64984334, 0.64988047,
```

```
               0.6498609 ], dtype=float32)

In [21]: Y_predict_cnn

Out[21]: array([0.73876154, 0.7096087 , 0.6279216 , ..., 0.69086987, 0.36187735,
               0.38662952], dtype=float32)

In [22]: Y_predict_lr

Out[22]: array([0.67529549, 0.69298721, 0.62284142, ..., 0.4544102 , 0.3863253 ,
               0.383736  ])

In [23]: Y_predict_rf

Out[23]: array([0.73684211, 0.94736842, 0.68421053, ..., 0.52631579, 0.15789474,
               0.52631579])

In [24]: Y_predict_gbc

Out[24]: array([0.82857143, 0.91071429, 0.63214286, ..., 0.62857143, 0.26785714,
               0.57857143])

In [ ]: Y_predict_dt

In [ ]: 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 [ ]: Y_predict

In [ ]: 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 [ ]: Y_predict_temp

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

In [ ]: 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
```