In [1]:
```
import numpy as np
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
from xgboost import XGBRegressor
warnings.filterwarnings("ignore")
```

```
/usr/local/lib/python2.7/dist-packages/pandas/computation/expressions.py:21: UserWarning: The installed versi
on of numexpr 1.4.2 is not supported in pandas and will be not be used
The minimum supported version is 2.1

  "version is 2.1\n".format(ver=ver), UserWarning)
```

In [2]:
```
def make_submission(answers, name):
    sample_submission = pd.read_csv("sample_submission.tsv")
    sample_submission['y'] = answers
    # In GBM you can get some negative predictions:
    print sample_submission[sample_submission['y'] < 0]
    sample_submission['y'] = sample_submission['y'].map(lambda x: x if x > 0 else 0.0)
    sample_submission.to_csv(name, sep=',', index=False)
```

In [3]:
```
def get_sMAPEError(y, y_pred):
    return 200 * np.sum(np.abs((y - y_pred) / (np.abs(y) + np.abs(y_pred)))) / y.size
```

In [4]:
```
hasher = {}
lb = preprocessing.LabelBinarizer()
back_lb = {}
back_hasher = {}
indexes = {}
```

In [5]:
```
def update_hash(column):
    global hasher
    for item in column:
        if item not in hasher:
            ttt = len(hasher)
            hasher[item] = ttt
            back_hasher[ttt] = item
def changed_column(column):
    global hasher
    new_column = np.array([hasher[item] for item in column])
    return new_column
```

In [6]:
```
def add_hash(column):
    global hasher
    for item in column:
        if item not in hasher:
            hasher[item] = item
```

In [7]:
```
def binarize_data(frame, i_p=3, o=True):
    frame_len = frame["item_id"].values.shape[0]

    binarized = lb.transform(frame["item_id"].values.astype(int))
    frame.drop('item_id', axis=1, inplace=True)
    insert_place = i_p

    for i in binarized.T:
        frame.insert(insert_place, "item_id == " + str(insert_place - i_p), i)
        insert_place += 1
```

In [8]:
```
def read_data(_frac=1.0):
    global hasher
    hasher = {}
    test = pd.read_csv("test.tsv")
    test = test[test["shift"] == 1]

    train = pd.read_csv("train.tsv")
    train = train.sample(frac=_frac, random_state=42)
    data = train[train["shift"] == 1]
    data.drop('Num', axis=1, inplace=True)
    test.drop('Num', axis=1, inplace=True)
    diff = np.setdiff1d(data["item_id"].values, test["item_id"].values)

    for i in diff:
        data.drop(data[data["item_id"] == i].index, inplace=True)

    update_hash(data['item_id'])
    update_hash(test['item_id'])
    data['item_id'] = changed_column(data['item_id'])
    test['item_id'] = changed_column(test['item_id'])

    data.sort(['item_id', 'year', 'week'], inplace=True)

    labels = data['y'] # * 0.62108408518104941
    data.drop('y', axis=1, inplace=True)


    lb.fit(np.unique(data["item_id"].values))

    for i in np.unique(data["item_id"].values):
        back_lb[''.join(lb.transform([i]).astype(str)[0])] = i
    binarize_data(data, o=False)
    binarize_data(test, 2)
    to_drop = ['f' + str(i) for i in np.arange(31, 61)]
    data.drop('shift', axis=1, inplace=True)
    test.drop('shift', axis=1, inplace=True)
    print(to_drop)
    for i in to_drop:
        data.drop(i, axis=1, inplace=True)
        test.drop(i, axis=1, inplace=True)
```

```
    return data, labels, test
```

In [9]:
```
%%time
data, labels, test = read_data()
```

```
['f31', 'f32', 'f33', 'f34', 'f35', 'f36', 'f37', 'f38', 'f39', 'f40', 'f41', 'f42', 'f43', 'f44', 'f45', 'f4
6', 'f47', 'f48', 'f49', 'f50', 'f51', 'f52', 'f53', 'f54', 'f55', 'f56', 'f57', 'f58', 'f59', 'f60']
CPU times: user 6.21 s, sys: 1.13 s, total: 7.34 s
Wall time: 7.34 s
```

In [10]:
```
data.head()
```

Out[10]:

	year	week	item_id == 0	item_id == 1	item_id == 2	item_id == 3	item_id == 4	item_id == 5	item_id == 6	item_id == 7	...	f21	f22	f23	f24	f25	f26
76	2012	52	1	0	0	0	0	0	0	0	...	140980	127850	152370	114950	127800	1262
769	2013	1	1	0	0	0	0	0	0	0	...	127850	152370	114950	127800	126230	1847
1464	2013	2	1	0	0	0	0	0	0	0	...	152370	114950	127800	126230	184760	9868
2155	2013	3	1	0	0	0	0	0	0	0	...	114950	127800	126230	184760	98680	1445
2847	2013	4	1	0	0	0	0	0	0	0	...	127800	126230	184760	98680	144500	1315

5 rows × 259 columns

In [11]:
```
test.head()
```

Out[11]:

	year	week	item_id == 0	item_id == 1	item_id == 2	item_id == 3	item_id == 4	item_id == 5	item_id == 6	item_id == 7	...	f21	f22	f23	f24	f25	f26
678	2015	3	0	0	0	0	0	0	0	0	...	969	1635	895	2140	1182	1020
679	2015	3	0	0	0	0	0	0	0	0	...	21195	18280	18270	15851	16920	18320
680	2015	3	0	0	0	0	0	0	0	0	...	221622	256605	240047	236630	206697	24565
681	2015	3	0	0	0	0	0	0	0	0	...	22450	22093	31175	23355	15358	18930
682	2015	3	0	0	0	0	0	0	0	0	...	60	30	50	20	20	30

5 rows × 259 columns

In [ ]:
```
```

Holdout

In [12]:
```
check_data = data[-1000:]
check_labels = labels[-1000:]
train_data = data[:-1000]
train_labels = labels[:-1000]
print (check_data.shape, check_labels.shape)
print (train_data.shape, train_labels.shape)
```

```
((1000, 259), (1000,))
((21275, 259), (21275,))
```

In [13]:
```
def cross_val_time_dependent(size, k=5):
    """
    [1] - [2]
    [1][2] - [3]
    [1][2][3] - [4]
    [1][2][3][4] - [5]
    """
    parts = [0] + [size / k] * (k - 1) + [size / k + size % k]
    parts = np.cumsum(parts)
    return np.array([(np.arange(parts[pc - 1]), np.arange(parts[pc - 1], parts[pc]))
                     for pc in np.arange(1, k + 1)])[1:]
```

In [14]:
```
def perform_cros_val(estimators, folds, fit_data=train_data,
                     fit_labels=train_labels, o=False):
    f1=open("cros_val", 'w+')
    err = 0
    counter = 1
    err = np.zeros(folds - 1)
    for cv in cross_val_time_dependent(train_data.shape[0], folds):
        train_indices, test_indices = cv
        for estimator in estimators:
            estimator.fit(train_data.iloc[train_indices],
                          train_labels.iloc[train_indices].values)


        mean_predicted = np.array(estimators[0].predict(train_data.iloc[test_indices]))

        for i in np.arange(1, len(estimators)):
            mean_predicted = np.vstack((mean_predicted,
                                        estimators[i].predict(train_data.iloc[test_indices])))

        if len(mean_predicted.shape) > 1:
            err[counter - 1] = get_sMAPEError(train_labels.iloc[test_indices].values,
                                              np.mean(mean_predicted, axis=0))
        else:
            err[counter - 1] = get_sMAPEError(train_labels.iloc[test_indices].values,
                                              mean_predicted)
        if o:
            print >>f1, "did %d folds" % counter

        counter += 1
    print >>f1, (np.sum(err) / (folds - 1), np.max(err), )
    f1.close()
    return (np.sum(err) / (folds - 1), np.max(err), )
```

In [15]:
```
def predict(estimator, t_d, splitted):
    if not splitted:
        return estimator.predict(t_d)
    ans = []
    for i in unique:
        ans += [estimator.predict(t_d)]
    ans = np.array(ans)
    return ans
```

In [16]:
```
def full_fit_predict_mean(estimators, fit_data=data,
                          fit_labels=labels, target=test):
    for estimator in estimators:
            estimator.fit(fit_data, fit_labels.values)

    mean_predicted = np.array(estimators[0].predict(target))
    for i in np.arange(1, len(estimators)):
        mean_predicted = np.vstack((mean_predicted,
                                    estimators[i].predict(target)))
    if len(mean_predicted.shape) > 1:
        return np.mean(mean_predicted, axis=0)
    else:
        return mean_predicted
```

In [17]:
```
def check_real_error(estimators, fit_data, fit_labels, target):
    return get_sMAPEError(full_fit_predict_mean(estimators,
                                                fit_data,
                                                fit_labels,
                                                target),
                          check_labels)
```

In [27]:
```
def perform_cros_val(estimators, folds, fit_data=train_data,
                     fit_labels=train_labels, o=False,
                     filename="cross_val.txt"):
    err = 0
    counter = 1
    f1 = open(filename, "w")
    err = np.zeros(folds - 1)
    for cv in cross_val_time_dependent(train_data.shape[0], folds):
        train_indices, test_indices = cv
        for estimator in estimators:
            estimator.fit(train_data.iloc[train_indices],
                          train_labels.iloc[train_indices].values)
        mean_predicted = np.array(estimators[0].predict(train_data.iloc[test_indices]))

        for i in np.arange(1, len(estimators)):
            mean_predicted = np.vstack((mean_predicted,
                                        estimators[i].predict(train_data.iloc[test_indices])))
        if len(mean_predicted.shape) > 1:
            err[counter - 1] = get_sMAPEError(train_labels.iloc[test_indices].values,
                                              np.mean(mean_predicted, axis=0))
        else:
            err[counter - 1] = get_sMAPEError(train_labels.iloc[test_indices].values,
                                              mean_predicted)
        if o:
            print >>f1, "did %d folds" % counter

        counter += 1
    print >>f1, (np.sum(err) / (folds - 1), np.max(err), )
    f1.close()
    return (np.sum(err) / (folds - 1), np.max(err), )
```

In [19]:
```
def full_fit_predict_mean(estimators, fit_data=train_data, fit_labels=train_labels, target=test):
    for estimator in estimators:
            estimator.fit(fit_data, fit_labels.values)

    mean_predicted = np.array(estimators[0].predict(target))
    for i in np.arange(1, len(estimators)):
        mean_predicted = np.vstack((mean_predicted, estimators[i].predict(target)))
    if len(mean_predicted.shape) > 1:
        return np.mean(mean_predicted, axis=0)
    else:
        return mean_predicted
```

In [20]:
```
def check_real_error(estimators, fit_data, fit_labels, target):
    return get_sMAPEError(full_fit_predict_mean(estimators,
                                                fit_data,
                                                fit_labels,
                                                target),
                          check_labels)
```

In [21]:
```
def append_estimator_answer(frame, estimator, name, fit_data, fit_labels, target):
    predicted = estimator.predict(target)
    new_data = frame
    new_data.insert(frame.shape[1], name + " answer", predicted)
    return new_data
```

In [70]:
```
xgb = XGBRegressor(n_estimators=500, nthread=16, learning_rate=0.1, max_depth=10)
perform_cros_val([xgb], 6, o=True, filename="cross_val_xgb.txt")
```

Out[70]: (22.986785615370053, 24.809570392040563)

In [37]:
```
check_real_error([xgb], train_data, train_labels, target=check_data)
```

Out[37]: 20.511363815313395

In [856]:
```
get_sMAPEError(full_fit_predict_mean([xgb], data, labels, data), labels)
```

Out[856]: 30.098054224586303

In [58]:
```
from sklearn.ensemble import ExtraTreesRegressor
etreg = ExtraTreesRegressor(n_estimators=5000, n_jobs=16)
```

In [33]:
```
perform_cros_val([etreg], 6, o=True, filename="cv_etreg.txt")
```

Out[33]:
```
(23.20944997826027, 26.634495569330543)
```

In [34]:
```
check_real_error([etreg, xgb], train_data, train_labels, target=check_data)
```

Out[34]:
```
18.711812314630716
```

In [41]:
```
from sklearn.ensemble import BaggingRegressor
bag_xgboost = BaggingRegressor(base_estimator=XGBRegressor(n_estimators=5000,
                                                           max_depth=10,
                                                           learning_rate=0.1),
                               n_estimators=10)
perform_cros_val([bag_xgboost], 6, o=True,  filename="cross_val_xgb.txt")
```

Out[41]:
```
(21.8050954642544, 23.978184366412915)
```

In [ ]:
```
predicted = full_fit_predict_mean([etreg], data, labels, test)
```

In [45]:
```
def renew_ans(df):
    d = {}
    for kk, i in enumerate(df.iterrows()):
        year = i[1]['year'].astype(int)
        week = (i[1]['week'] - i[1]['shift']).astype(int)
        item_id = hasher[i[1]['item_id']]
        f30 = (i[1]['f30']).astype(int)
        if week <= 0:
            year -= 1
            week += 52
        d[(year, week, item_id)] = f30 / 0.6211
    return d
```

In [46]:
```
real_test = pd.read_csv("test.tsv")
```

In [47]:
```
d = renew_ans(real_test)
```

In [48]:
```
def build_pred(predicted, df):
    ff = {}
    for k, i in enumerate(df.iterrows()):
        year = i[1]['year'].astype(int)
        week = (i[1]['week']).astype(int)
        item_id = back_lb[''.join(i[1][2:229].astype(int).astype(str))]
        print year, week, item_id
        ff[(year, week, item_id)] = predicted[k]

    return ff
```

In [49]:
```
ff = build_pred(predicted, test)
```

...

In [53]:
```
def build_answer(ff):
    global real_test
    global d
    t = 0
    n = 0
    new_y = np.arange(real_test.shape[0])
    for k, i in enumerate(real_test.iterrows()):
        year = (i[1]['year']).astype(int)
        week = (i[1]['week']).astype(int)
        item_id = hasher[i[1]['item_id']]

        if (year, week, item_id) in d:
            t += 1
            new_y[k] = d[(year, week, item_id)]
        else:
            n+=1
            new_y[k] = ff[(year, week, item_id)]
    print t, n
    return new_y
```

In [54]:
```
pred_no_shift = build_answer(ff)
```

```
1335 681
```

In [55]:
```
pred_no_shift
```

Out[55]:
```
array([  1497,  27034, 297568, ...,  18775,    718,   6331])
```

In [71]:
```
make_submission(pred_no_shift, "etreg_5000.tsv")
```

```
Empty DataFrame
Columns: [Num, y]
Index: []
```

In [ ]:
```

```