Описание решения

В начале, изучив признаки, я увидел, что признак shift не нужен, так как из-за того, что встречаются данные со всеми shift'ами, можно предсказывать ответ для shift = 1, и писать его для остальных (это реализовано в функции add_shift).

Дальше, построив графики признаков, я увидел, что график таргета очень похож на график f60, по сути это он же, сдвинутый на 1 неделю и домноженный на константу. Из-за того, что данные округлены, точно узнать эту константу невозможно, я брал полусумму медиан за 2014 и 2013 года (это реализовано в функции get_k). То есть осталось построить модель для 5 недели (предсказания для остальных я добавлял с помощью функции add_good). Я убрал все признаки кроме f55, ..., f60 и добавил разницу между рядом стоящими - diff_f_i, и значения за прошлый год со сдвигом вперёд на две недели (То есть для 40 недели 2014 года я добавил в качестве признака 37, ..., 42 недели 2013 года). В частности, из-за этого я валидировался только на 2014 году. Генерация фич реализована в функции feature_gen.

Потом я попробовал на этих признаках несколько моделей - линейную регрессию, RF, XGBoost. Для XGBoost'а я искал параметры своим grid search (функция grid_search), но всё равно лучшей моделью оказался RF, параметры для которого я подбирал руками - обучался на первых 40 неделях 2014 года и валидировался на 41-50 неделях 2014 года. Я так же написал свою кросс валидацию с учетом времени (функция my_cross_validation), но мне она почти не пригодилась.

Ещё я сделал преобразование таргета - изначально домножил его на константу из функции get_k, и потом применил функцию log(1 + x), чтобы сгладить выбросы - результат стал лучше.

Вот в общем-то и всё :)

In [1]:
```
from __future__ import print_function
import pandas as pd
import xgboost as xgb
import numpy as np
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

%matplotlib inline
```

```
/Users/ocksumoron/anaconda/lib/python2.7/site-packages/sklearn/cross_validation.py:44: DeprecationWarning: Th
is module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored
classes and functions are moved. Also note that the interface of the new CV iterators are different from that
of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
train = pd.read_csv("train.tsv")
test = pd.read_csv("test.tsv")
sample_submission = pd.read_csv("sample_submission.tsv")
```

In [3]:
```
X = train[train['shift'] == 1].drop(['Num','y','shift'], axis=1)
y = train[train['shift'] == 1]['y']
print(len(X), len(y))
```

```
24385 24385
```

In [4]:
```
def get_k(item_id, year):
    vals = []
    ys = []
    data = train[(train['item_id'] == item_id) & (train['year'] == year) & (train['shift'] == 1)].sort_values('w
    if (data.shape[0] == 0):
        return 1.0
    ys.append(data.iloc[0]['y'])
    vals.append(data.iloc[0]['f60'])

    for i in range(1, data.shape[0]):
        vals.append(data.iloc[i]['f60'])
        ys.append(data.iloc[i]['y'])

    vals = np.array(vals)
    ys = np.array(ys)
    k = np.median(vals[1:] / ys[:-1])
    return k

ks_2013 = {}
ks_2014 = {}

for item_id in test['item_id'].unique():
    ks_2013[item_id] = get_k(item_id, 2013)
    ks_2014[item_id] = get_k(item_id, 2014)

best_k = (np.median(list(ks_2013.values())) + np.median(list(ks_2014.values()))) * 0.5

y *= best_k
```

In [5]:
```
X_13, y_13 = X[X['year'] == 2013], y[X['year'] == 2013]
X_14, y_14 = X[(X['year'] == 2014) & (X['week'] <= 50)], y[(X['year'] == 2014) & (X['week'] <= 50)]
```

In [6]:
```
def feature_gen():
    features = ['f' + str(i) for i in range(55, 61)]
    for i, row in X_14.iterrows():
        prev_vals = X_13[(X_13['item_id'] == row['item_id']) & \
                         (X_13['week'] == row['week'] + 2)]

        if prev_vals.shape[0] == 0:
            for f in ['f' + str(j) for j in range(55, 61)]:
                X_14.set_value(i, 'prev_' + f, 0.0)

            continue


        prev_val = prev_vals.iloc[0]

        for f in ['f' + str(j) for j in range(55, 61)]:
            X_14.set_value(i, 'prev_' + f, prev_val[f])


    for i, row in test.iterrows():
        prev_vals = X_14[(X_14['item_id'] == row['item_id']) & \
                         (X_14['week'] == row['week'] + 2)]

        if prev_vals.shape[0] == 0:
            for f in ['f' + str(j) for j in range(55, 61)]:
                test.set_value(i, 'prev_' + f, 0.0)

            continue

        prev_val = prev_vals.iloc[0]

        for f in ['f' + str(j) for j in range(55, 61)]:
            test.set_value(i, 'prev_' + f, prev_val[f])

    features += ['prev_f' + str(i) for i in range(55, 61)]
    return features
```

In [7]:
```
features = feature_gen()
```

```
/Users/ocksumoron/anaconda/lib/python2.7/site-packages/pandas/core/indexing.py:288: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view
-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  self.obj[key] = _infer_fill_value(value)
/Users/ocksumoron/anaconda/lib/python2.7/site-packages/pandas/core/indexing.py:465: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view
-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  self.obj[item] = s
```

In [8]:
```
X_train, y_train = X_14[X_14['week'] < 40], y_14[X_14['week'] < 40]
X_val, y_val = X_14[X_14['week'] >= 40], y_14[X_14['week'] >= 40]
```

In [9]:
```
def get_smape(y_pred, y_true):
    y_pred = np.array(y_pred)
    y_true = np.array(y_true)
    n = len(y_pred)
    return 100. / n * np.sum(np.abs(y_pred - y_true) / (np.abs(y_pred) + np.abs(y_true)))

def get_smape_d(y_pred, d_y_true):
    y_true = d_y_true.get_label()
    y_pred = np.array(y_pred)
    y_true = np.array(y_true)
    n = len(y_pred)
    return 'smape', 100. / n * np.sum(np.abs(y_pred - y_true) / (np.abs(y_pred) + np.abs(y_true)))
```

In [87]:
```
param = {}
param['max_depth'] = 15
param['booster'] = 'gbtree'
param['objective'] = 'reg:linear'
param['eta'] = 0.05
param['colsample_bytree'] = 1
param['subsample'] = 0.8

numround = 2001
```

In [89]:
```
%%time
#Xdata = xgb.DMatrix(data = X[features], label = y)
Xdatatrain = xgb.DMatrix(data = X_train[features], label = y_train)
Xdatatest = xgb.DMatrix(data = X_val[features], label = y_val)

plst = list(param.items())
watchlist = [(Xdatatrain, 'train'), (Xdatatest, 'eval')]

output = {}

bst = xgb.train(plst, Xdatatrain, numround, feval=get_smape_d, evals=watchlist, evals_result=output, verbose_eva
# ypredxgb_tr = bst.predict(Xdatatrain)
```

```
[0]     train-smape:89.8795     eval-smape:88.767
[100]   train-smape:3.8602      eval-smape:14.3006
[200]   train-smape:4.01799     eval-smape:14.5061
[300]   train-smape:3.51311     eval-smape:14.4481
[400]   train-smape:2.96415     eval-smape:14.5934
[500]   train-smape:2.56326     eval-smape:14.5526
[600]   train-smape:2.0909      eval-smape:14.6037
[700]   train-smape:1.73367     eval-smape:14.6019
[800]   train-smape:1.35807     eval-smape:14.6326
[900]   train-smape:1.00322     eval-smape:14.6459
[1000]  train-smape:0.798811    eval-smape:14.6189
[1100]  train-smape:0.648265    eval-smape:14.6528
[1200]  train-smape:0.518767    eval-smape:14.6299
[1300]  train-smape:0.431089    eval-smape:14.6764
[1400]  train-smape:0.416075    eval-smape:14.7061
[1500]  train-smape:0.365439    eval-smape:14.6592
[1600]  train-smape:0.33548     eval-smape:14.6558
[1700]  train-smape:0.327377    eval-smape:14.6469
[1800]  train-smape:0.323039    eval-smape:14.6212
[1900]  train-smape:0.30874     eval-smape:14.6947
[2000]  train-smape:0.315407    eval-smape:14.6953
CPU times: user 54.8 s, sys: 373 ms, total: 55.2 s
Wall time: 55.8 s
```

In [ ]:
```
def grid_search():
    max_depths = [3, 4, 5, 6, 7, 8, 10, 12, 15]
    learning_rates = [0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.13, 0.15, 0.17, 0.2]
    subsamples = [0.8, 0.9, 1]
    colsample_bytrees = [0.8, 0.9, 1]


    numround = 2001

    best_q = 15.0

    for max_depth in max_depths:

        print("MAX_DEPTH =", max_depth)

        with open("output" + str(max_depth), 'w') as f:

            for learning_rate in learning_rates:

                print("LEARNING_RATE =", learning_rate)

                for subsample in subsamples:

                    print ("SUBSAMPLE =", subsample)

                    for colsample_bytree in colsample_bytrees:

                        print ("COLSAMPLE_BYTREE =", colsample_bytree)

                        param = {}
                        param['max_depth'] = max_depth
                        param['booster'] = 'gbtree'
                        param['objective'] = 'reg:linear'
                        param['eta'] = learning_rate
                        param['colsample_bytree'] = colsample_bytree
                        param['subsample'] = subsample

                        plst = list(param.items())

                        output = {}
```

```
                    bst = xgb.train(plst, Xdatatrain, \
                                    numround, feval=get_smape_d, \
                                    evals=watchlist, evals_result=output, \
                                    verbose_eval=False)

                    f.write("LEARNING_RATE =" + str(learning_rate) + \
                            ", SUBSAMPLE =" + str(subsample) + \
                            ", COLSAMPLE_BYTREE =" + str(colsample_bytree) + "\n")

                    output_smape = output['eval']['smape']

                    for i in range(0, len(output_smape), 100):
                        f.write(str(i) + ": " + str(output_smape[i]) + "\n")
                        if (output_smape[i] < best_q):
                            best_q = output_smape[i]
                            print("BEST = " + str(best_q))
```

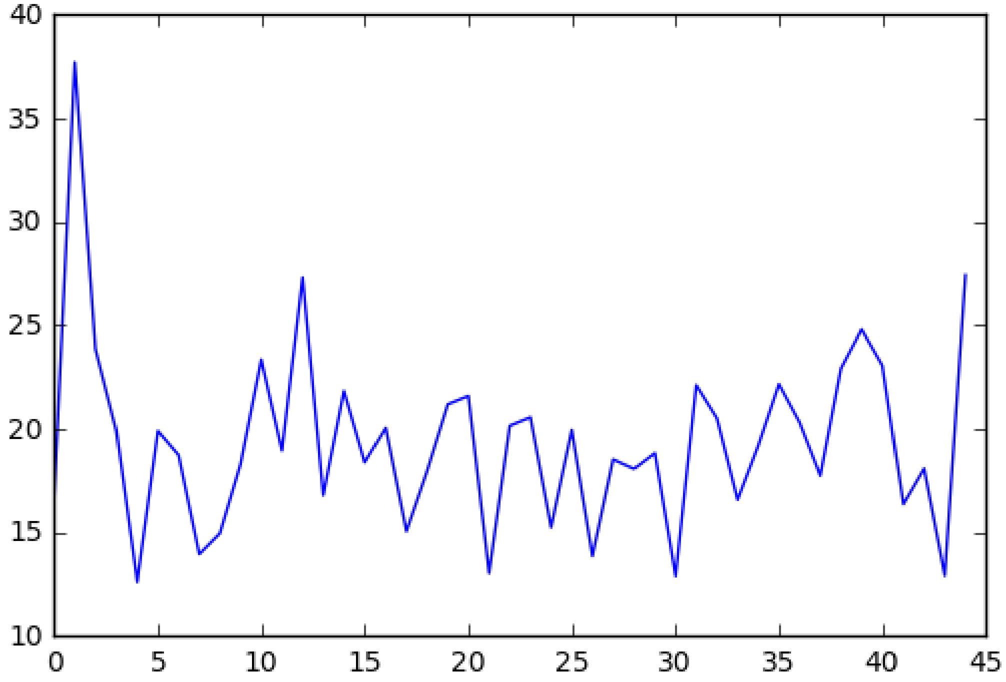
In [128]:

```
%%time

rf_model = RandomForestRegressor(n_estimators=60, min_samples_leaf=10, \
                                 min_samples_split=5, criterion='mae')
rf_model.fit(X_train[features], np.log(1 + y_train))

print (get_smape(np.exp(rf_model.predict(X_val[features])) - 1, y_val))
```

```
13.7083786122
CPU times: user 2min 7s, sys: 2.31 s, total: 2min 9s
Wall time: 2min 25s
```

In [150]:
```
size = 5
train_vals =  (X_14['week'] <= size)
X_train, y_train = X_14[train_vals], y_14[train_vals]

res = []

rf_model.fit(X_train[features], y_train)

for i in range(size + 1, 51):

    test_vals = (X_14['week'] == i)
    X_val, y_val = X_14[test_vals], y_14[test_vals]


    y_pred = rf_model.predict(X_val[features])

    res.append(get_smape(y_pred, y_val))

plt.plot(res)
```

Out[150]:
```
[<matplotlib.lines.Line2D at 0x11a7b86d0>]
```



In [ ]:

In [109]:
```
def my_cross_validation(model, features, f=lambda x: x, inv=lambda x: x):
    cv = []
    for fold in [20, 30, 40]:

        train_vals = X_14['week'] <= fold
        X_train, y_train = X_14[train_vals], y_14[train_vals]

        test_vals = (X_14['week'] > fold) & (X_14['week'] <= fold + 10)
        X_val, y_val = X_14[test_vals], y_14[test_vals]

        model.fit(X_train[features], f(y_train))

        y_pred = model.predict(X_val[features])

        cv.append(get_smape(inv(y_pred), y_val))

    return cv
```

In [239]:
```
%%time
cv = my_cross_validation(rf_model, features)
print(cv)
```

```
[9.4106218557152506, 11.571065672062668, 14.897333727267171]
CPU times: user 2min 20s, sys: 1.2 s, total: 2min 21s
Wall time: 2min 29s
```

In [252]:
```
%%time
cv = my_cross_validation(rf_model, features, np.log, np.exp)
print(cv)
```

```
[9.2360837860543814, 11.081189051954642, 14.063932545790538]
CPU times: user 2min 17s, sys: 2.33 s, total: 2min 20s
Wall time: 2min 38s
```

In [253]:
```
%%time
rf_model = RandomForestRegressor(n_estimators=1000)
cv = my_cross_validation(rf_model, features, np.log, np.exp)
print(cv)
```

```
[9.2268165148619055, 11.062919227203171, 14.062203114591687]
CPU times: user 3min 11s, sys: 3.62 s, total: 3min 14s
Wall time: 3min 39s
```

In [268]:
```
%%time
rf_model = RandomForestRegressor(n_estimators=1000)
cv = my_cross_validation(rf_model, features, lambda x: np.log(x + 1), lambda x: np.exp(x) - 1)
print(cv)
```

```
[9.2135936606397291, 11.067382395644351, 14.042284714545389]
CPU times: user 3min 1s, sys: 2.55 s, total: 3min 3s
Wall time: 3min 15s
```

In [223]:
```
%%time

rf_model = RandomForestRegressor(n_estimators=700, oob_score = True)
rf_model.fit(X_14[features], y_14)
```

```
CPU times: user 1min 24s, sys: 1.11 s, total: 1min 25s
Wall time: 1min 36s
```

In [119]:
```
def add_shift(model, features, is_df=False):
    test_mod = test[test['shift'] == 1].drop(['Num', 'shift'], axis=1)
    if (is_df):
        to_test = xgb.DMatrix(data = test_mod[features])
        predicted = model.predict(to_test)
    else:
        predicted = model.predict(test_mod[features])
    all_predicted_dict = {}
    all_predicted = []
    i = 0
    for _, row in test_mod.iterrows():
        all_predicted_dict[(row.item_id, row.week)] = predicted[i]
        i += 1
    for _, row in test.iterrows():
        all_predicted.append(all_predicted_dict[(row.item_id, row.week)])
    return all_predicted
```

In [129]:
```
# to_fit = X_14['week'] >= 40
rf_model.fit(X_14[features], np.log(1 + y_14))
```

Out[129]:
```
RandomForestRegressor(bootstrap=True, criterion='mae', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=10,
           min_samples_split=5, min_weight_fraction_leaf=0.0,
           n_estimators=60, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

In [130]:
```
sample_submission['y'] = np.ceil((np.exp(add_shift(rf_model, features)) - 1) / best_k)
```

In [134]:
```
def add_good(submission):
    ans = {}
    for item_id in test['item_id'].unique():
        if test[(test['item_id'] == item_id) & (test['week'] == 5) & (test['shift'] == 1)].shape[0] == 1:
            ans[(item_id, 4)] = \
                np.array(test[(test['item_id'] == item_id) & \
                              (test['week'] == 5) & \
                              (test['shift'] == 1)]['f60'])[0] / best_k
            ans[(item_id, 3)] = \
                np.array(test[(test['item_id'] == item_id) & \
                              (test['week'] == 5) & \
                              (test['shift'] == 1)]['f59'])[0] / best_k
        elif test[(test['item_id'] == item_id) & (test['week'] == 4) & (test['shift'] == 1)].shape[0] == 1:
            ans[(item_id, 3)] = \
                np.array(test[(test['item_id'] == item_id) & \
                              (test['week'] == 4) & \
                              (test['shift'] == 1)]['f60'])[0] / best_k

    for i, row in test.iterrows():
        if (row.item_id, row.week) in ans:
            submission.set_value(i, 'y', int(ans[(row.item_id, row.week)]))
```

In [135]:
```
add_good(sample_submission)
```

In [136]:
```
sample_submission
```

Out[136]:

	Num	y
0	348622	1497.0
1	348623	27033.0
2	348624	297556.0
3	348625	29470.0
4	348626	16.0
5	348627	146240.0
6	348628	67735.0
7	348629	111940.0
8	348630	102629.0
9	348631	3541.0
10	348632	1851.0
11	348633	54996.0
12	348634	278975.0
13	348635	21963.0
14	348636	1948.0
15	348637	178788.0
16	348638	121617.0
17	348639	509835.0
18	348640	510738.0
19	348641	2336.0
20	348642	113224.0
21	348643	83467.0

	Num	y
22	348644	1215.0
23	348645	24231.0
24	348646	10949.0
25	348647	150510.0
26	348648	7489.0
27	348649	18666.0
28	348650	5530.0
29	348651	23602.0
...	...	...
1986	351499	52.0
1987	351500	167546.0
1988	351501	14767.0
1989	351502	17899.0
1990	351503	122.0
1991	351504	395.0
1992	351505	17569.0
1993	351506	486.0
1994	351507	65.0
1995	351508	109030.0
1996	351509	537817.0
1997	351510	175326.0
1998	351511	511192.0
1999	351512	27659.0

	Num	y
2000	351513	998.0
2001	351514	30274.0
2002	351515	492639.0
2003	351516	113856.0
2004	351517	152980.0
2005	351518	495497.0
2006	351519	438.0
2007	351520	73.0
2008	351521	163.0
2009	351522	277.0
2010	351523	5051.0
2011	351524	46891.0
2012	351525	19913.0
2013	351526	16924.0
2014	351527	329.0
2015	351528	4841.0

2016 rows × 2 columns

In [137]:
```
sample_submission.to_csv('FINAL_overfit_submission.tsv', sep=',', index=False)
```

In [121]:
```
X_13.head()
```

Out[121]:

	year	week	item_id	f1	f2	f3	f4	f5	f6	f7	...	f51	f52	f53	f54	f55
696	2013	1	20442092	270.0	980.0	6140.0	8292.0	780.0	426.0	120.0	...	370.0	151.0	220.0	220.0	450.0
697	2013	1	20442091	670.0	2830.0	11530.0	10397.0	260.0	960.0	760.0	...	460.0	400.0	350.0	500.0	480.0
698	2013	1	20442076	38056.0	40185.0	45733.0	59710.0	39982.0	45846.0	43680.0	...	41765.0	52590.0	31452.0	44420.0	41865.
699	2013	1	20441997	18817.0	20110.0	26368.0	31412.0	23182.0	24565.0	27075.0	...	25230.0	27850.0	21390.0	27090.0	23170.
700	2013	1	20441990	47480.0	47619.0	89708.0	166338.0	37620.0	85607.0	92461.0	...	44290.0	46412.0	29320.0	21140.0	28406.

5 rows × 63 columns

In [151]:
```
X_14.iloc[0]['prev_f59']
```

Out[151]: 869.38824759426757

In [148]:
```
y_14.iloc[0]
```

Out[148]: 941

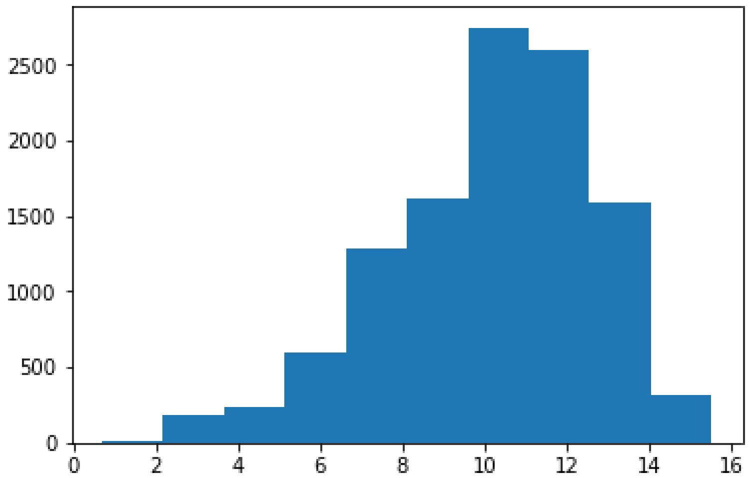
In [145]:
```
X_14.iloc[0]
```

Out[145]:
```
year        2.014000e+03
week        1.000000e+00
item_id     2.044232e+07
f1          8.291388e+02
f2          9.418373e+02
f3          1.231633e+03
f4          9.498872e+02
f5          7.405900e+02
f6          7.405900e+02
f7          8.130390e+02
f8          9.981865e+02
f9          1.078685e+03
f10         9.015878e+02
f11         9.981865e+02
f12         1.135035e+03
f13         1.215534e+03
f14         5.554425e+02
f15         1.038436e+03
f16         1.135035e+03
f17         9.096377e+02
f18         1.263833e+03
f19         4.668937e+02
f20         9.981865e+02
f21         7.969392e+02
f22         1.529479e+03
f23         3.300455e+02
f24         1.416781e+03
f25         1.424831e+03
f26         1.585829e+03
f27         6.761909e+02
                ...
f37         8.130390e+02
f38         9.981865e+02
f39         1.078685e+03
f40         9.015878e+02
f41         9.981865e+02
f42         1.135035e+03
f43         1.215534e+03
f44         5.554425e+02
```

```
f45          1.038436e+03
f46          1.135035e+03
f47          9.096377e+02
f48          1.263833e+03
f49          4.668937e+02
f50          9.981865e+02
f51          7.969392e+02
f52          1.529479e+03
f53          3.300455e+02
f54          1.416781e+03
f55          1.424831e+03
f56          1.585829e+03
f57          6.761909e+02
f58          7.888893e+02
f59          9.418373e+02
f60          1.722677e+03
prev_f55     6.037418e+02
prev_f56     9.418373e+02
prev_f57     8.130390e+02
prev_f58     1.964173e+03
prev_f59     8.693882e+02
prev_f60     4.346941e+02
Name: 36242, dtype: float64
```

In [267]:
```
plt.hist(np.log(1 + y_14))
```

Out[267]:
```
(array([    6.,   185.,   231.,   597.,  1290.,  1615.,  2747.,  2591.,
         1592.,   318.]),
 array([  0.69314718,   2.17711137,   3.66107556,   5.14503975,
          6.62900394,   8.11296813,   9.59693233,  11.08089652,
         12.56486071,  14.0488249 ,  15.53278909]),
 <a list of 10 Patch objects>)
```



In [248]:
```
np.log(y_14)
```

Out[248]:
```
36242     6.846943
36243     6.453625
36244     7.012115
36245     9.617071
36246     8.203578
36247    10.480269
36248     9.473320
36249     3.465736
36250    10.589560
36251     2.772589
36252    10.458521
36253    12.151854
36254    14.079077
36255    13.267497
36256    14.618260
36257    12.922018
36258    10.170227
36259    14.506457
36260    10.934196
36261    13.046626
36262    11.350301
36263     8.860925
36264     8.511980
36265    11.460526
36266    11.470133
36267    10.623058
36268    11.393894
36269     9.069928
36270    12.740130
36271    10.656365
           ...
69285     3.465736
69286    12.526441
69287     9.868793
69288     5.451038
69289     2.772589
69290    10.053845
69291     5.996452
69292     4.158883
```

```
69293     6.669498
69294     6.073045
69295    11.727650
69296    13.515954
69297    12.663596
69298    13.332060
69299    10.198766
69300     7.075809
69301    10.485228
69302    11.822049
69303    13.258552
69304     4.718499
69305     6.244167
69306     6.011267
69307     8.645410
69308    10.731493
69309     9.862613
69310     9.798738
69311     6.467699
69312     8.462737
69313    12.327976
69314    13.289245
Name: y, dtype: float64
```

In [229]:
```
test.head()
```

Out[229]:

	Num	year	week	shift	item_id	f1	f2	f3	f4	f5	...	f57
0	348622	2015	3	3	20447918	1545.579107	1320.182154	1816.055451	2899.570804	1682.427257	...	1902.994275
1	348623	2015	3	3	20447902	14628.262255	20261.576104	18667.697650	15592.639218	20796.088878	...	27240.831758
2	348624	2015	3	3	20447732	185287.565279	237128.864488	283460.818172	230395.935504	325326.692228	...	332777.671506
3	348625	2015	3	3	20443951	33648.545138	39315.668530	43578.890900	32940.154714	41183.243284	...	24726.045753
4	348626	2015	3	3	20443944	7132.203587	9440.912378	5329.027962	2983.289672	4565.898278	...	32.199565

5 rows × 71 columns