Stock category prediction using XGBoost

In [1]:
```
import pickle
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

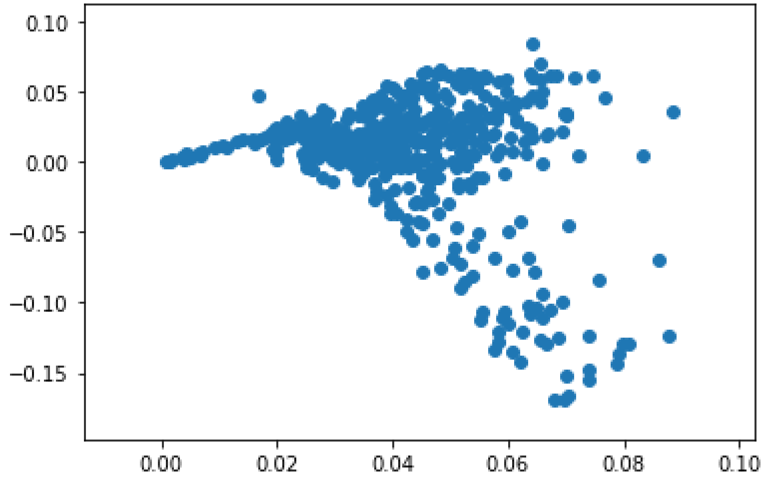
Read PCA parameters

In [2]:
```
D=pickle.load(open('data/PCA_true.pickle','rb'))
col=D['columns']
eigvec=D['eigvec']
eigval=D['eigval']
```

Sanity check

The following scatterplot should be very similar to the scatter plot you produced in notebook 3 for eigvec 1, eigvec 2 (indexing starts with 1)

In [3]:
```
scatter(eigvec[:,0],eigvec[:,1])
```
Out[3]:
```
<matplotlib.collections.PathCollection at 0x1a1884af10>
```



compute features

The features that we use are the coefficients of the top 20 eigenvectors.

Those can be read directly from the eigenvectors matrix.

In [4]:
```
#Taking the top 20 features(Eigen vectors)
features=eigvec[:,:20]
features.shape
```

Out[4]:
```
(481, 20)
```

Compute labels (sectors)

In [5]:
```
TickerInfo=pd.read_csv('data/tickerInfo.tsv',sep='\t')
print(TickerInfo.shape)
TickerInfo.head()
```

```
(505, 5)
```

Out[5]:

	Unnamed: 0	Ticker	Name	Sector	SECTOR_ID
0	0	MMM	3M|3M Company	Industrials	INDS
1	1	ABT	Abbott Laboratories	Health Care	HC
2	2	ABBV	AbbVie Inc.	Health Care	HC
3	3	ACN	Accenture plc	Information Technology	IT
4	4	ATVI	Activision Blizzard	Information Technology	IT

Creating necessary dictionaries

The Sectors dictionary below has the Sector name mapped to the sector ID. Using the Sectors dictionary below, create a dictionary mapping of the sector ID to indices incrementally as follows:

```
{
    'CD': 0,
    'CS': 1,
    'EN': 2,
    'FIN': 3,
    'HC': 4,
    'INDS': 5,
    'IT': 6,
    'MAT': 7,
    'RE': 8,
    'TS': 9,
    'UTIL': 10
}
```

In addition to this, you will need to create one more dictionary mapping index number to the sector name:

```
{
    0: 'Consumer Discretionary',
    1: 'Consumer Staples',
    2: 'Energy',
    3: 'Financials',
    4: 'Health Care',
    5: 'Industrials',
    6: 'Information Technology',
    7: 'Materials',
    8: 'Real Estate',
    9: 'Telecommunication Services',
    10: 'Utilities'
}
```

Input
A dictionary **Sectors** as given in the following cell.

Output
Return two dictionaries **sector2number** and **number2sectorName** as mentioned in the description.

In [6]:
```python
Sectors={'Consumer Discretionary':'CD',
 'Consumer Staples':'CS',
 'Energy':'EN',
 'Financials':'FIN',
 'Health Care':'HC',
 'Industrials':'INDS',
 'Information Technology':'IT',
 'Materials':'MAT',
 'Real Estate':'RE',
 'Telecommunication Services':'TS',
 'Utilities':'UTIL'}
```

In [7]:
```python
#(3 points)

def get_sector_dicts(Sectors):

    ### Your code here

    # import libraries
    from collections import defaultdict

    # initialize dictionaries
    sector2number, number2sectorName = {}, {}

    # loop over every sector
    for idx,(k,v) in enumerate(Sectors.items()):

        # add data to dictionaries
        sector2number[v] = idx
        number2sectorName[idx] = k

    return sector2number, number2sectorName
```

In [8]:
```python
sector2number, number2sectorName = get_sector_dicts(Sectors)
```

In [9]:
```
labels=[]
feature_vectors=[]
feature_vectors_test=[]
test_nos = []
for i in range(len(col)):
    c=col[i]
    if 'train' in c:
        ticker=c[6:-2]
        answer=list(TickerInfo[TickerInfo.Ticker==ticker]['SECTOR_ID'])
        if len(answer)==1:
            sector_no=sector2number[answer[0]]
            labels.append(sector_no)
            feature_vectors.append(features[i,:])
        else:
            print('error: could not find sector for ticker:',ticker)
    if 'test' in c:
        test_nos.append(c[5:-2])
        feature_vectors_test.append(features[i,:])
```

In [10]:
```
#verify lengths
assert len(labels) == 392
assert len(feature_vectors) == 392
assert len(test_nos) == 89
assert len(feature_vectors_test) == 89
```

Placing the data into numpy arrays as expected by xgboost

In [11]:
```
X=np.array(feature_vectors)
y=np.array(labels)
X_test = np.array(feature_vectors_test)
y_test = np.array(test_nos)
X.shape, y.shape, X_test.shape, y_test.shape
```

Out[11]:
```
((392, 20), (392,), (89, 20), (89,))
```

In [12]:
```
#Splitting between train and validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.30, random_state=6)
X_train.shape, X_valid.shape
```

Out[12]:
```
((274, 20), (118, 20))
```

In [13]:
```
#Parameters
param = {}
param['max_depth']= 3   # depth of tree
param['eta'] = 0.3      # shrinkage parameter
param['silent'] = 1     # not silent
param['objective'] = 'multi:softmax'
param['nthread'] = 7 # Number of threads used
param['num_class']=11

num_round = 100
```

Generating scores using XGBoost

The function get_margin_scores is used to predict the sector for each of the given samples.

Input

1. **Training set** (X_train)
2. **Validation set** (X_valid)
3. **Training labels** (y_train)
4. **XGBoost Parameter List** (param)

Output
Return the following:

1. **y_pred_valid**: The raw output scores for the validation set

Note:

1. Round all raw scores to **three** decimal places
2. Remember to use **verbose_eval = False** while training and **ntree_limit=bst.best_ntree_limit** and **output_margin=True** while predicting
3. Remember to provide the **num_round** parameter while training and do not change it. We have currently set it to 100 (Refer previous cell). Not providing the parameter or changing it could produce different results.

In [14]:
```
#(3 points)

def get_margin_scores(X_train, X_valid, y_train, param):

    ### Your code here

    # define dtrain and dvalid
    dtrain = xgb.DMatrix(np.round(X_train, 3), label=y_train)
    dvalid = xgb.DMatrix(np.round(X_valid, 3))

    # define eval list
    evallist = [(dtrain, 'train')]

    # train model
    num_round = 100
    bst = xgb.train(param, dtrain, num_round, evallist, verbose_eval = False)

    # predict
    y_pred_valid = np.array(bst.predict(dvalid, ntree_limit=bst.best_ntree_limit, output_margin=True))


    return y_pred_valid
```

In [15]:
```
y_pred_valid = get_margin_scores(X_train, X_valid, y_train, param)
```

```
[23:35:28] WARNING: /Users/travis/build/dmlc/xgboost/src/learner.cc:480:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

In [16]:
```
assert y_pred_valid.shape == (118, 11), "Incorrect shape"
assert type(y_pred_valid) == numpy.ndarray, "Incorrect type"
```

In [17]:
```
#(1 points)
# Hidden Tests
```

In [18]:
```
#(2 points)
# Hidden Tests
```

In [19]:
```
#(5 points)
# Hidden Tests
```

Computing Top1 and Top5 predictions

Using the margin scores generated, calculate top1 and top5 predictions for the given data:

top1: Find the most probable prediction for each example in the validation set

top5: Find the top 5 most probable predictions in descending order for each example in the validation set (most probable to fifth most probabale)

Input

1. **Validation Output Scores** (y_pred_valid)

Output

1. **predictions_valid**: The most probable prediction for each example in the the validation set
2. **predictions_topn**: The top 5 predictions for each example in the validation set

Sample input

1. **y_pred_valid**: [[-0.3, 1.2, 0.3, 0.5, -0.4, 0.0, 0.01, 1.0, -1.3, 0.2, -1.2], [0.4, -0.5, 1.3, -0.2, 0.6, -2.2, -0.8, 1.9, 0.9, -0.2, -1.7]]

Sample output

1. **predictions_valid**: [1, 7]
2. **predictions_top5**: [[1, 7, 3, 2, 9], [7, 2, 8, 4, 0]]

In [20]:
```
def get_predictions(y_pred_valid):

    ### Your code here

    # most probbable prediction
    predictions_valid = np.apply_along_axis(lambda x: numpy.argsort(x)[::-1][0], 1, y_pred_valid)

    # top 5 predictions
    predictions_topn = np.apply_along_axis(lambda x: numpy.argsort(x)[::-1][:5], 1, y_pred_valid)

    return predictions_valid, predictions_topn
```

In [21]:
```
predictions_valid, predictions_top5 = get_predictions(y_pred_valid)
```

In [22]:
```
assert predictions_valid.shape == (118,), "Incorrect shape"
assert predictions_top5.shape == (118, 5), "Incorrect shape"
assert type(predictions_valid) == numpy.ndarray, "Incorrect type"
assert type(predictions_top5) == numpy.ndarray, "Incorrect type"
```

In [23]:
```
#(3 points)
# Hidden Tests Here
```

In [24]:
```
#(3 points)
# Hidden Tests Here
```

In [25]:
```
acc = 0
for i in range(5):
    acc += sum(predictions_top5[:, i]==y_valid)
    print("Top ", i+1, ": \t", acc/len(y_valid))
```

```
Top  1 :         0.6864406779661016
Top  2 :         0.8305084745762712
Top  3 :         0.9322033898305084
Top  4 :         0.940677966101695
Top  5 :         0.9576271186440678
```

Generating the confusion matrix

What is a confusion matrix? This is a useful link that explains this: https://en.wikipedia.org/wiki/Confusion_matrix#Example (https://en.wikipedia.org/wiki/Confusion_matrix#Example)

We will now be using the top 2 values of the **predictions_top5** matrix generated, to produce the confusion matrix. We will be creating a confusion matrix by comparing the most probable prediction against the second most probable prediction. We are doing this to analyse the scenarios where the second most probable prediction is in fact the correct prediction. So remember to take **only the top2** from the **predictions_top5** matrix. For generating this matrix we do not need more than that.

An example with 4 classes (0, 1, 2, 3):

If 2 is getting confused with 1, i.e., if 1 is the top prediction and 2 is the second-top prediction, then your confusion matrix should add 1 to the cell (1, 2).

Different example scenarios:

Scenario 1:

Sample Input

y_label = 3
top2_predictions = [2, 3]

Output
confusion_matrix:
[0 0 0 0]
[0 0 0 0]
[0 0 0 1]
[0 0 0 0]

Scenario 2:

Sample Input

Say we have 7 sample prediction values
y_label = [3, 2, 3, 1, 0, 2, 2]
top2_predictions = [[2, 3], [0, 3], [2, 3], [1, 2], [3, 0], [2, 0], [3, 2]]

Output

confusion_matrix:
[0 0 0 0]
[0 0 0 0]
[0 0 0 2]
[1 0 1 0]

Explanation:

The first example has top two predictions: (2, 3) and the label 3. So we add 1 to the position (2,3).

The second example has 7 sample predictions:

1. In two scenarios 2 is predicted in place of 3 and the cell (2,3) is incremented twice.
2. In two other scenarios we have a case where the second prediction is right. The corresponding cells {(3,0) and (3,2)} are incremented once each.
3. In two scenarios, the first element is the correct prediction. No cell is incremented then.
4. In one scenario, neither of the top 2 predictions is correct. No cell is incremented then.

Note: y_label is the same as y_valid here.

In [26]:
```
def get_confusion_matrix(predictions_top5, y_valid):

    ### Your code here

    # find matrix dimensions and initialize it
    n = len(set(y_valid))
    confusion_matrix = np.zeros((n,n), int64)

    # loop over every label
    for label, predictions in zip(y_valid, predictions_top5[:,:2]): # only take top 2

        # case when no increment is made
        if (label == predictions[0]) | (label not in predictions):
            continue

        # case when the second prediction is right, increment by one
        if (label == predictions[1]):
            confusion_matrix[predictions[0], predictions[1]]+=1

    return confusion_matrix
```

In [27]:
```
confusion_matrix = get_confusion_matrix(predictions_top5, y_valid)
```

In [28]:
```
assert confusion_matrix.shape == (11, 11), "Incorrect shape"
assert type(confusion_matrix) == numpy.ndarray, "Incorrect type"
assert type(confusion_matrix[0][0]) == numpy.int64, "Incorrect type"
```

In [29]:
```
#(4 points)
# Hidden Tests Here
```

In [30]:
```
for i in range(confusion_matrix.shape[0]):
    print("%30s" % number2sectorName[i], "\t", confusion_matrix[i, :])
```

```
        Consumer Discretionary 	 [0 1 0 0 0 1 1 0 0 0 0]
              Consumer Staples 	 [1 0 0 0 0 0 0 0 0 0 0]
                        Energy 	 [0 0 0 0 0 0 0 0 0 0 0]
                    Financials 	 [0 0 0 0 0 0 1 0 0 0 0]
                   Health Care 	 [0 0 0 0 0 1 2 0 0 0 0]
                   Industrials 	 [2 0 0 1 0 0 0 2 0 0 0]
        Information Technology 	 [2 0 0 0 0 0 0 0 0 0 0]
                     Materials 	 [0 0 0 0 0 1 0 0 0 0 0]
                   Real Estate 	 [1 0 0 0 0 0 0 0 0 0 0]
    Telecommunication Services 	 [0 0 0 0 0 0 0 0 0 0 0]
                     Utilities 	 [0 0 0 0 0 0 0 0 0 0 0]
```

Interpretation of confusion matrix

Based on the confusion matrix generated, answer the following questions.

Categories:

```
0) Consumer Discretionary
1) Consumer Staples
2) Energy
3) Financials
4) Health Care
5) Industrials
6) Information Technology
7) Materials
8) Real Estate
9) Telecommunication Services
10) Utilities
```

Some standard instructions while answering the questions:

1. Each question has two parts: **ans** and **num_scen**

A. **ans**: A list with numbers corresponding to the different categories. For example, if the answer is Consumer Discretionary and Consumer Staples, the **ans** should be a **list** containing **[0, 1]**.
B. **num_scen**: Number of scenarios where the condition in the given question is applicable. For example, if the condition given in the question happens 5 times, the **num_scen** should return **5** as an integer.
2. Type checks have been provided to validate the type of your answer
3. Remember that just the answer for the given matrix should be enough. You do not have to write a generic function that answers the question for all confusion matrices.

Question 1

Which two sectors are most often confused with each other and how many times?

```
0) Consumer Discretionary
1) Consumer Staples
2) Energy
3) Financials
4) Health Care
5) Industrials
6) Information Technology
7) Materials
8) Real Estate
9) Telecommunication Services
10) Utilities
```

Output format for this question: The **ans1** must be list of tuples. If say sector 0 and sector 1 are confused with each other, then your answer must be [(0,1)]. If there are multiple pairs, then it must be a list of all the tuples. For example, **[(0,1),(2,3)]**.

The format for **num_scen1** is the same as mentioned in 1.9.

In [31]:
```
def question_1():

    ### Your code here

    # find the max number of confusions
    num_scen1 = int(np.max(confusion_matrix))

    # find the two sectors that are most often confused
    ans1 = [(int(i[0]), int(i[1])) for i in np.argwhere(confusion_matrix==num_scen1)]

    return ans1, num_scen1
```

In [32]:
```
ans1, num_scen1 = question_1()
assert type(ans1) == list, "Incorrect type"
assert type(ans1[0]) == tuple, "Incorrect type"
assert type(ans1[0][0]) == int, "Incorrect type"
assert type(num_scen1) == int, "Incorrect type"
```

In [33]:
```
#(2 points)
# Hidden tests here
```

In [34]:
```
#(2 points)
# Hidden tests here
```

Question 2

Which sector(s) has/have the most number of scenarios where the prediction with the second highest score is actually correct? In other words, the sector receives the second highest score and but is correct. How many scenarios?

```
0) Consumer Discretionary
1) Consumer Staples
2) Energy
3) Financials
4) Health Care
5) Industrials
6) Information Technology
7) Materials
8) Real Estate
9) Telecommunication Services
10) Utilities
```

Note: Output format for this question is the same as the output format mentioned in Section 1.9.

In [35]:
```
def question_2():

    ### Your code here

    # take the sum along the columns (actual label)
    sum_actual_label = confusion_matrix.sum(axis=0)

    # find the max number of scenarios where the second highest score is correct
    num_scen2 = int(max(sum_actual_label))

    # find the index labels (sectors)
    ans2 = [int(i) for i in np.argwhere(sum_actual_label == num_scen2)]

    return ans2, num_scen2
```

In [36]:
```
ans2, num_scen2 = question_2()
assert type(ans2) == list, "Incorrect type"
assert type(ans2[0]) == int, "Incorrect type"
assert type(num_scen2) == int, "Incorrect type"
```

In [37]:
```
#(1 point)
# Hidden tests here
```

In [38]:
```
#(1 point)
# Hidden tests here
```

Question 3

Which sector(s) most often identified incorrectly? In other words, the sector recieves the highest score even though the sector with the second highest score is the correct sector. How many times does this happen for each of these sectors?

```
0) Consumer Discretionary
1) Consumer Staples
2) Energy
3) Financials
4) Health Care
5) Industrials
6) Information Technology
7) Materials
8) Real Estate
9) Telecommunication Services
10) Utilities
```

Note: Output format for this question is the same as the output format mentioned in Section 1.9.

In [39]:
```
def question_3():

    ### Your code here

    # take the sum along the rows (predicted labels)
    sum_pred_label = confusion_matrix.sum(axis=1)

    # find the max number of scenarios where the sector with the highest score is incorrect
    num_scen3 = int(max(sum_pred_label))

    # find the index labels (sectors)
    ans3 = [int(i) for i in np.argwhere(sum_pred_label == num_scen3)]

    return ans3, num_scen3
```

In [40]:
```
ans3, num_scen3 = question_3()
assert type(ans3) == list, "Incorrect type"
assert type(ans3[0]) == int, "Incorrect type"
assert type(num_scen3) == int, "Incorrect type"
```

In [41]:
```
#(1 point)
# Hidden tests here
```

In [42]:
```
#(1 point)
# Hidden tests here
```

Note for question 4 and 5

Note: The next set of questions might require you to generate a different confusion matrix. Feel free to change the original function or use the box below to write a new function for the same.

In [43]:
```
confusion_matrix2 = np.zeros((11,11), dtype=int)
confusion_matrix3 = np.zeros((11), dtype=int)
i=0
for entry in predictions_top5[:, :1]:
    if entry[0] != y_valid[i]:
        confusion_matrix2[entry[0]][y_valid[i]] += 1
        confusion_matrix3[y_valid[i]] += 1
    i += 1
for i in range(confusion_matrix2.shape[0]):
    print("%30s" % number2sectorName[i], "\t", confusion_matrix2[i, :])
```

```
        Consumer Discretionary 	 [0 2 0 0 0 3 1 0 0 0 0]
              Consumer Staples 	 [3 0 0 0 0 0 1 0 0 1 0]
                        Energy 	 [0 0 0 0 0 0 0 0 0 0 1]
                    Financials 	 [1 0 0 0 0 0 2 0 0 0 0]
                   Health Care 	 [0 1 0 0 0 1 4 0 0 0 0]
                   Industrials 	 [2 0 0 1 0 0 1 3 0 0 0]
        Information Technology 	 [2 0 0 0 0 1 0 0 0 0 0]
                     Materials 	 [0 0 0 0 0 3 0 0 0 0 0]
                   Real Estate 	 [1 0 0 0 1 1 0 0 0 0 0]
    Telecommunication Services 	 [0 0 0 0 0 0 0 0 0 0 0]
                     Utilities 	 [0 0 0 0 0 0 0 0 0 0 0]
```

Question 4

Which category/categories is/are never identified? In other words, which category/categories are never predicted as the top prediction?

```
0) Consumer Discretionary
1) Consumer Staples
2) Energy
3) Financials
4) Health Care
5) Industrials
6) Information Technology
7) Materials
8) Real Estate
9) Telecommunication Services
10) Utilities
```

Note:

1. This question does not have a second part, so your function has to return just one answer.
2. Output format for this question is the same as the output format mentioned in Section 1.9.

In [44]:
```python
def question_4():

    ### Your code here

    # take the sum along the rows (predicted labels)
    sum_pred_label = confusion_matrix2.sum(axis=1)

    # find the index labels where the sum is zero (never predicted as the top prediction)
    ans4 = [int(i) for i in np.argwhere(sum_pred_label == 0)]

    return ans4
```

In [45]:
```python
ans4 = question_4()
assert type(ans4) == list, "Incorrect type"
assert type(ans4[0]) == int, "Incorrect type"
```

In [46]:
```python
#(2 points)
# Hidden tests here
```

Question 5

Which sector(s) is/are most often missed while classifying? In other words, find the sector for which there is the largest number of examples such that the correct label does not appear as the top prediction.

```
0) Consumer Discretionary
1) Consumer Staples
2) Energy
3) Financials
4) Health Care
5) Industrials
6) Information Technology
7) Materials
8) Real Estate
9) Telecommunication Services
10) Utilities
```

Note: Output format for this question is the same as the output format mentioned in Section 1.9.

In [47]:
```
def question_5():

    ### Your code here

    # take the sum along the columns (actual label)
    sum_actual_label = confusion_matrix2.sum(axis=0)

    # find the max number of times the sector is missed while classifying
    # it does not appear as the top first prediction
    num_scen5 = int(max(sum_actual_label))

    # find the index labels (sectors)
    ans5 = [int(i) for i in np.argwhere(sum_actual_label == num_scen5)]

    return ans5, num_scen5
```

In [48]:
```
ans5, num_scen5 = question_5()
assert type(ans5) == list, "Incorrect type"
assert type(num_scen5) == int, "Incorrect type"
```

In [49]:
```
#(2 points)
# Hidden tests here
```

In [50]:
```
#(2 points)
# Hidden tests here
```

Test set

List the pair of top two sectors for each test ticker. Based on your validation results, estimate:

1. ** Accuracy1:** What is the frequency in which the correct sector is the first element in the pair.
2. ** Accuracy2:** What is the frequency in which the correct sector is in the pair.

Generating test scores

The function get_margin_scores_test is used to predict the sector for each of the given test samples. Split the input data into train and validation in different ways and average the prediction scores over a number of iterations. You can experiment with the number, but you will need to ensure you keep to the time limit. (This should not take you more than a minute)

We estimate the predictions on the validation and the test set, compute the accuracy on the validation set and report the test predictions with the validation accuracy. In this scenario, we report the average top1 and top2 accuracy.

Input

1. **Input data** (X)
2. **Test data** (X_test)
3. **Input labels** (y)
4. **XGBoost Parameter List** (param)

Output
Return the following:

1. **y_pred_test**: The raw output scores for the test set
2. **top1_acc**: Top 1 accuracy on the validation set
3. **top2_acc**: Top 2 accuracy on the validation set

Note:

1. You can reuse/call the get_margin_scores function or rewrite it in this function.
2. Instructions for the get_margin_scores apply for this too.

In [51]:
```python
def get_margin_scores_test(X, X_test, y, param):

    ### Your code here

    # calculate the scores for the test set
    final_y_test = get_margin_scores(X, X_test, y, param)

    # define lists to save accuracies
    top1_acc, top2_acc = [], []

    # generate 30 different predictions on val
    for _ in range(30):

        # split data into train and validation
        X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.3)

        # save predictions on val
        y_pred_valid = get_margin_scores(X_train, X_valid, y_train, param)

        # compute top1 and top5 predictions
        predictions_valid, predictions_top5 = get_predictions(y_pred_valid)

        # compute final accuracies, and save on a list
        top1_acc.append(sum(predictions_valid==y_valid)/len(y_valid))
        top2_acc.append(sum([i in j for i,j in zip(y_valid,predictions_top5[:,:2])])/len(y_valid))

    # save final averaged accuracies
    final_acc = [np.mean(top1_acc), np.mean(top2_acc)]


    return final_y_test, final_acc[0], final_acc[1]
```

```
In [52]: y_pred_test, top1_acc, top2_acc = get_margin_scores_test(X, X_test, y, param)
```

```
[23:35:29] WARNING: /Users/travis/build/dmlc/xgboost/src/learner.cc:480:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[23:35:30] WARNING: /Users/travis/build/dmlc/xgboost/src/learner.cc:480:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[23:35:30] WARNING: /Users/travis/build/dmlc/xgboost/src/learner.cc:480:
Parameters: { silent } might not be used.
```

```
In [53]: assert type(y_pred_test) == np.ndarray, ""
         assert y_pred_test.shape == (89, 11)
```

```
In [54]: #(2 points)
         # Hidden tests here
```

```
In [55]: #(2 points)
         # Hidden tests here
```

```
In [56]: #(2 points)
         # Hidden tests here
```

Computing the sector predictions

Note: This section is not evaluated. This is merely to see your predictions on the test set and report the top1 and top2 accuracy

In [57]:
```python
predictions1, predictions5 = get_predictions(y_pred_test) #This line will fail if get_predictions is not defined
i=0
print("%10s" % "Test ID", "%30s" % "Top prediction", "%30s" % "Second Top prediction")
for entry in predictions5[:, :2]:
    print("%10s" % test_nos[i], "%30s" % number2sectorName[entry[0]], "%30s" % number2sectorName[entry[1]])
    i += 1
top1_acc_perc = top1_acc*100
top2_acc_perc = top2_acc*100
print("\n\n\nTop 1 accuracy: %s" % top1_acc_perc, "%")
print("\nTop 2 accuracy: %s" % top2_acc_perc, "%")
```

```
   Test ID                 Top prediction          Second Top prediction
         0                      Utilities     Telecommunication Services
        10         Information Technology                         Energy
        11                      Materials                    Real Estate
        12                      Materials                    Industrials
        13         Consumer Discretionary                    Real Estate
        14                     Financials         Consumer Discretionary
        15         Consumer Discretionary         Information Technology
        16         Information Technology                    Industrials
        17                         Energy         Consumer Discretionary
        18                         Energy         Consumer Discretionary
        19                         Energy                    Health Care
         1                     Financials               Consumer Staples
        20               Consumer Staples         Information Technology
        21                      Materials                    Industrials
        22         Consumer Discretionary         Information Technology
        23                     Financials     Telecommunication Services
        24                    Industrials                      Materials
        25         Consumer Discretionary         Information Technology
        26                     Financials         Information Technology
        27                    Industrials                      Materials
        28         Consumer Discretionary         Information Technology
        29         Information Technology                      Utilities
         2                    Health Care         Consumer Discretionary
        30                      Materials                    Health Care
        31                      Materials                    Industrials
        32         Consumer Discretionary                     Financials
        33         Information Technology                     Financials
        34                      Utilities     Telecommunication Services
        35                     Financials                    Industrials
        36                     Financials         Consumer Discretionary
        37         Consumer Discretionary                         Energy
        38         Consumer Discretionary               Consumer Staples
```

39	Energy	Materials
3	Information Technology	Energy
40	Consumer Staples	Consumer Discretionary
41	Financials	Information Technology
42	Health Care	Industrials
43	Industrials	Materials
44	Information Technology	Telecommunication Services
45	Financials	Health Care
46	Consumer Discretionary	Financials
47	Health Care	Consumer Discretionary
48	Consumer Staples	Health Care
49	Industrials	Materials
4	Financials	Health Care
50	Information Technology	Telecommunication Services
51	Consumer Discretionary	Information Technology
52	Real Estate	Consumer Discretionary
53	Consumer Staples	Health Care
54	Information Technology	Consumer Discretionary
55	Energy	Consumer Discretionary
56	Information Technology	Energy
57	Real Estate	Consumer Discretionary
58	Consumer Staples	Financials
59	Energy	Industrials
5	Health Care	Consumer Discretionary
60	Consumer Discretionary	Consumer Staples
61	Energy	Consumer Staples
62	Industrials	Health Care
63	Consumer Discretionary	Materials
64	Energy	Consumer Discretionary
65	Financials	Real Estate
66	Utilities	Telecommunication Services
67	Consumer Discretionary	Information Technology
68	Consumer Discretionary	Industrials
69	Consumer Staples	Consumer Discretionary
6	Energy	Consumer Discretionary
70	Energy	Materials
71	Consumer Staples	Health Care
72	Utilities	Telecommunication Services
73	Materials	Financials
74	Information Technology	Consumer Discretionary
75	Information Technology	Financials
76	Health Care	Industrials
77	Consumer Staples	Industrials
78	Industrials	Consumer Discretionary
79	Consumer Discretionary	Industrials

7	Information Technology	Financials
80	Industrials	Consumer Discretionary
81	Industrials	Materials
82	Materials	Industrials
83	Industrials	Health Care
84	Health Care	Consumer Staples
85	Materials	Industrials
86	Consumer Discretionary	Consumer Staples
87	Materials	Industrials
88	Energy	Consumer Staples
8	Utilities	Telecommunication Services
9	Consumer Discretionary	Industrials

```
Top 1 accuracy: 75.42372881355932 %

Top 2 accuracy: 88.36158192090394 %
```

In [ ]: