# project

April 18, 2020

[50]:
```
import numpy as np
import pandas as pd
from sklearn import preprocessing

df = pd.read_csv('training.csv') # use test which is smaller for development
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9500 entries, 0 to 9499
Data columns (total 3 columns):
article_number    9500 non-null int64
article_words     9500 non-null object
topic             9500 non-null object
dtypes: int64(1), object(2)
memory usage: 222.8+ KB
```

[51]:
```
df.head()
```

[51]:

| | article_number | article_words | topic |
|---|---|---|---|
| 0 | 1 | open,absent,cent,cent,cent,stock,inflow,rate,k… | FOREX MARKETS |
| 1 | 2 | morn,stead,end,end,day,day,day,patch,patch,pat… | MONEY MARKETS |
| 2 | 3 | socc,socc,world,world,recent,law,fifa,fifa,fif… | SPORTS |
| 3 | 4 | open,forint,forint,forint,forint,cent,cent,ste… | FOREX MARKETS |
| 4 | 5 | morn,complet,weekend,minut,minut,minut,arrow,d… | IRRELEVANT |

[52]:
```
input_cols = ['article_words']
out_cols = ['topic']
X = df[input_cols]
y = df[out_cols]
split = int(X.shape[0] * 0.9)
#X_t = X[:split]
```

```
#y_t = y[:split]
#X_v = X[split:]
#y_v = y[split:]
```

[59]:
```
TOP = 5
words = set() # every word in the doc
top_words = set() # a set of TOP # words from each article
features = [] # array of array of top word
for i in range (X.shape[0]):
    tally = dict()
    # this for loop get all words and count their frequency of each article,
    for word in X['article_words'][i].split(','):
        words.add(word)
        if word not in tally.keys():
            tally[word] = 1
        else:
            tally[word] +=1
    #print(tally)
    sorted_tally = sorted(tally.items(), key=lambda kv: kv[1], reverse=True)
    #print(sorted_tally[:TOP], y['topic'][i])

    # this loop puts top words to the list of features
    loc_feat = []
    for j in range(TOP):
        top_words.add(sorted_tally[j][0])
        loc_feat.append(sorted_tally[j][0])
    features.append(loc_feat)

print(len(words))
#print(words)
print(len(top_words))
#print(top_words)
#print(features)
```

```
35823
6486
```

[60]:
```
index = dict()
count = 0
for word in top_words:
    index[word] = count
    count+=1
# print(index)
```

[ ]:
```
```

[63]:
```
bool_features = []
for i in range (len(features)):
    bool_f = [0] * len(top_words)
    for t in range (TOP):
        bool_f[index[features[i][t]]] = 1
    bool_features.append(tuple(bool_f))
df_cleaned = pd.DataFrame(bool_features, columns = list(top_words))
# print(df_cleaned.head())
# df_cleaned.head().to_csv('tmp.csv') # write to file to validate output
```

[64]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score

enc = OneHotEncoder(handle_unknown='ignore')
enc.fit(y)
y_trans = enc.transform(y).toarray()

START = 2
END = 5 # change 20 to 5 to reduce run time
def optimal_min_leaf(tx, ty, true_y, pred_xarg):
    scores = []
    for k in range(START, END):
        dtc = DecisionTreeClassifier(min_samples_leaf=k)
        dtc.fit(tx, ty)
        score = roc_auc_score(true_y,dtc.predict(pred_xarg))
        scores.append(score)
        print(dtc.min_samples_leaf,score)
    return START+np.argmax(scores), scores


op_min_leaf, test_scores= optimal_min_leaf(df_cleaned[:split], y_trans[:split],␣
 ↪y_trans[split:],df_cleaned[split:])
print('op_min_samples_leaf =', op_min_leaf)
print('score =', test_scores)
```

```
2 0.671532813454826
3 0.6818761244015527
4 0.6602139358295629
op_min_samples_leaf = 3
score = [0.671532813454826, 0.6818761244015527, 0.6602139358295629]
```

[ ]:

[ ]: