Classifying New York Times articles

Three main sections:

Data Collection
Feature Extraction
Classfication

Data Collection

- New York Times articles are used as input Source and nytimesarticle API in Python is used for article extraction and BeautifulSoup is used for crawling the article urls collected
- A dynamic script “Part2/code/dataCollection/nyTimesArticleExtraction.py” is implemented which will collect new data and write the articles collection to “Part2/data/...” folders based on the category
- A total of 75 articles in each classes for training will be used. 30 for testing and 10 other articles from Chicago Tribune as Unknowndata to test the efficiency of our implementation
- Script is capable of taking multiple keywords and scraping multiple pages at once

Method to extract the content of an NYTimes url

```
def parseURL(url):
    content = []
    g = urllib.request.urlopen(url)
    soup = BeautifulSoup(g.read(), 'html.parser')
    # Article = soup.find(id='story') - denoted only the cont
ent

    # Classes that containg the main contents of the articles
    mydivs = soup.findAll("p", {"class": "css-1cy1v93 e2kc3sl
0"})

    # For articles in which  the above class extraction comma
nd fails
    if (mydivs == []):
        mydivs = soup.findAll("p", {"class": "story-body-text
 story-content"})

    if (mydivs != []):
        # Adding title to the content
        content = soup.title.text
```

```
        #return []

    for j in range(0,len(mydivs)):
        content = content + '\n' + mydivs[j].text

    return content
```

Method to collect articles from NYTimes and save them

```
def collectArticles(PAGE, DATE, search_keyword, keyword, cate
gory):
    print('Collecting articles from page:%d' % PAGE)
    articles = api.search(q=search_keyword, begin_date = DATE
, page=PAGE)
    response = articles['response']
    docs = response['docs']

    # Index contains the metadata - url of all the articles c
ollected so far
    index = open("../../data/%s/metadata/index.txt" %(categor
y),"r")

    # Creating an index file if this the first time articles
are collected on a topic
    if (index.readlines() == []):
        index = open("../../data/%s/metadata/index.txt" %(cat
```

```
egory),"w+")
        web_url=[]
        for i in range(0,len(docs)):
            if (keyword.lower() in docs[i]['web_url']): #Chec
ks if articles in from the relevant category
                web_url.append(docs[i]['web_url'])
                index.writelines("%s\n" % docs[i]['web_url'])
    index.close()

    # Reading index file
    index = open("../../data/%s/metadata/index.txt" %(categor
y),"r")
    web_url = index.read()
    web_url = web_url.splitlines()

    # Appending all collected articles to the existing URLs a
nd saving to the index file
    for i in range(0,len(docs)):
        if (keyword.lower() in docs[i]['web_url']): #Checks i
f articles in from the relevant category
            web_url.append(docs[i]['web_url'])
    web_url = list(set(web_url))    #removes duplicates
    index = open("../../data/%s/metadata/index.txt" %(categor
y),"w+")
    for i in range(0,len(web_url)):
            index.writelines("%s\n" % web_url[i])
    index.close()
    print("Articles successfully collected from page:%d and a
```

```
ppended to index file" % PAGE)

    return web_url
```

Feature Extraction

- Created script file “Part2/code/featureExtraction/featureExtraction.py” to extract top 20 features and create a feature matrix for training, testing and unknown data-sets separately
- Data is cleaned before extracting features and stop words are removed from the articles
- The Feature Extraction script when compiled, creates a Feature Matrix in “SVM” format which is used to train classifiers featureMatrixTrainingdata.txt → used for training classfier modes featureMatrixTestinggdata.txt, featureMatrixUnknowndata.txt → used for evaluating the models

Method to extract top 20 features of a class

```
def top_words(sc, path):
    icount=0;
    feature_list=[]
    textRDD=sc.textFile(path)
    words = textRDD.flatMap(lambda x: x.split(' ')).map(lambd
a x: (x, 1))
    wordcount = words.reduceByKey(add).map(lambda (x,y): (y,x
```

```
)).sortByKey(ascending=False).collect()


    for (count, word) in wordcount:
        try:
            mynewstring = word.encode('ascii')
        except:
            #print("there are non-ascii characters in there")
            continue

        if word.lower() in stop_words:
            continue
        else:
            #print("%s: %i" % (word, count))
            if(icount!=20):
                feature_list.append(word.lower())
                icount=icount+1
            else:
                break
```

Method to create and write feature matrix for a dataset

```
def sparse_matrix(sc, path, feature_list,train_length, length
):
    category_list=["Business/","Sports/","Politics/","Health/
"]
    count_list=[]
    sm_file=open('../../data/featureMatrixUnknowndata.txt','w
+')
    Label=-1

    for category in category_list:
        i=0
        Label=Label+1

        for i in range(test_length):
            count_list=[]
            if ("/Testing" in path):
                dir_path=path+str(category)+str(i + train_len
gth)+".txt"
            else:
                dir_path=path+str(category)+str(i)+".txt"

            textRDD=sc.textFile(dir_path)
            words = textRDD.flatMap(lambda x: x.split(' ')).m
ap(lambda x: (x, 1))
            wordcount = words.reduceByKey(add).map(lambda (x,
y): (y,x)).sortByKey(ascending=False).collect()
            count_list.append(Label)

            for feature in feature_list:
                flag=0

                for (count,word) in wordcount:
                    if word == feature:
                        count_list.append(count)
```

Building Classifiers

- Built a Naive Bayes and Neural Network Classifier (“Part2/code/mlclassifiers/*.py”
- Each of these classifers takes in the feature matrix from training data - extracted in the previous step and trains a classification model
- The model is then tested using the Test and Unknown feature Matrix

Naive Bayes Classifier

```
# Load and parse the data file, converting it to a DataFrame.
data = sqlContext.read.format("libsvm").option("delimiter", "
 ").load("test_data.txt")
train = sqlContext.read.format("libsvm").option("delimiter",
" ").load("../../data/featureMatrixTrainingdata.txt")
test = sqlContext.read.format("libsvm").option("delimiter", "
 ").load("../../data/featureMatrixTestingdata.txt")

# create the trainer and set its parameters
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

# train the model
model = nb.fit(train)

# select example rows to display.
predictions = model.transform(test)
predictions.show()
predictionAndLabels = predictions.select("prediction", "label
")

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label
", predictionCol="prediction",
                                              metricName="acc
uracy")
accuracy = evaluator.evaluate(predictions)
```

```
print("Test set accuracy = " + str(accuracy))


#Print the confusion matrix of prediction on test data
metrics = MulticlassMetrics(predictionAndLabels.rdd)


#print(metrics.confusionMatrix().toArray())
print("Confusion Matrix:\n" + str(metrics.confusionMatrix().t
oArray()))
```

Results

Test set accuracy = 94.16%

Confusion Matrix:
[27 - 0 - 2 - 1]
[0 - 30 - 0 - 0]
[2 - 0 - 28 - 0]
[1 - 0 - 1 – 28]

Unknown dataset accuracy = 80.00%

Confusion Matrix:
[[3 - 0 - 0 - 2]
[0 - 4 - 1 - 0]
[0 - 0 - 4 - 1]
[0 - 0 - 0 – 5]]

Neural Network Classifier

```
# Load and parse the data file, converting it to a DataFrame.
train = sqlContext.read.format("libsvm").option("delimiter",
" ").load("../../data/featureMatrixTrainingdata.txt")
test = sqlContext.read.format("libsvm").option("delimiter", "
 ").load("../../data/featureMatrixTestingdata.txt")

# specify layers for the neural network:
# input layer of size 73 (features), two intermediate of size
 100 and 25
# and output of size 4 (classes)
layers = [73, 100,25, 4]

# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=
layers, blockSize=128, seed=1234)

# train the model
model = trainer.fit(train)

# compute accuracy on the test set
result = model.transform(test)
predictionAndLabels = result.select("prediction", "label")
result.select("prediction", "label").show(60, False)
evaluator = MulticlassClassificationEvaluator(metricName="acc
uracy")
print("Test set accuracy = " + str(evaluator.evaluate(predict
ionAndLabels)))
```

```
#Print the confusion matrix of prediction on test data
metrics = MulticlassMetrics(predictionAndLabels.rdd)

#print(metrics.confusionMatrix().toArray())
print("Confusion Matrix:\n" + str(metrics.confusionMatrix().t
oArray()))
```

Results

Test set accuracy = 90.00%

Confusion Matrix:
[[27 - 0 - 2 - 1]
[1 - 28 - 1 - 0]
[3 - 0 - 26 - 1]
[1 - 0 - 2 - 27]]

Unknown set accuracy = 75.00%

Confusion Matrix:
[[3 - 0 - 0 - 2]
[1 - 4 - 0 - 0]
[1 - 0 - 4 - 0]
[0 - 1 - 0 - 4]]