DATA ANALYTICS PIPELINE USING APACHE SPARK

Introduction

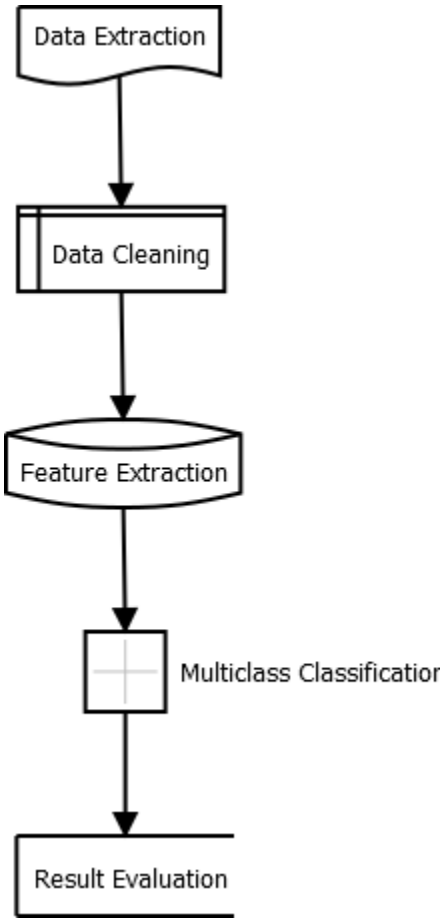
Our project is a text based classification mechanism for categorizing words into specific categories. This is done by building a pipeline for multi-class text-classification using Apache Spark. We have used Spark 2.3.0 build in pyspark[4] kernel supported by python 3.5.2. We have used four categories of dataset, i.e. movies, politics, sports, and business.

There are two part of this project.

The first part is implemented using Kaggle dataset (Titanic) [7] [8]

Learnings from the first part were extensively used in building the second part of the project

Flow Diagram (Pipeline):



Flow Diagram – Pipeline for Multiclass classification model

a. **Data Extraction:**
This stage is the primary stage in building a pipeline mechanism wherein data is extracted by hitting NY-Times API with an appropriate api-key. The extracted data gives a JSON return which has a field named 'web_url'. We are collecting all this web_url and scarping the data using html

parser and Beautiful Soup [6] python library. We have scraped all the paragraph tags (comprising article content) from the extracted data and used it for further processing. This is saved in their respective csv files having format "url", "contents", and "category"

b. Data Cleaning:

This stage is done inside the spark framework and is used for removing stopwords. First the data is filtered using pyspark.ml.feature.RegexTokenizer[1] such that we get only words. We are using pyspark.ml.feature.StopWordsRemover[1] combined with nltk[5] python library and google dictionary for stop words to remove unnecessary words in the text.

c. Feature Extraction:

The data after cleaning categorized by labels as per [0, numLabels) indexing. We are using pyspark.ml.feature.StringIndexer[1] for assigning approriate labels. For extracting features, there are two possible methods, i.e. TFIDF, and Wordcount.

- **TFIDF[1]:** TFIDF stands for Term-Freqency Inverse Document Frequency. This method is used for finding text that uniquely appear on a particular document. The term frequency is the appearance of term t in the document d. IDF is calculated by measuring the information gain from a term in the whole of the corpus D. This is given by the formula below

$$IDF(t, D) = log \frac{|D| + 1}{DF(t, D) + 1}$$

TFIDF is the is caulated by multiplying TF and IDF i.e.

$$TFIDF(t, d) = TF(t, d, D) \times IDF(t, D)$$

In Spark, we can do it using pyspark.ml.feature.HasingTF[1] and pyspark.ml.feature.IDF[1] function

- **WordCount [1]:** In this method, we are just taking a word-count of top N words accross a certain threshold over the whole corpus of data. This is done by pyspark.ml.feature.CountVectorizer[1], which converts a collection of text into a vector of tokens.

The feature is a vector of tokens into a character

d. Multi-class Classification:

This stage handles the training and testing of a data model. The classifiers such as Logistic Regressions, Random Forest, and Naïve Bayes are implemented at this stage which finally creates an evaluation model consisiting of actual labels and predicted labels

- **Logistic Regression:** Logistic Regression is a statistical analysis technique which is used to analyse dataset there are more than one independent variable to determine an outcome.

Generally, the variable has two possible outcome. In our case-study, we can assume that the possibility of a word to belong in a particular category can be true, or false. Mathematically, logistic regression is symbolized as below

$$logit(p) = ln\frac{p}{1-p}$$

Here p is the presence of a word in a category. In Spark, Logistic Regression can be implemented by calling the function, pyspark.ml.classification.LogisticRegression[2]. It takes in certain parameters like maxIter, value of alpha etc.

- **Naïve Bayes:** Naïve Bayes is among the primary set classifier which uses Bayes algorithm to determine priorities with a strong assumption that the different classes are independent. Naïve Bayes classifier is derived from the below mathematical formula

$$posterior = \frac{prior\ x\ likelihood}{evidence}$$

Here posterior priority is calculated some likelihood based on some evidence taking into account the prior priority of the independent set of variables. In Spark, Naïve Bayes can be implemented using the method pyspark.ml.classificaton.NaiveBayes[2] classifier.

- **Random Forest:** Random Forest belongs to the ensemble set of classifiers. This is basically the collection of Decision Tree where the predictions are made by aggregating the output obtained from the individual decision tree. To improve the performance of the random forest, various ensemble methods like bagging and boosting is used. In Spark, Random Forest is implemented by calling pyspark.ml.classification.RandomForest[2] which takes in parameters like number of trees, depth of the tree etc.

e. Result Evaluation:

A multiclass evaluator takes the responsibilty of deriving accuracy from the evaluation model. This is handled by a pyspark.ml.evaluation.MulticlassClassificationEvaluator[3]

Results:

	TF-IDF (in %)	Count-Vector (in %)
Logistic Regression	**63.1**	**73.3**
Naïve Bayes	**65.4**	**69.2**
Random Forest	**73.7**	**73.2**

References:

[1] ML Features Extraction: https://spark.apache.org/docs/2.2.0/ml-features.html

[2] ML Classification Algorithms: https://spark.apache.org/docs/2.2.0/ml-classification-regression.html

[3] ML Multiclass Evaluator: https://spark.apache.org/docs/2.2.0/mllib-evaluation-metrics.html

[4] Spark, PySpark: http://spark.apache.org/docs/2.2.0/api/python/pyspark.html

[5] NLTK Python Library: https://www.nltk.org/

[6] Beautiful Soup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[7] Titanic Kaggle competition: https://benfradet.github.io/blog/2015/12/16/Exploring-spark.ml-with-the-Titanic-Kaggle-competition

[8] Titanic Kaggle competition tutorial in pyspark: https://creativedata.atlassian.net/wiki/spaces/SAP/pages/83237142/Pyspark+-+Tutorial+based+on+Titanic+Dataset