Introduction to Big Data
Assignment 2 Report

Team: Fadi Younes, Artur Akhmetshin, Dinar Zayahov
Group: DS18-02

1 Task Description

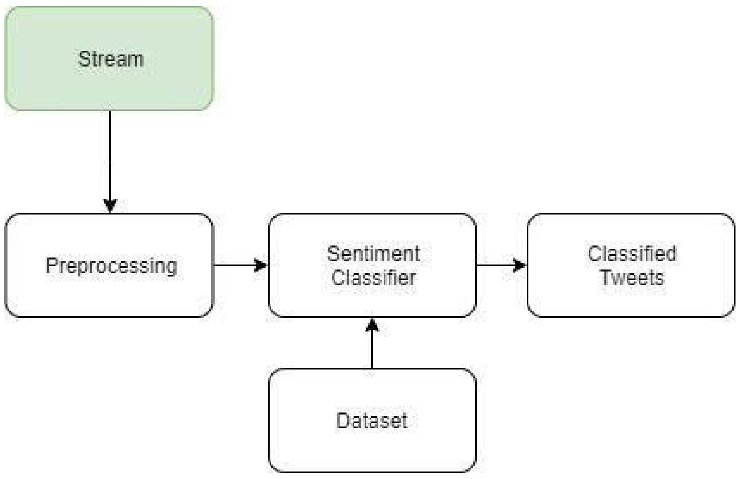
1.1 Goal

Get tweets from the stream and perform preprocessing, classify tweets by the sentiment.

1.2 **Project Structure**:

```
─── src
 |  └──── main
 |      ├──── resources
 |        └──── stopWords.txt
 |      └──── scala
 |        ├──── Preprocessing.scala
 |        └──── Tweets.scala
```

1.3 Code pipeline

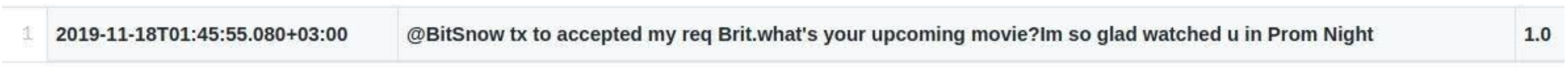


2 Tweets Classification Description

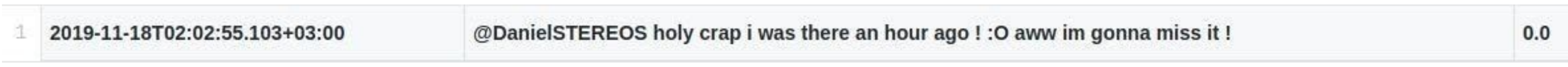
2.1 Stream Processing

From the stream we receive tweets and they are being preprocessed.
After that program creates output directory.
During the stream execution next folders are created: checkpoints, result and wordCount. The first created automatically by stream processing and contains system state, second contains tweet information, its timestamp, text and predicted label in csv format, last contains word count in file part-###.

1 is positive sentiment
0 is negative



1	2019-11-18T01:45:55.080+03:00	@BitSnow tx to accepted my req Brit.what's your upcoming movie?Im so glad watched u in Prom Night	1.0



1	2019-11-18T02:02:55.103+03:00	@DanielSTEREOS holy crap i was there an hour ago ! :O aww im gonna miss it !	0.0

2.2 Preprocessing

In order to build classifier and get good f1 score we need to perform some preprocessing steps: remove aliases, links, punctuation, stop words and then make text lowercase.

2.3 Dataset and feature extraction

The task is to figure out positive and negative sentiment of tweets, so we decided to choose the fourth dataset to train the model.

To get good results in classification we tried following feature extraction approaches: word2vec, CountVectorizer, HashingTF
which gave different F1 scores in combination with different classification models

Word2Vec is an Estimator which takes sequences of words representing documents and trains a Word2VecModel. The model maps each word to a unique fixed-size vector. The Word2Vec Model transforms each document into a vector using the average of all words in the document

CountVectorizer and CountVectorizerModel aim to help convert a collection of text documents to vectors of token counts. The CountVectorizerModel produces sparse representations for the documents over the vocabulary.During the fitting process,
CountVectorizer will select the top vocabSize words ordered by term frequency across the corpus.

HashingTF is a Transformer which takes sets of terms and converts those sets into fixed-length feature vectors.

2.4 Description of Algorithms

We decided to test embedded spark implementations of algorithms such as linear SVM, random forest classifier, naive bayes classifier, and logistic regression.

As we have only two classes we use binomial logistic regression
For binary classification problems, the algorithm outputs a binary logistic regression model. Given a new data point, denoted by x, the model makes predictions by applying
the logistic function. Unlike linear SVMs, the raw output of the logistic regression model has a probabilistic interpretation.

Random forests are ensembles of decision trees. Random forests combine many decision trees in order to reduce the risk of overfitting. Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

SVMs are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.Support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data points of any class (so-called
functional margin), since in general the larger the margin the lower the generalization error of the classifier. LinearSVC in Spark ML supports binary classification with linear SVM.

Naive Bayes classifiers are "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features.Naive Bayes can be trained very efficiently. With a single pass over the training data, it computes the conditional probability distribution of each feature given each label. For prediction, it applies Bayes’ theorem to compute the conditional probability distribution of each label given an observation.

2.5 Accuracy of each model

Here are results of testing selected algorithms on Twitter Dataset. Data was splitted on train and test in ratio 80/20 using spark sql lib function randomSplit(Array(0.8, 0.2))

F1	Logistic Regression	Random Forest	Naive Bayes	LinearSVC
HashingTF	0.41	0.40	0.65	0.67
CountVectorizer	0.39	0.42	0.67	0.68
Word2Vec	0.42	0.53	0.40	0.40

After testing all feature extractors with all models we decided to use LinearSVC with CountVectorizer to achieve maximum F1 score

WeightPrecision	Logistic Regression	Random Forest	Naive Bayes	LinearSVC
HashingTF	0.31	0.7	0.67	0.45
CountVectorizer	0.31	0.71	0.7	0.71
Word2Vec	0.32	0.58	0.69	0.68

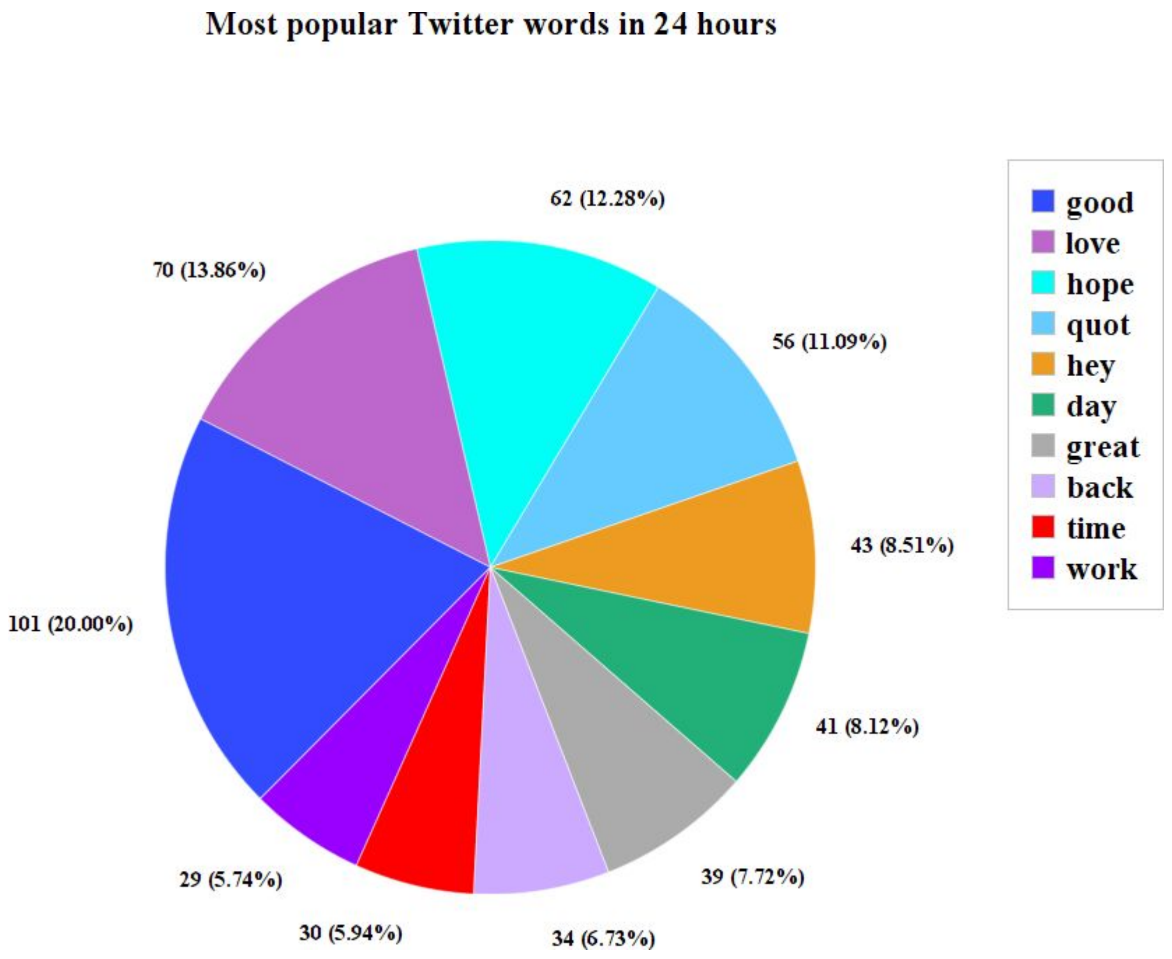
WeightedRecall	Logistic Regression	Random Forest	Naive Bayes	LinearSVC
HashingTF	0.56	0.63	0.67	0.56
CountVectorizer	0.56	0.64	0.7	0.7
Word2Vec	0.57	0.58	0.69	0.68

2.6 24 Hours test

After getting tweets for a day, top 10 most popular words can be seen below as a pie-chart. All files are saved in the review inside the folder src/24hours.

F1 score of model on tweets for 24 hours is 0.68

Script hi.py was used to write all tweets into out.csv, then all tweets was classified by hand and Preprocessing.Eval24HoursPred() used to measure f1


Most popular Twitter words in 24 hours
101 (20.00%)
70 (13.86%)
62 (12.28%)
56 (11.09%)
43 (8.51%)
41 (8.12%)
39 (7.72%)
34 (6.73%)
30 (5.94%)
29 (5.74%)
good
love
hope
quot
hey
day
great
back
time
work