*// Topology.java*

**package** piprescott;

**import** **backtype.storm.Config**;

**import** **backtype.storm.LocalCluster**;

**import** **backtype.storm.StormSubmitter**;

**import** **backtype.storm.topology.TopologyBuilder**;

**import** **backtype.storm.tuple.Fields**;

**import** **backtype.storm.generated.StormTopology**;

*/\*\**

*\* Main Topology class to set up Storm topology for COMP518 assignment.*

*\**

*\* @author PI Prescott*

*\*/*

**public** **class** **Topology** {

**public** **static** void main(String[] args) **throws** InterruptedException {

*// build topology*

TopologyBuilder builder = **new** TopologyBuilder();

builder.setSpout("twitter-spout", **new** TwitterSpout());

builder.setBolt("filter-bolt", **new** FilterBolt())

.shuffleGrouping("twitter-spout");

builder.setBolt("positive-bolt", **new** PositiveBolt())

.shuffleGrouping("filter-bolt");

builder.setBolt("negative-bolt", **new** NegativeBolt())

.shuffleGrouping("filter-bolt");

builder.setBolt("score-bolt", **new** ScoreBolt())

.fieldsGrouping("positive-bolt", **new** Fields("tweetId"))

.fieldsGrouping("negative-bolt", **new** Fields("tweetId"));

*// configure...*

Config conf = **new** Config();

conf.setDebug(**true**);

StormTopology topology = builder.createTopology();

*// ...comment/uncomment depending on whether running local/distributed...*

*// configure local cluster*

LocalCluster cluster = **new** LocalCluster();

cluster.submitTopology("local-tweet-topology", conf, topology);

Thread.sleep(1000000);

cluster.shutdown();

*// // configure distributed cluster*

*// conf.setNumWorkers(3);*

*// conf.setMaxSpoutPending(5000);*

*// StormSubmitter.submitTopology("tweet-topology", conf, topology);*

}

}

--------------------------------------------------------------------------------

*// TwitterSpout.java*

**package** piprescott;

**import** **backtype.storm.spout.SpoutOutputCollector**;

**import** **backtype.storm.task.TopologyContext**;

**import** **backtype.storm.topology.OutputFieldsDeclarer**;

**import** **backtype.storm.topology.base.BaseRichSpout**;

**import** **backtype.storm.tuple.Fields**;

**import** **backtype.storm.tuple.Values**;

**import** **backtype.storm.utils.Utils**;

**import** **twitter4j.\***;

**import** **twitter4j.conf.ConfigurationBuilder**;

**import** **java.util.Map**;

**import** **java.util.concurrent.LinkedBlockingQueue**;

*/\*\**

*\* Reads Twitter stream using the twitter4j library. Set to track COVID19 hashtag and keywords.*

*\**

*\* @author PI Prescott*

*\*/*

**public** **class** **TwitterSpout** **extends** BaseRichSpout {

*// Queue for tweets*

**private** LinkedBlockingQueue<Status> queue;

*// stream of tweets*

**private** TwitterStream twitterStream;

*// standard Storm Spout output collector*

**private** SpoutOutputCollector collector;

*// open Spout*

**public** void open(Map conf, TopologyContext context,SpoutOutputCollector collector) {

**this**.collector = collector;

*// configure Twitter API connection*

ConfigurationBuilder cb = **new** ConfigurationBuilder();

cb.setDebugEnabled(**true**)

.setOAuthConsumerKey("xxxx")

.setOAuthConsumerSecret("xxxxx")

.setOAuthAccessToken("xxxxx")

.setOAuthAccessTokenSecret("xxxxx");

**this**.twitterStream = **new** TwitterStreamFactory(cb.build()).getInstance();

**this**.queue = **new** LinkedBlockingQueue<Status>();

*// listen for new Twitter status messages*

**final** StatusListener listener = **new** StatusListener() {

**public** void onStatus(Status status) {

*// add tweets to queue*

queue.offer(status);

}

**public** void onDeletionNotice(StatusDeletionNotice sdn) {}

**public** void onTrackLimitationNotice(int i) {}

**public** void onScrubGeo(long l, long l1) {}

**public** void onException(Exception e) {}

**public** void onStallWarning(StallWarning warning) {}

};

twitterStream.addListener(listener);

*// track tweets related to coronavirus*

**final** FilterQuery query = **new** FilterQuery();

query.track(**new** String[]{"#COVID19","Vaccine","COVIDVaccine"});

twitterStream.filter(query);

}

**public** void nextTuple() {

**final** Status status = queue.poll();

**if** (status == **null**) {

*// wait if necessary*

Utils.sleep(50);

} **else** {

*// emit tweets as they come*

collector.emit(**new** Values(status));

}

}

**public** void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(**new** Fields("tweet"));

}

**public** void close() {

twitterStream.shutdown();

}

}

--------------------------------------------------------------------------------

*// FilterBolt.java*

**package** piprescott;

**import** **backtype.storm.topology.BasicOutputCollector**;

**import** **backtype.storm.topology.OutputFieldsDeclarer**;

**import** **backtype.storm.topology.base.BaseBasicBolt**;

**import** **backtype.storm.tuple.Fields**;

**import** **backtype.storm.tuple.Tuple**;

**import** **backtype.storm.tuple.Values**;

**import** **twitter4j.Status**;

**import** **java.util.Arrays**;

**import** **java.util.ArrayList**;

**import** **java.util.HashSet**;

**import** **java.util.Set**;

*/\*\**

*\* Bolt to filter irrelevant words by matching them with set.*

*\**

*\* @author PI Prescott*

*\*/*

**public** **class** **FilterBolt** **extends** BaseBasicBolt {

*// set of irrelevant words*

**private** Set<String> irrelevantWords = **new** HashSet<String>(Arrays.asList(**new** String[] {

"http", "https", "the", "you", "que", "and", "for", "that",

"like", "have", "this", "just", "with", "all", "get", "about",

"can", "was", "not", "your", "but", "are", "one", "what",

"out", "when", "get", "lol", "now", "para", "por", "want",

"will", "know", "good", "from", "las", "don", "people", "got",

"why", "con", "time", "would", "is", "at", "football"

}));

*// simple tracking of tweets as they come*

**private** int tweetId = 0;

**public** void execute(Tuple input,BasicOutputCollector collector) {

*// Convert tweet text to array of words*

Status status = (Status) input.getValueByField("tweet");

String tweetText = status.getText();

String text = tweetText

.replaceAll("\\p{Punct}", "")

.replaceAll("\\r|\\n", "")

.toLowerCase();

String[] words = text.split(" ");

*// Create an extensible ArrayList for filtered words*

ArrayList<String> filteredWords = **new** ArrayList<String>();

**for** (String word: words){

**if** (!irrelevantWords.contains(word)){

filteredWords.add(word);

}

}

*// pass filtered words as string for simplicity*

String filteredText = filteredWords.toString();

collector.emit(**new** Values(tweetId, tweetText, filteredText));

*// increment tweetId*

tweetId++;

}

**public** void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(**new** Fields("tweetId", "tweetText", "filteredText"));

}

**public** void cleanup() { }

}

--------------------------------------------------------------------------------

*// PositiveBolt.java*

**package** piprescott;

**import** **backtype.storm.topology.BasicOutputCollector**;

**import** **backtype.storm.topology.OutputFieldsDeclarer**;

**import** **backtype.storm.topology.base.BaseBasicBolt**;

**import** **backtype.storm.tuple.Fields**;

**import** **backtype.storm.tuple.Tuple**;

**import** **backtype.storm.tuple.Values**;

**import** **java.util.ArrayList**;

**import** **java.util.Arrays**;

**import** **java.util.HashSet**;

**import** **java.util.Set**;

*/\*\**

*\* Naive Sentiment Analysis:*

*\* Calculate positive sentiment score by counting positive words.*

*\**

*\* @author PI Prescott*

*\*/*

**public** **class** **PositiveBolt** **extends** BaseBasicBolt {

*// set of positive words*

**private** Set<String> positiveWords = **new** HashSet<String>(Arrays.asList(**new** String[] {

"absolutely", "adorable", "accepted", "acclaimed", "accomplish", "accomplishment", "achievement", "action", "active", "admire", "adventure", "affirmative", "affluent", "agree", "agreeable", "amazing", "angelic", "appealing", "approve", "aptitude", "attractive", "awesome", "beaming", "beautiful", "believe", "beneficial", "bliss", "bountiful", "bounty", "brave", "bravo", "brilliant", "bubbly", "calm", "celebrated", "certain", "champ", "champion", "charming", "cheery", "choice", "classic", "classical", "clean", "commend", "composed", "congratulation", "constant", "cool", "courageous", "creative", "cute", "dazzling", "delight", "delightful", "distinguished", "divine", "earnest", "easy", "ecstatic", "effective", "effervescent", "efficient", "effortless", "electrifying", "elegant", "enchanting", "encouraging", "endorsed", "energetic", "energized", "engaging", "enthusiastic", "essential", "esteemed", "ethical", "excellent", "exciting", "exquisite", "fabulous", "fair", "familiar", "famous", "fantastic", "favorable", "fetching", "fine", "fitting", "flourishing", "fortunate", "free", "fresh", "friendly", "fun", "funny", "generous", "genius", "genuine", "giving", "glamorous", "glowing", "good", "gorgeous", "graceful", "great", "green", "grin", "growing", "handsome", "happy", "harmonious", "healing", "healthy", "hearty", "heavenly", "honest", "honorable", "honored", "hug", "idea", "ideal", "imaginative", "imagine", "impressive", "independent", "innovate", "innovative", "instant", "instantaneous", "instinctive", "intuitive", "intellectual",

"intelligent", "inventive", "jovial", "joy", "jubilant", "keen", "kind", "knowing", "knowledgeable", "laugh", "legendary", "light", "learned", "lively", "lovely", "lucid", "lucky", "luminous", "marvelous", "masterful", "meaningful", "merit", "meritorious", "miraculous", "motivating", "moving", "natural", "nice", "novel", "now", "nurturing", "nutritious", "okay", "one", "one-hundred percent", "open", "optimistic", "paradise", "perfect", "phenomenal", "pleasurable", "plentiful", "pleasant", "poised", "polished", "popular", "positive", "powerful", "prepared", "pretty", "principled", "productive", "progress", "prominent", "protected", "proud", "quality", "quick", "quiet", "ready", "reassuring", "refined", "refreshing", "rejoice", "reliable", "remarkable", "resounding", "respected", "restored", "reward", "rewarding", "right", "robust", "safe", "satisfactory", "secure", "seemly", "simple", "skilled", "skillful", "smile", "soulful", "sparkling", "special", "spirited", "spiritual", "stirring", "stupendous", "stunning", "success", "successful", "sunny", "super", "superb", "supporting", "surprising", "terrific", "thorough", "thrilling", "thriving", "tops", "tranquil", "transforming", "transformative", "trusting", "truthful", "unreal", "unwavering", "up", "upbeat", "upright", "upstanding", "valued", "vibrant", "victorious", "victory", "vigorous", "virtuous", "vital", "vivacious", "wealthy", "welcome", "well", "whole", "wholesome"}));

**public** void execute(Tuple input,BasicOutputCollector collector) {

*// get tweet id*

int tweetId = input.getIntegerByField("tweetId");

*// get tweet text*

String tweetText = input.getStringByField("tweetText");

*// get filtered list of words*

String filteredText = input.getStringByField("filteredText");

int length = filteredText.length();

String[] filteredWords = filteredText.substring(1, (length-1))

.split(" ");

*// naive sentiment analysis:*

*// count positive words to give positive sentiment score*

String scoreType = "positive";

int sentimentScore = 0;

**for** (String word: filteredWords){

**if** (positiveWords.contains(word)){

sentimentScore++;

}

}

collector.emit(**new** Values(tweetId, tweetText, scoreType, sentimentScore));

}

**public** void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(**new** Fields("tweetId",

"tweetText",

"scoreType",

"sentimentScore"));

}

**public** void cleanup() { }

}

--------------------------------------------------------------------------------

*// NegativeBolt.java*

**package** piprescott;

**import** **backtype.storm.topology.BasicOutputCollector**;

**import** **backtype.storm.topology.OutputFieldsDeclarer**;

**import** **backtype.storm.topology.base.BaseBasicBolt**;

**import** **backtype.storm.tuple.Fields**;

**import** **backtype.storm.tuple.Tuple**;

**import** **backtype.storm.tuple.Values**;

**import** **java.util.ArrayList**;

**import** **java.util.Arrays**;

**import** **java.util.HashSet**;

**import** **java.util.Set**;

*/\*\**

*\* Naive Sentiment Analysis:*

*\* Calculate negative sentiment score by counting negative words.*

*\**

*\* @author PI Prescott*

*\*/*

**public** **class** **NegativeBolt** **extends** BaseBasicBolt {

*// set of negative words*

**private** Set<String> negativeWords = **new** HashSet<String>(Arrays.asList(**new** String[] {

"abysmal", "adverse", "alarming", "angry", "annoy", "anxious", "apathy", "appalling", "atrocious", "awful", "bad", "banal", "barbed", "belligerent", "bemoan", "beneath", "boring", "broken", "callous", "can't", "clumsy", "coarse", "cold", "cold-hearted", "collapse", "confused", "contradictory", "contrary", "corrosive", "corrupt", "crazy", "creepy", "criminal", "cruel", "cry", "cutting", "dead", "decaying", "damage", "damaging", "dastardly", "deplorable", "depressed", "deprived", "deformed", "deny", "despicable", "detrimental", "dirty", "disease", "disgusting", "disheveled", "dishonest", "dishonorable", "dismal", "distress", "don't", "dreadful", "dreary", "enraged", "eroding", "evil", "fail", "faulty", "fear", "feeble", "fight", "filthy", "foul", "frighten", "frightful", "gawky", "ghastly", "grave", "greed", "grim", "grimace", "gross", "grotesque", "gruesome", "guilty", "haggard", "hard", "hard-hearted", "harmful", "hate", "hideous", "homely", "horrendous", "horrible", "hostile", "hurt", "hurtful", "icky", "ignore", "ignorant", "ill", "immature", "imperfect", "impossible", "inane", "inelegant", "infernal", "injure", "injurious", "insane", "insidious", "insipid", "jealous", "junky", "lose", "lousy", "lumpy", "malicious", "mean", "menacing", "messy", "misshapen", "missing", "misunderstood", "moan", "moldy", "monstrous", "naive", "nasty", "naughty", "negate", "negative", "never", "no", "nobody", "nondescript", "nonsense", "not", "noxious", "objectionable", "odious", "offensive", "old", "oppressive", "pain", "perturb", "pessimistic", "petty", "plain", "poisonous", "poor", "prejudice", "questionable", "quirky", "quit", "reject", "renege", "repellant", "reptilian", "repulsive", "repugnant", "revenge", "revolting", "rocky", "rotten", "rude", "ruthless", "sad", "savage", "scare", "scary", "scream", "severe", "shoddy", "shocking", "sick", "sickening", "sinister", "slimy", "smelly", "sobbing", "sorry", "spiteful", "sticky", "stinky", "stormy", "stressful", "stuck", "stupid", "substandard", "suspect", "suspicious", "tense", "terrible", "terrifying", "threatening", "ugly", "undermine", "unfair", "unfavorable", "unhappy", "unhealthy", "unjust", "unlucky", "unpleasant", "upset", "unsatisfactory", "unsightly", "untoward", "unwanted", "unwelcome", "unwholesome", "unwieldy", "unwise", "upset", "vice", "vicious", "vile", "villainous", "vindictive", "wary", "weary", "wicked", "woeful", "worthless", "wound", "yell", "yucky", "zero",

}));

**public** void execute(Tuple input,BasicOutputCollector collector) {

*// get tweet id*

int tweetId = input.getIntegerByField("tweetId");

*// get tweet text*

String tweetText = input.getStringByField("tweetText");

*// get filtered list of words*

String filteredText = input.getStringByField("filteredText");

int length = filteredText.length();

String[] filteredWords = filteredText.substring(1, (length-1))

.split(" ");

*// count negative words to give negative sentiment score*

String scoreType = "negative";

int sentimentScore = 0;

**for** (String word: filteredWords){

**if** (negativeWords.contains(word)){

sentimentScore++;

}

}

collector.emit(**new** Values(tweetId, tweetText, scoreType, sentimentScore));

}

**public** void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(**new** Fields("tweetId",

"tweetText",

"scoreType",

"sentimentScore"));

}

**public** void cleanup() { }

}

--------------------------------------------------------------------------------

*// ScoreBolt.java*

**package** piprescott;

**import** **backtype.storm.task.TopologyContext**;

**import** **backtype.storm.topology.BasicOutputCollector**;

**import** **backtype.storm.topology.OutputFieldsDeclarer**;

**import** **backtype.storm.topology.base.BaseBasicBolt**;

**import** **backtype.storm.tuple.Fields**;

**import** **backtype.storm.tuple.Tuple**;

**import** **backtype.storm.tuple.Values**;

**import** **java.util.Map**;

**import** **java.util.HashMap**;

**import** **java.io.PrintWriter**;

*/\*\**

*\* Complete naive sentiment analysis by comparing*

*\* positive and negative word-counts; tweet is*

*\* classified as whichever is greater.*

*\**

*\* @author PI Prescott*

*\*/*

**public** **class** **ScoreBolt** **extends** BaseBasicBolt {

**private** PrintWriter writer;

**private** int positiveCount = 0;

**private** int negativeCount = 0;

**private** String sentiment;

**private** HashMap<Integer, Integer> positiveScores = **new** HashMap<Integer, Integer>();

**private** HashMap<Integer, Integer> negativeScores = **new** HashMap<Integer, Integer>();

**public** void prepare(Map stormConf, TopologyContext context){

*// open file to write simple logs in a CSV that could be analysed further*

String fileName = "logs.csv";

**try** {

**this**.writer = **new** PrintWriter(fileName, "UTF-8");

} **catch** (Exception e){

System.out.println("Error: Unable to open logs.csv file");

}

}

**public** void execute(Tuple input,BasicOutputCollector collector) {

*// sentiment is unknown until scores are compared*

sentiment = "unknown";

int positiveScore, negativeScore;

*// get incoming inputs*

int tweetId = input.getIntegerByField("tweetId");

String tweetText = input.getStringByField("tweetText");

String scoreType = input.getStringByField("scoreType");

*// Positive and negative sentiment scores are computed in parallel,*

*// and either could come first, so whichever we receive, we need*

*// to check if we have already received the other.*

*// If so, we can calculate the sentiment.*

*// If not, then we store the score to be found when the other arrives.*

**if** (scoreType=="positive"){

positiveScore = input.getIntegerByField("sentimentScore");

**if** (negativeScores.containsKey(tweetId)){

*// get negativeScore*

negativeScore = negativeScores.get(tweetId);

*// calculate Sentiment*

int sentimentTotal = positiveScore - negativeScore;

**if** (sentimentTotal>0){

sentiment = "Positive";

**this**.positiveCount++;

} **else** **if** (sentimentTotal<0){

sentiment = "Negative";

**this**.negativeCount++;

}

System.out.println(tweetText + "\n\n" + sentiment);

} **else** {

*// put positiveScore*

positiveScores.put(tweetId, positiveScore);

}

} **else** **if** (scoreType=="negative"){

negativeScore = input.getIntegerByField("sentimentScore");

**if** (positiveScores.containsKey(tweetId)){

*// get negativeScore*

positiveScore = positiveScores.get(tweetId);

*// calculate Sentiment*

int sentimentTotal = positiveScore - negativeScore;

**if** (sentimentTotal>0){

sentiment = "Positive";

**this**.positiveCount++;

} **else** **if** (sentimentTotal<0){

sentiment = "Negative";

**this**.negativeCount++;

}

System.out.println(tweetText + "\n\n" + sentiment);

} **else** {

*// put positiveScore*

negativeScores.put(tweetId, negativeScore);

}

}

*// Log the results in CSV format*

String logRecord = tweetId + "," + sentiment + ",'" + tweetText + "'\n";

writer.println(logRecord);

*// Print running record of positive and negative counts*

System.out.println(logRecord);

System.out.println("Positive:");

System.out.println(**this**.positiveCount);

System.out.println("Negative:");

System.out.println(**this**.negativeCount);

}

**public** void declareOutputFields(OutputFieldsDeclarer declarer) {}

**public** void cleanup() {}

}