```python
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
from pyspark.sql import SQLContext
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import Normalizer
from pyspark.sql.functions import desc
import numpy as np


from collections import defaultdict
from pyspark import SparkContext, SparkConf
from math import sqrt, log

# Initialize Spark Context
conf = SparkConf().setAppName("MovieSearchEngine")
sc = SparkContext.getOrCreate(conf)

# Load the data
summaries = sc.textFile("/FileStore/tables/plot_summaries.txt").map(lambda line: line.split('\t'))
metadata = sc.textFile("/FileStore/tables/movie_metadata.tsv").map(lambda line: line.split('\t')).map(lambda x: (x[0], x[2]))
search_terms = sc.textFile("/FileStore/tables/search_terms.txt").collect()

# Define stopwords
stop_words = set(["the", "a", "and", "is", "in", "it", "you", "are"])

# Tokenize and remove stop words
def tokenize_and_remove_stop_words(text):
    return [word for word in text.lower().split() if word not in stop_words]

tokenized_summaries = summaries.map(lambda x: (x[0], tokenize_and_remove_stop_words(x[1])))

# Compute TF
tf = tokenized_summaries.flatMap(lambda x: [((x[0], word), 1) for word in x[1]]).reduceByKey(lambda x, y: x + y)

# Before computing IDF, calculate the count of summaries
summaries_count = tokenized_summaries.count()

# Now, use this count in IDF calculation
idf = tokenized_summaries.flatMap(lambda x: set(x[1])).map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y).map(lambda x: (x[0], log(float(summaries_count) / x[1])))

# Compute TF-IDF
tfidf = tf.map(lambda x: (x[0][1], (x[0][0], x[1]))).join(idf).map(lambda x: (x[1][0][0], (x[0], x[1][0][1] * x[1][1])))

# Function to create a vector for each document
def create_vector(tfidf_scores):
    vector = defaultdict(float)
    length = 0.0
    for word, score in tfidf_scores:
        vector[word] += score
        length += score ** 2
    return (vector, sqrt(length))

# Create a vector for each document
doc_vectors = tfidf.groupByKey().mapValues(create_vector)

# Handle search terms
for term in search_terms:
    query_vector = defaultdict(float)
    for word in tokenize_and_remove_stop_words(term):
        query_vector[word] += 1.0

    query_length = sqrt(sum(val ** 2 for val in query_vector.values()))

    # Compute cosine similarity
    similarities = doc_vectors.map(lambda x: (x[0], sum(x[1][0].get(word, 0.0) * score for word, score in query_vector.items()) / (x[1][1] * query_length)))

    # Get top 10 movies
    top_movies = similarities.top(10, key=lambda x: x[1])

    # Get movie names
    top_movie_names = sc.parallelize(top_movies).join(metadata).map(lambda x: (x[1][1], x[1][0])).collect()

    print("Search term:", term)
    for movie in top_movie_names:
        print(movie)
```

```
Search term: funny movie
('Swapnam Kondu Thulabharam', 0.21877855419851827)
('Iruvattam Manavaatti', 0.18554009591016227)
('Funny Man', 0.17030305823835867)
('Kottarathil Kuttibhootham', 0.20571273118986474)
('Odessa in Flames', 0.16670430457874022)
('Chingari', 0.21013721794124235)
('Josettante Hero', 0.17294203155357696)
("The Major Lied 'Til Dawn", 0.18202882659161884)
('The Mirror', 0.16464439357300395)
('Lu and Bun', 0.16633423494619695)
Search term: action
('Crayon Shin-chan: Action Kamen vs Leotard Devil', 0.28929256352344335)
('Action Man: Robot Atak', 0.4033188474822755)
('Giri', 0.23973185672602845)
('Tiger', 0.21131780670132638)
('Cracker Jack', 0.19938480118489044)
('Time, Forward!', 0.219085867332534)
('The Daredevil Men', 0.2660106051383738)
('Hitler', 0.23084630092054473)
("Someone's Watching Me", 0.21336843004451336)
```