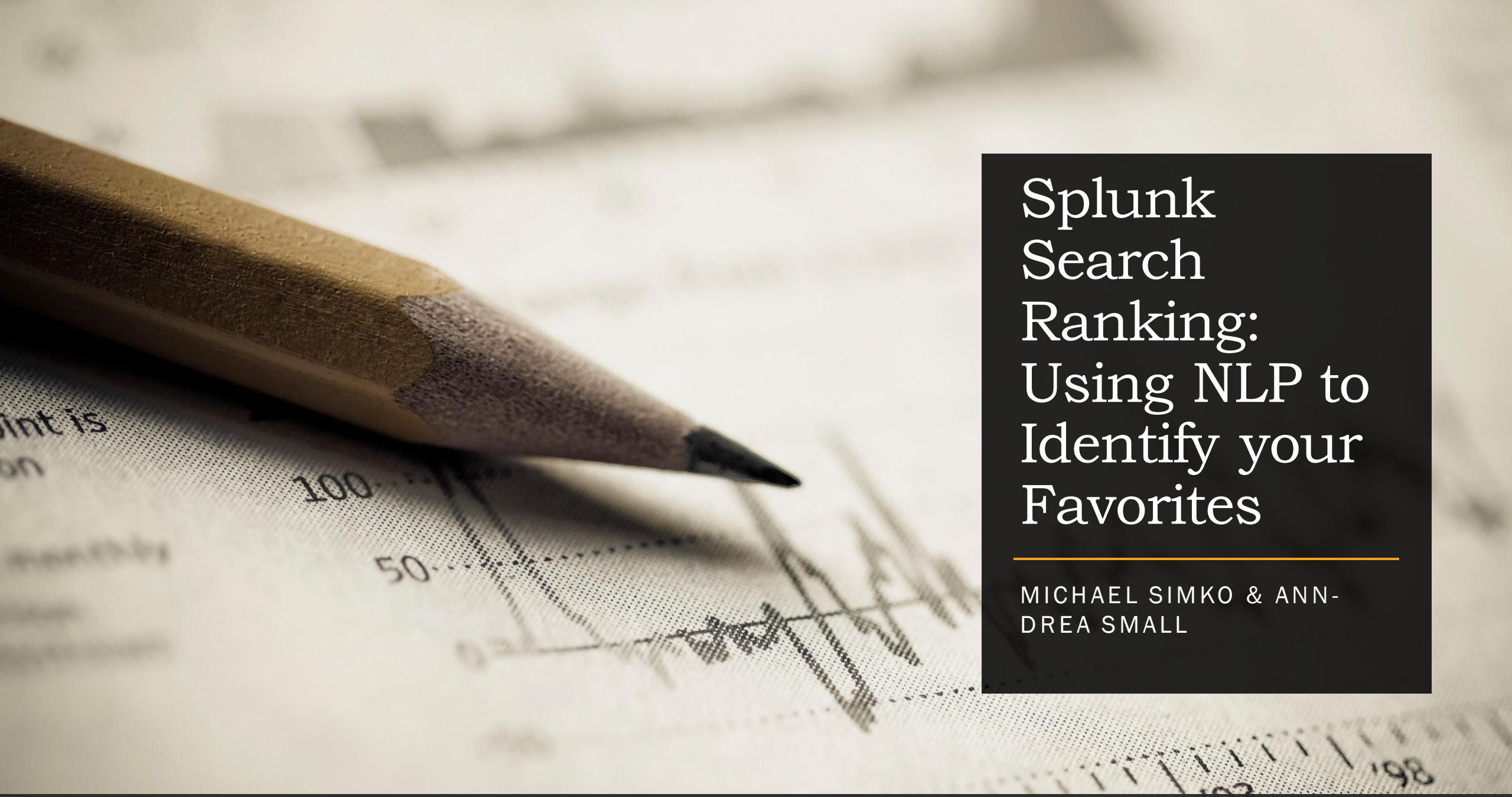


Splunk Search Ranking: Using NLP to Identify your Favorites

MICHAEL SIMKO & ANN-DREA SMALL

The About Us Slide

Ann-Drea Small is a Principal Engineer, and Team Lead of IT Operations Analytics, at Kinney Group. Ann-Drea has a Masters in Quantitative Methods and Modeling.

ann-drea.small@kinneygroup.com

Michael Simko is a Principal Architect at Kinney Group, long-time Splunk Instructor, Professional Services Consultant, and a member of the Splunk Trust.

michael@kinneygroup.com

Agenda

01

Define Natural Language Processing (NLP)

02

Discuss Popular NLP Algorithms

03

Discuss our objective

04

Discuss Machine Learning Toolkit (MLTK)

05

Discuss where to find the data

06

Use sample data to apply our models

What is NLP?

Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken and written. It uses **artificial intelligence** (AI) to take real-world input, process it, and convert it to code that a computer can understand.

NLP Phases

There are two main phases to natural language processing:

Data preprocessing, which involves preparing and "cleaning" text data for machines to be able to analyze it. Data preprocessing puts data in workable form and highlights features in the text that an algorithm can work with.

Algorithm development, which involves using carefully defined rules on cleaned data in order to find analyze the data and find patterns.

Types of Data Preprocessing

Segmentation involves breaking larger text down into phrases or sentences, using things like punctuation to break it up.

Tokenization involves breaking text down into smaller units, such as words, by recognizing the white space between them.

Stop word removal involves removing common words such as “are”, “and”, “the” which are only present to make sentences more cohesive and don’t add much value to the intent of the words

Stemming involves removing stems from root words, so that the algorithm will recognize the root words as the same. For example, “laughs”, “laughing”, “laughed”, “laughter”, “laughable”, “unlaughable”, and “laughably”, all have “laugh” as their root word. The stems, or prefixes and suffixes, are “-s”, “-ing”, “-ed”, “-ter”, “-able”, “un-”, and “-ably”, respectively.

Lemmatization involves removing tense or conjugation from various forms of a verb so that the algorithm will recognize the words as the same. For example, “were”, “was”, “is”, “are”, “am” are all different versions of the verb “to be”.

Part-of-speech tagging involves identifying the which of the 8 parts of speech (noun, pronoun, verb, adverb, adjective, conjunction, preposition, or interjection) that a word belongs to. For example, “I went to the store.”

Types of Algorithm Development

- **Rule-based system** involves using linguistic rules to process text.
- **Machine learning-based system** involves using statistical methods to train data and adjust to be applied to large sets of data as the algorithm learns.
  - **Unsupervised Machine Learning** takes data that hasn't been sorted or classified and tries to find new patterns or relationships in the data. **Clustering** is a popular method used in recommendation engines based on past viewership or purchases.
  - **Supervised Learning** uses sorted data to apply patterns previously seen to new data sets. Based on options provided, it can classify data if the categories have been predefined. **Sentiment analysis** is a great method of classification for social media data.

Top Algorithms for NLP

- **term frequency-inverse document frequency (TFIDF)** is a statistical measure that calculates the importance of a word based on how many times a word appears in a document times the number of documents that the word appears in. The formula for calculation where the term **t** in the document **d** from the document set **D** is as follows:

$$tf\ idf\ (t, d, D) = tf\ (t, d)\ .\ idf\ (t, D)$$

where:

$$tf\ (t, d) = log\ (1 + freq\ (t, d))$$

$$idf\ (t, D) = log\ (\ \frac{N}{count\ (d \in D : t \in d)}\ )$$

Algorithms (cont'd)

- **support vector machine (SVM)** is a supervised machine learning algorithm that classifies data into two categories. In our example, it would be used to help us categorize the search as good or bad based on our criteria.
- **Naïve Bayes** uses conditional probability theory to classify data with tags based on word frequency. It assumes every word in a phrase is independent of the other words and cares nothing for word order. It compares each word to a tag and if the probability that the phrase is related to the tag is higher than the probability that it is not, then that tag is assigned to that phrase. The formula associated is:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Algorithms (cont'd)

- **Decision Tree** is a supervised machine learning algorithm that breaks down similarities based on tree structure. Using "trunk", "branch", and "leaf" as categories, it uses if-then rules to classify data with increasingly close similarities into more detailed groupings.
- **Random Forest** is a combination of decision tree algorithms. It takes the data and puts it into several decision trees and then fits new data to one of the trees already created. It chooses the tree with the nearest similarity.
- **K-nearest neighbors** is a supervised machine learning algorithm that looks classifies new data based on the nearest classification of previous data. For example, if there are two categories available, based on how similar the new data is to the data previously classified each category, it decides the category of the new data point.

Back Story

Splunk can be a powerful search tool that can return results that can help us find the proverbial "needle in a haystack". However, our results are made better by the way the search is written. We have all seen searches that fail to complete. These searches take up time and resources. In large environments with a lot of scheduled searches, poorly written searches can result in skipped searches that fail to execute. And yet, not all skipped searches are written poorly.
So, ***how do we determine if a search is written poorly?***

Our Objective

To determine if our Splunk search is good by evaluating whether it is written using Splunk search best practices parameters.

Splunk Search Best Practices

Define a time range, index, sourcetype, and host

Filter the fields you want to include in your results

Include as many search terms as possible

Limit the use of wildcards

Order search commands

Streaming commands: eval, rex, where, fields, spath

Transforming commands: table, chart, timechart, stats, top, rare

Our Criteria to Determine a Good Search

- One: Is index included?
- Two: Is sourcetype included?
- Three: Do evals come before stats?
- Four: Do we limit fields?

Splunkbase

Python for Scientific Computing (for Mac, Linux, or Windows)

- https://splunkbase.splunk.com/app/2881
- https://splunkbase.splunk.com/app/2882
- https://splunkbase.splunk.com/app/2883

Splunk Machine Learning Toolkit (MLTK)

- https://splunkbase.splunk.com/app/2890

Splunk MLTK Algorithms on GitHub

- https://splunkbase.splunk.com/app/4403

MLTK Algorithms

TFIDF

Syntax

fit TFIDF <field_to_convert> [into <model name>] [max_features=<int>] [max_df=<int>] [min_df=<int>] [ngram_range=<int>-<int>] [analyzer=<str>] [norm=<str>] [token_pattern=<str>] [stop_words=english]

SVM

Syntax

fit SVM <field_to_predict> from <explanatory_fields> [into <model name>] [C=<float>] [gamma=<float>]

MLTK Algorithms (cont'd)

DecisionTreeClassifier

Syntax

fit DecisionTreeClassifier <field_to_predict> from <explanatory_fields> [into <model_name>] [max_depth=<int>] [max_features=<str>] [min_samples_split=<int>] [max_leaf_nodes=<int>] [criterion=<gini|entropy>] [splitter=<best|random>] [random_state=<int>]

Random ForestClassifier

Syntax

fit RandomForestClassifier <field_to_predict> from <explanatory_fields> [into <model name>] [n_estimators=<int>] [max_depth=<int>] [criterion=<gini | entropy>] [random_state=<int>] [max_features=<str>] [min_samples_split=<int>] [max_leaf_nodes=<int>]

Missing Algorithm

K-nearest Neighbor

Since it is included in the Scikit-Learn Python Library, we would need to import it and load it as a dataframe. We could upload it to the bin folder of :

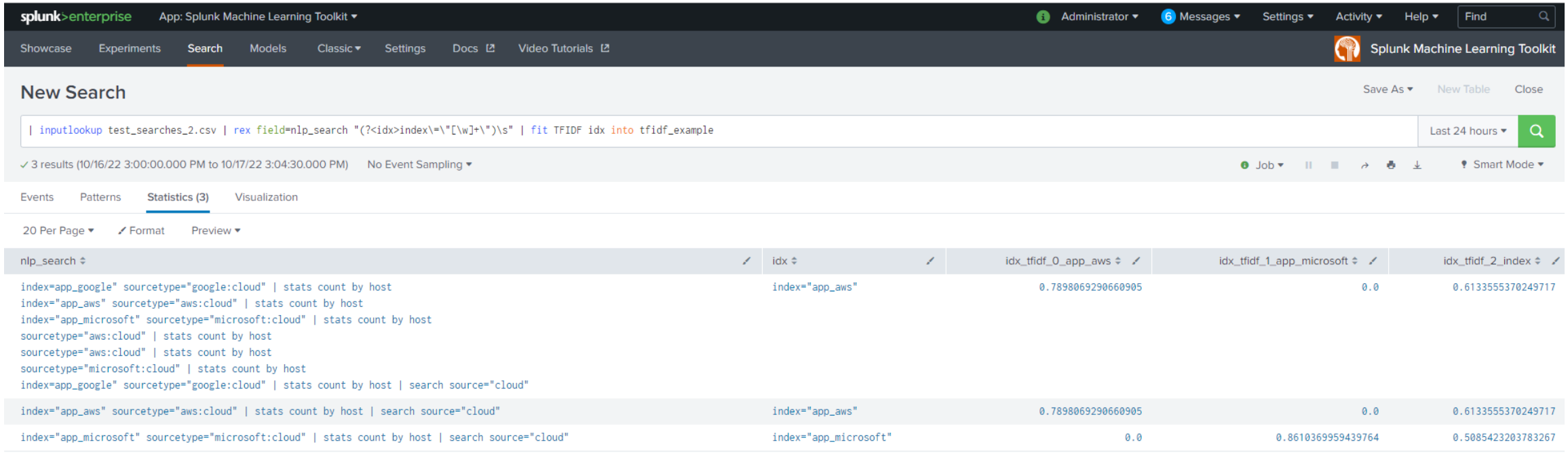
/opt/splunk/etc/apps/SA_mltk_contrib_app/bin/algos_contrib/KNeighborsRegressor.py

```
#!/usr/bin/env python

import pandas as pd

from sklearn.neighbors import KNeighborsRegressor as _KNeighborsRegressor

from codec import codecs_manager
from util.param_util import convert_params


KNeighborsRegressor(ClassifierMixin, BaseAlgo):

    def __init__(self, options):
        self.handle_options(options)

        out_params = convert_params(
```

Data Location

- REST API
  - Saved searches
    - | rest /servicesNS/-/-/saved/searches
  - Dashboards
    - | rest /servicesNS/-/-/data/ui/views
- Audit index
  - Ad-Hoc, Scheduled, API searches
    - Index="_audit" action="search" search="*"
  - CSV lookup
    - | inputlookup test_searches.csv

Splunk MLTK Example



nlp_search	idx	idx_tfidf_0_app_aws	idx_tfidf_1_app_microsoft	idx_tfidf_2_index
index=app_google" sourcetype="google:cloud" | stats count by host index="app_aws" sourcetype="aws:cloud" | stats count by host index="app_microsoft" sourcetype="microsoft:cloud" | stats count by host sourcetype="aws:cloud" | stats count by host sourcetype="aws:cloud" | stats count by host sourcetype="microsoft:cloud" | stats count by host index=app_google" sourcetype="google:cloud" | stats count by host | search source="cloud"	index="app_aws"	0.7898069290660905	0.0	0.6133555370249717
index="app_aws" sourcetype="aws:cloud" | stats count by host | search source="cloud"	index="app_aws"	0.7898069290660905	0.0	0.6133555370249717
index="app_microsoft" sourcetype="microsoft:cloud" | stats count by host | search source="cloud"	index="app_microsoft"	0.0	0.8610369959439764	0.5085423203783267