

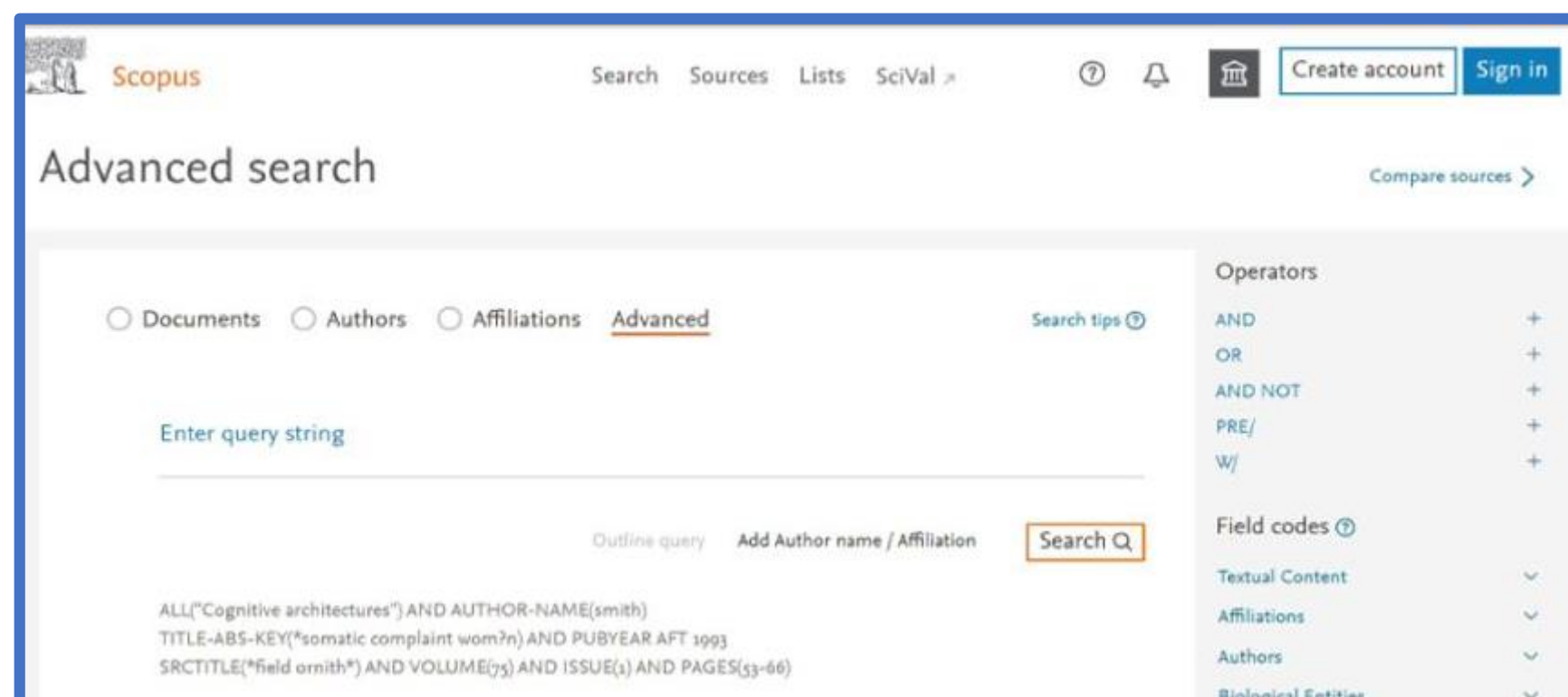
# Scopus<sup>®</sup> text mining

Maria Vallarelli  
Simone Zambetti  
Giorgio Martelli  
11/27/2020

# Business target

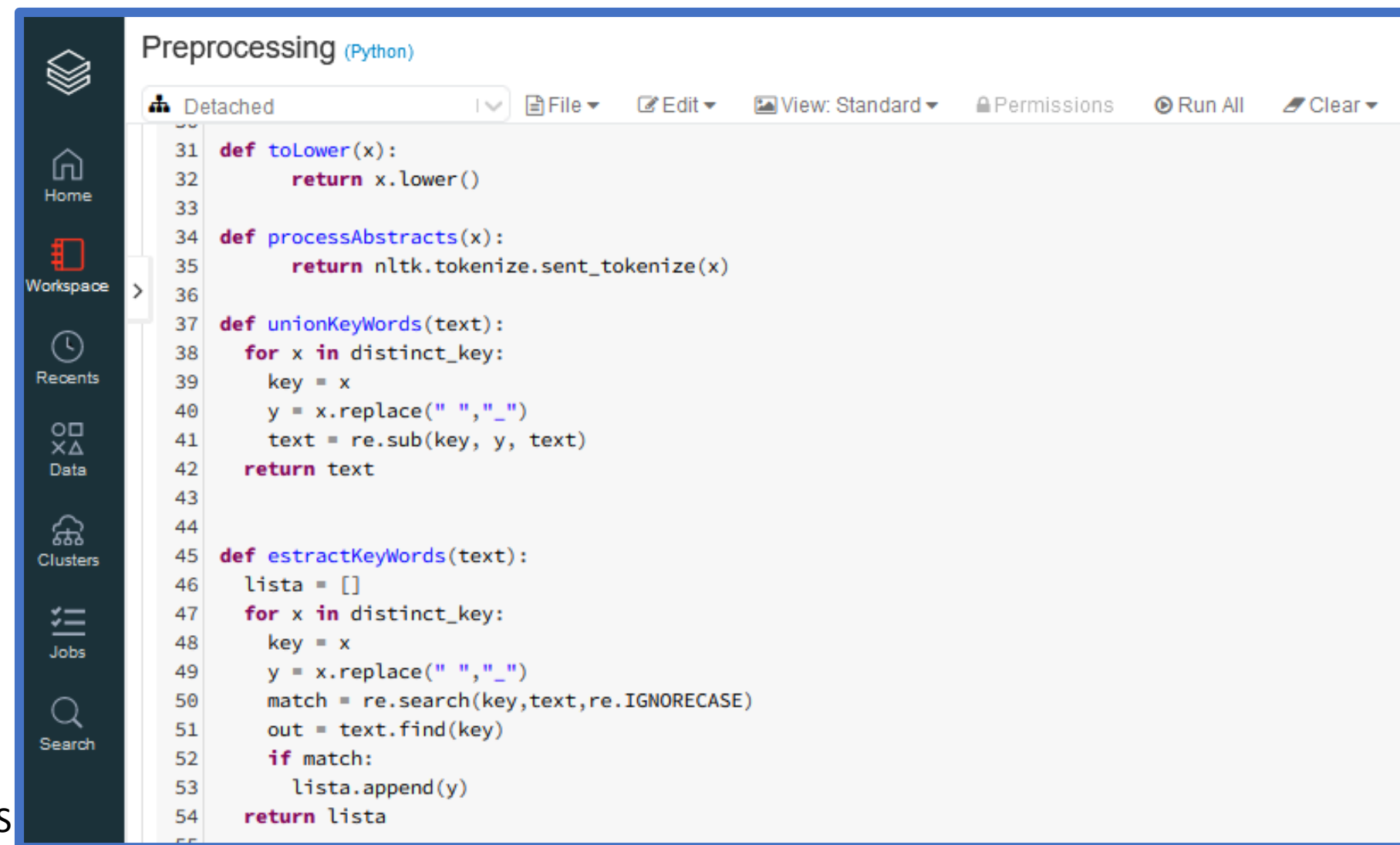
## Classificazione delle keyword di papers accademici in base all'abstract

- Per indicizzare al meglio gli abstract, proponiamo un sistema di raccomandazione delle keywords, in modo tale che il ricercatore, in fase pubblicazione possa individuare facilmente le parole chiave più collegate al contenuto dell'abstract stesso
- Al momento, su scopus le keyword sono inserite manualmente dal ricercatore
- Abbiamo ristretto i paper al campo del machine learning

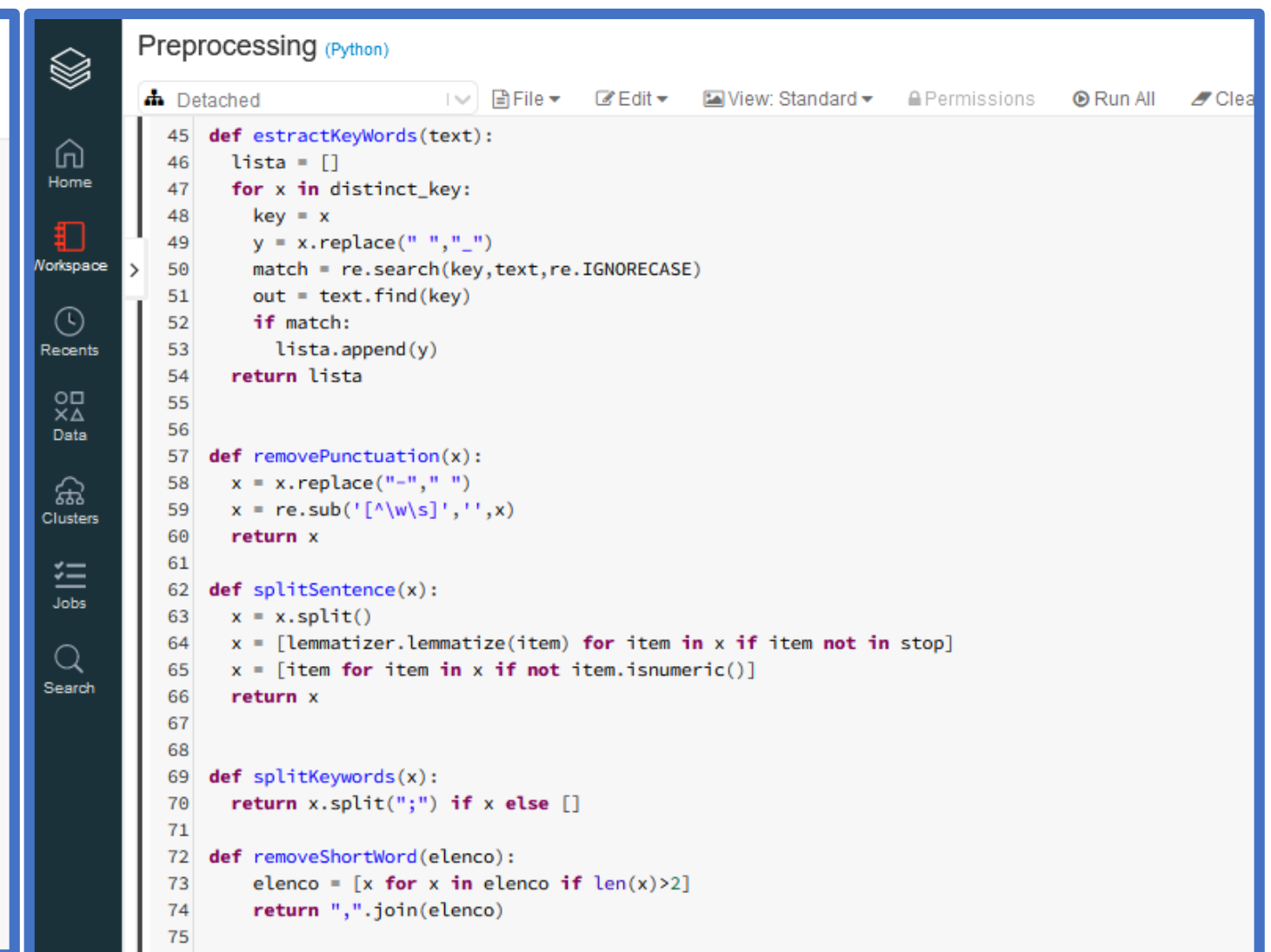


# Scraping e initial profiling

- La fonte dati è stata generata tramite scraping da scopus.com, ed è composta da 19.000 articoli, già scaricati durante un modulo precedente
- Data la grande quantità di dati, non gestibile tramite pandas, abbiamo usato Spark per effettuare un primo preprocessing, usando lemmatization, lowering, stopwords removal e anche un approccio deterministico degli n-grammi
  - Ad esempio, «machine learning» è stato sostituito con «machine\_learning», utilizzando come riferimento le keywords composte da più parole

A screenshot of the Databricks workspace interface. The left sidebar shows navigation icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area displays a Python notebook titled "Preprocessing (Python)". The code in the notebook includes functions for tokenization, keyword extraction, and punctuation removal. The code is as follows:

```
31 def toLower(x):
32     return x.lower()
33
34 def processAbstracts(x):
35     return nltk.tokenize.sent_tokenize(x)
36
37 def unionKeyWords(text):
38     for x in distinct_key:
39         key = x
40         y = x.replace(" ", "_")
41         text = re.sub(key, y, text)
42     return text
43
44
45 def extractKeyWords(text):
46     lista = []
47     for x in distinct_key:
48         key = x
49         y = x.replace(" ", "_")
50         match = re.search(key, text, re.IGNORECASE)
51         out = text.find(key)
52         if match:
53             lista.append(y)
54     return lista
```

A continuation of the Databricks workspace interface, showing the same "Preprocessing (Python)" notebook. The code continues with functions for removing punctuation, splitting sentences, and splitting keywords. The code is as follows:

```
55
56
57 def removePunctuation(x):
58     x = x.replace("-", " ")
59     x = re.sub('[^\w\s]', '', x)
60     return x
61
62 def splitSentence(x):
63     x = x.split()
64     x = [lemmatizer.lemmatize(item) for item in x if item not in stop]
65     x = [item for item in x if not item.isnumeric()]
66     return x
67
68
69 def splitKeywords(x):
70     return x.split(";") if x else []
71
72 def removeShortWord(elenco):
73     elenco = [x for x in elenco if len(x)>2]
74     return ", ".join(elenco)
75
76 #def processKeywords(x):
```



# Pre-processing II

- Dopo aver constatato che non tutti i paper avevano delle keyword, abbiamo creato un corpus dalla concatenazione del testo dell'abstract e delle keyword
- Le keyword sono state anch'esse trasformate in ngrams in modo «deterministico»
- Dunque, su questo corpus abbiamo applicato spacy, rimuovendo verbi, aggettivi, avverbi, pronomi e congiunzioni che non aggiungono valore all'embedding

```
1  !pip install spacy
2  import spacy
3  import re
4  import tqdm
5
6  nlp = spacy.load('en_core_web_sm')
7
8  excluded_tags = {"VERB", "ADJ", "ADV", "ADP", "PROPN", "CCONJ", "DET"}
9  document = [line.strip() for line in text]
10
11 sentences = document[:]
12 new_sentences = []
13 for sentence in sentences:
14     new_sentence = []
15     sentence.replace(",", " ")
16     for token in nlp(sentence):
17         if token.pos_ not in excluded_tags:
18             new_sentence.append(token.text)
19     new_sentence = re.sub(",,+", ", ", "".join(new_sentence))
20     new_sentence = new_sentence.strip(", ")
21     new_sentences.append(new_sentence)
```



# Pre-processing III

- Dopo di che, è stato effettuato un phrases di abstract + keywords, come segue:

```
[51] 1 from gensim.models.phrases import Phrases
      2 bigram = Phrases(corpus, min_count=3, threshold=0.2)
      3 bigrams = [bigram[item] for item in df_tf_idf]
      4 ngrams = [bigram[item] for item in bigrams]
      5 ngrams_ = [", ".join(item) for item in ngrams]
```

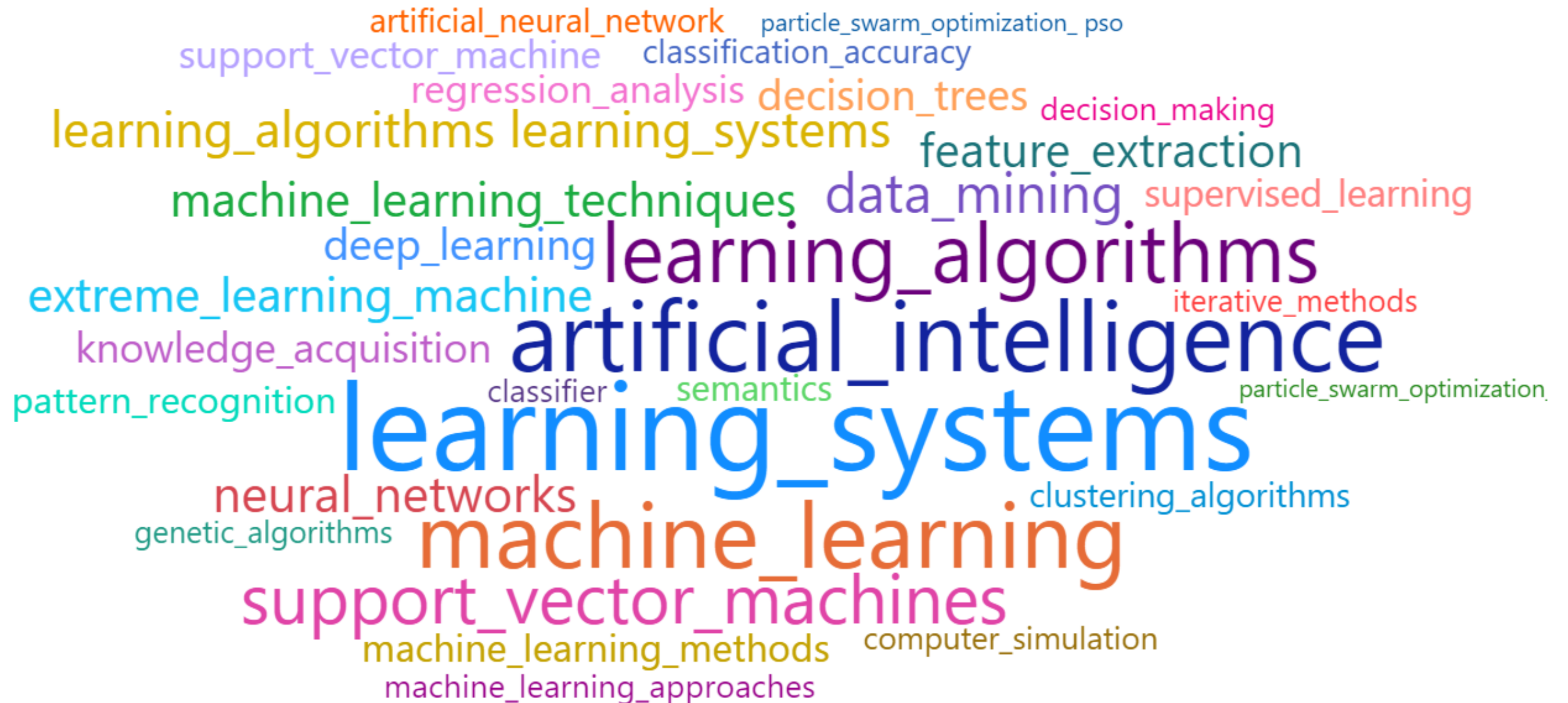
- Per individuare le parole più importanti su questo nuovo campo combinato, è stato effettuato un tf-idf per i 30 termini più importanti:

```
[55] 1 from sklearn.feature_extraction.text import TfidfVectorizer
      2 tfidf = TfidfVectorizer(max_features=1000, stop_words=stopwords, ngram_range=(1, 2))
      3 X = tfidf.fit_transform(ngrams_)
      4 df1 = pd.DataFrame(X.toarray(), columns=tfidf.get_feature_names())
```



# Pre-processing III (contd.)

- Le seguenti sono le 30 parole più importanti secondo tf-idf:





# Embedding models



- Abbiamo dunque optato per fasttext e non Word2Vec, perché ci consente di usare parole non in vocabolario

```
[89] 1 ft_model_cb.most_similar('machine_learning', topn = 3)
      2

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: DeprecationWarning: Call to deprecated
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second
if np.issubdtype(vec.dtype, np.int):
[('applied_machine_learning', 0.9698399305343628),
 ('hybrid_machine_learning', 0.9392151832580566),
 ('learning_automata', 0.9099183678627014)]

[117] 1 ft_model_sg.most_similar('machine_learning', topn = 3)

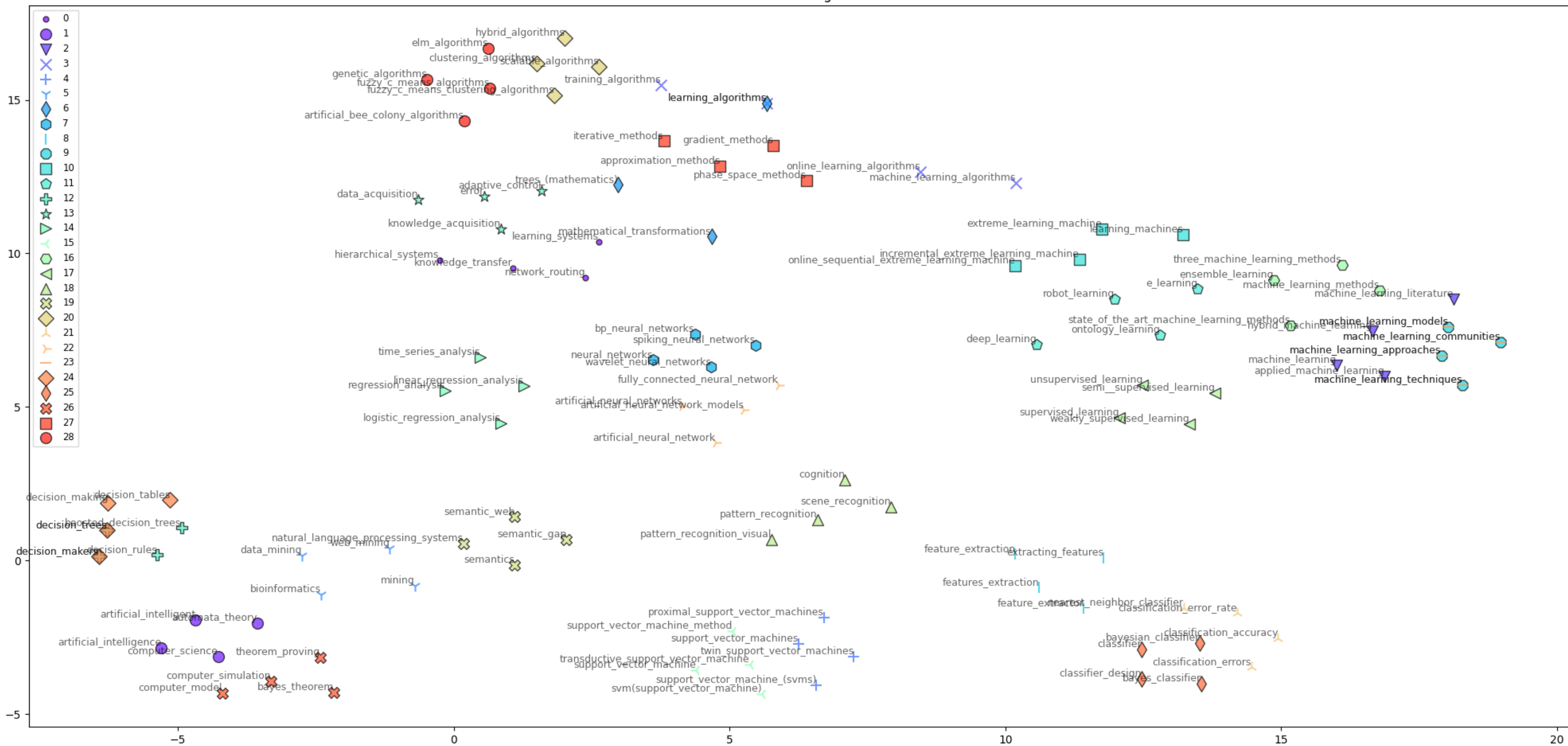
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated
"""Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second
if np.issubdtype(vec.dtype, np.int):
[('applied_machine_learning', 0.9698399305343628),
 ('hybrid_machine_learning', 0.9392151832580566),
 ('learning_automata', 0.9099183678627014)]
```

- Sono stati creati dunque vari modelli, con configurazioni diverse, ad esempio modificando window, min e max count, sg (sia CBOW che SkipGrams). Tramite silhouette-score, abbiamo scelto il modello che clusterizza meglio:

```
1 from gensim.models import KeyedVectors
2 import os
3 from sklearn.metrics import silhouette_score
4
5 wn = out_df
6 with open('silhouette_grid.csv', 'w') as file:
7     file.write('model,silhouette\n')
8     for filename in os.listdir('model'):
9         model = KeyedVectors.load_word2vec_format('model/'+filename, binary=False)
10         in_vocab = [True if word in model.wv.vocab else False for word in wn.word]
11         wn['in_vocab'] = in_vocab
12         wn_in_vocab = wn[wn['in_vocab']==True]
13         vectors = [model[i] for i in wn_in_vocab['word']]
14         sil = silhouette_score(vectors, wn_in_vocab['cat'])
15         print(f'{filename}: sil={sil}')
16         file.write(f'{filename},{sil}\n')
17 file.close()

ft_model_cb.vec: sil=0.234257772564888
ft_model_sg.vec: sil=0.22233004868030548
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: DeprecationWarning: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
# This is added back by InteractiveShellApp.init_path()
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: DeprecationWarning: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
# This is added back by InteractiveShellApp.init_path()
```

# UMAP

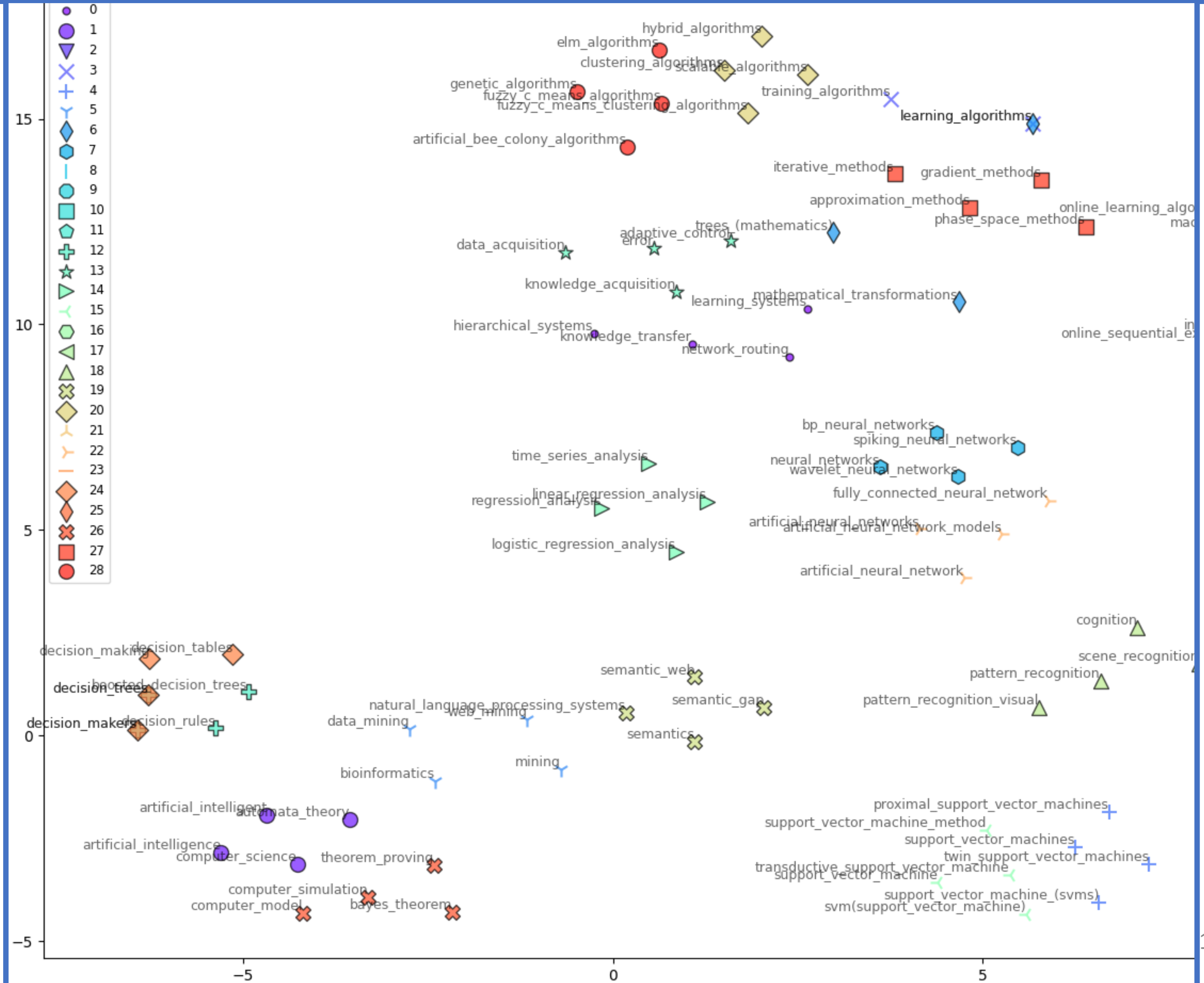




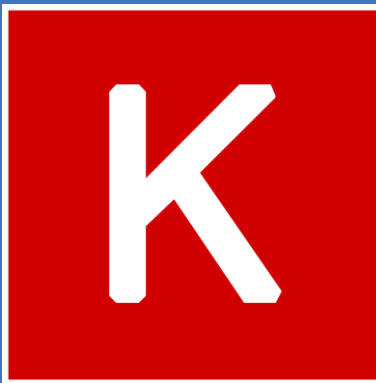
# UMAP II



# UMAP III



# Embedding classification



- Una volta trovato l'embedding migliore, è stato creato un modello di classificazione dei testi degli abstract per individuare le keyword
- Partendo dallo stesso dataset, abbiamo trasformato 'esplodendo' gli abstract per le singole keyword, in modo da ottenere un dataframe in cui vi è una relazione uno a molti tra keyword (Y) e abstract (X)
- Purtroppo la cardinalità del dataset non è sufficiente per avere un'accuracy ragionevole su tutte le keywords, dunque abbiamo utilizzato solo sottoinsieme per fare convergere meglio l'algoritmo LSTM, considerando solo le Y più presenti nei singoli abstract:

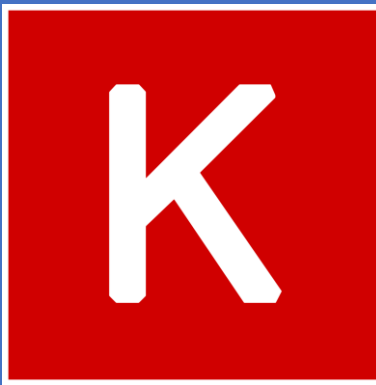
```
[ ] 1 import pandas as pd
    2
    3 df = pd.read_csv("ds_train1.csv", encoding = "UTF-8")
    4 df_not_na = df[~(df['abstract_cleaned'].isna())]
    5
[ ] 1 abstract_0 = df_not_na['abstract_cleaned']
    2 abstract = abstract_0.apply(lambda x: x.split(","))
    3
[ ] 1 df_not_na['abstract_cleaned'] = abstract
    2 df_not_na['Keywords'] = df_not_na['Keywords'].apply(lambda x: x.split(","))
    3 df = df_not_na.explode('Keywords')
    4 df[:10]
```

|   | year | abstract  | Keywords                 | text  | abstract_spacy                                    |
|---|------|---|--------------------------|---|---|
| 0 | 2014 | visual,analytics,inherently,collaboration,curr... | analytic_provenance      | visual,analytics,inherently,collaboration,curr... | analytics collaboration analytics interaction ... |
| 0 | 2014 | visual,analytics,inherently,collaboration,curr... | applied_machine_learning | visual,analytics,inherently,collaboration,curr... | analytics collaboration analytics interaction ... |
| 0 | 2014 | visual,analytics,inherently,collaboration,curr... | user_interaction         | visual,analytics,inherently,collaboration,curr... | analytics collaboration analytics interaction ... |
| 0 | 2014 | visual,analytics,inherently,collaboration,curr... | flow_visualization       | visual,analytics,inherently,collaboration,curr... | analytics collaboration analytics interaction ... |

|                                 |      |
|---------------------------------|------|
| learning_systems                | 8587 |
| artificial_intelligence         | 5873 |
| machine_learning                | 5155 |
| learning_algorithms             | 3916 |
| support_vector_machines         | 2854 |
| classification_(of_information) | 2514 |
| data_mining                     | 1927 |
| neural_networks                 | 1625 |
| feature_extraction              | 1532 |
| support_vector_machine          | 1399 |
| extreme_learning_machine        | 1389 |
| decision_trees                  | 1289 |
| machine_learning_techniques     | 1223 |
| deep_learning                   | 1131 |
| knowledge_acquisition           | 1020 |
| regression_analysis             | 1010 |
| artificial_neural_network       | 925  |
| machine_learning_methods        | 832  |
| procedure                       | 819  |
| supervised_learning             | 801  |
| pattern_recognition             | 793  |
| computer_simulation             | 740  |
| male                            | 728  |
| semantics                       | 684  |
| classifier                      | 682  |
| classification_accuracy         | 637  |
| classification                  | 624  |
| clustering_algorithms           | 603  |
| image_processing                | 599  |
| prediction                      | 581  |



# Embedding classification



- Addestrando il modello di LSTM otteniamo un accuracy del 25%
- Essendo una accuracy piuttosto bassa, abbiamo stampato la matrice delle probabilità in cui la prima riga è la probabilità di appartenenza alla prima classe, la seconda alla seconda classe e così via

```
***CROSS VALIDATED GRID SEARCH***
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed: 11.7min finished
Model: "sequential_3"

Layer (type)                 Output Shape              Param #
=====
embedding_3 (Embedding)      (None, 100, 10)          545310
lstm_6 (LSTM)                 (None, 100, 50)          12200
lstm_7 (LSTM)                 (None, 50)                20200
dense_3 (Dense)              (None, 9)                 459
=====
Total params: 578,169
Trainable params: 32,859
Non-trainable params: 545,310
None
Epoch 1/10
372/372 [=====] - 40s 108ms/step - loss: 2.0488 - accuracy: 0.2524
Epoch 2/10
372/372 [=====] - 41s 111ms/step - loss: 2.0368 - accuracy: 0.2527
Epoch 3/10
372/372 [=====] - 46s 125ms/step - loss: 2.0366 - accuracy: 0.2527
Epoch 4/10
372/372 [=====] - 41s 109ms/step - loss: 2.0359 - accuracy: 0.2527
Epoch 5/10
372/372 [=====] - 40s 108ms/step - loss: 2.0363 - accuracy: 0.2527
Epoch 6/10
372/372 [=====] - 40s 108ms/step - loss: 2.0359 - accuracy: 0.2527
Epoch 7/10
372/372 [=====] - 40s 108ms/step - loss: 2.0358 - accuracy: 0.2527
Epoch 8/10
372/372 [=====] - 41s 109ms/step - loss: 2.0359 - accuracy: 0.2527
Epoch 9/10
372/372 [=====] - 40s 108ms/step - loss: 2.0357 - accuracy: 0.2527
Epoch 10/10
372/372 [=====] - 40s 108ms/step - loss: 2.0355 - accuracy: 0.2527
```

```
1 probability_class_1 = best_model.predict_proba(x_test[0])[0][1]
2 probability_class_2 = best_model.predict_proba(x_test[0])[0][2]
3 probability_class_3 = best_model.predict_proba(x_test[0])[0][3]
4 probability_class_4 = best_model.predict_proba(x_test[0])[0][4]
5 probability_class_5 = best_model.predict_proba(x_test[0])[0][5]
6 probability_class_6 = best_model.predict_proba(x_test[0])[0][6]
7 probability_class_7 = best_model.predict_proba(x_test[0])[0][7]
8 probability_class_8 = best_model.predict_proba(x_test[0])[0][8]
9
10 probability_class_1

WARNING:tensorflow:Model was constructed with shape (None, 100) for input Tensor
array([[0.10225769, 0.10225769, 0.10225769, 0.10225769, 0.10225769,
        0.10225769, 0.1022577 , 0.10225769, 0.10225769, 0.10225769,
        0.10225769, 0.10225769, 0.10225769, 0.10225769, 0.10225769,
        0.10225769, 0.10225769, 0.10225769, 0.10225769, 0.10225769,
```



# Next steps e criticità

- **Criticità:**
- La cardinalità del dataset non è sufficiente per tutte le keyword
- Abbiamo riscontrato alcune difficoltà con la lemmatization di worknet, alcuni verbi -ing non venivano correttamente lemmatizzati
- È stato speso molto tempo in fase di pre-processing su spark
- Creazione di un esteso dizionario di stopwords specifiche per il mondo del machine learning
- **Next steps:**
- Aumentare il numero di abstract nel dataset
- Estendere il numero di stopwords
- XAI



LINK