International Journal of Engineering and Management Sciences (IJEMS) Vol. 6. (2021). No. 3
DOI: 10.21791/IJEMS.2021.3.2.

LSI with Support Vector Machine for Text Categorization – a Practical Example with Python

J. CAVALCANTI[1], J. MENYHART[2]

[1]University of Debrecen, Faculty of Engineering, Department of Engineering Management and Enterprise, jh_gcc@hotmail.com
[2]University of Debrecen, Faculty of Engineering, Department of Air- and Road Vehicles, jozsef.menyhart@eng.unideb.hu

Abstract. Artificial intelligence is becoming a powerful tool of modernity science, there is even a science consensus about how our society is turning to a data-driven society. Machine learning is a branch of Artificial intelligence that has the ability to learn from data and understand its behavers. Python programming language aiming the challenges of this new era is becoming one of the most popular languages for general programming and scientific computing. Keeping all this new era circumstances in mind, this article has as a goal to show one example of how to use one supervised machine learning method, Support Vector Machine, and to predict movie's genre according to its description using the programming language of the moment, python. Firstly, Omdb official API was used to gather data about movies, then tuned Support Vector Machine model for Latent semantic indexing capable of predicting movies genres according to its plot was coded. The performance of the model occurred to be satisfactory considering the small dataset used and the occurrence of movies with hybrid genres. Testing the model with larger dataset and using multi-label classification models were purposed to improve the model.

Keywords: Machine learning, Support Vector Machine, Text Categorization

Introduction

Artificial intelligence is becoming one of the most powerful tools of modernity. Many mathematical models and methods are being used now days to support decision making in business environment. Machine learning is a branch of Artificial intelligence that has the ability to learn from experience, which means more data better the model can predict or classify unseen data. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves, this made data become one of the most important assets of XXI century and machine learning a crucial competitive differential to the companies that apply it to its data.

Following up with the data revolution, programming languages are in the way to develop themselves to support more and more Machine Learning methods. Frequently there are new releases of machine learning and IA libraries, most of them supported by the programming language Python, one of the most popular programming language of the moment, mostly because of its clean coding style, and its flexibility, being capable of being used in pretty much ever developing environment.

DOI: 10.21791/IJEMS.2021.3.2.

Keeping all this new era circumstances in mind, this article has as a purpose to show one example of how to use a supervised machine learning method, Support Vector Machine, and to predict movies genre according to its description using the programming language of the moment, python and its scientific library called Scikit-learn. Support Vector Machine method is frequently used to classify data with different labels, providing a classification prediction of what an unseen data can be.

1. Introduction

Artificial intelligence is becoming one of the most powerful tools of modernity. Many mathematical models and methods are being used now days to support decision making in business environment. Machine learning is a branch of Artificial intelligence that has the ability to learn from experience, which means more data better the model can predict or classify unseen data. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves, this made data become one of the most important assets of XXI century and machine learning a crucial competitive differential to the companies that apply it to its data.

Following up with the data revolution, programming languages are in the way to develop themselves to support more and more Machine Learning methods. Frequently there are new releases of machine learning and IA libraries, most of them supported by the programming language Python, one of the most popular programming language of the moment, mostly because of its clean coding style, and its flexibility, being capable of being used in pretty much ever developing environment.

Keeping all this new era circumstances in mind, this article has as a purpose to show one example of how to use a supervised machine learning method, Support Vector Machine, and to predict movies genre according to its description using the programming language of the moment, python and its scientific library called Scikit-learn. Support Vector Machine method is frequently used to classify data with different labels, providing a classification prediction of what an unseen data can be.

In the first part of the article there is a theoretical explanation about the Python language, Scikit-learn and Support Vector Machine, second part of this article regards about how to use Omdb official API to gather around 3,000 data about movies and with that data code a tuned Support Vector Machine model for Latent semantic indexing, capable of predicting movies genres according to its plot. Finally, in the last section a prediction on an unseen test dataset is done, measuring its performance and predictions capabilities. The performance of the model occurred to be satisfactory considering the small dataset used and the occurrence of movies with hybrid genres. Testing the model with larger dataset and using multi-label classification models was purposed to improve the model.

2. Background Review

Python programming language is becoming one of the most popular languages for general programming and scientific computing, mostly because of to its high-level interactive nature and its diversity of scientific libraries [7]. For that reason, getting knowledge about how it works and how to create outstanding machine learning models with that promising language is really valuable. Python has risen in the Stackoverflow continuously in a good pace for years. In 2019 Stackoverflow survey, Python just

DOI: 10.21791/IJEMS.2021.3.2.

edged out Java in overall ranking, much like it surpassed C# last year and PHP the year before. Python is the fastest-growing programming language of the moment [13].

Machine learning is a study field focused on two questions: How to construct computer systems that automatically improve through experience? That means more data the system has, more efficient the model is. The other question is what are the fundamental statistical-computational-information-theoretic laws that govern all learning systems, including computers, humans, and organizations? Based on those questions all the machine learning methods were created and implemented into our lives [8].

Machine learning algorithms are often categorized as supervised or unsupervised, the main difference between those two categories are the existence or non-existence of labeled data. Supervised machine learning algorithms use labeled data, which means that its real categories are known, to learn from it and create a function that predicts categories from unseen datasets. Opposing supervised learning, unsupervised machine learning uses data that is neither classified nor labeled. Unsupervised learning draws and describe hidden structure from unlabeled data. This article dataset will consist of movie's name, movie's description and movie's genre, therefore supervised learning algorithms should be use to learn from the genre labels and make assumptions about unseen movie's descriptions predicting what genre categories they belong to [3][4].

Python has many scientific libraries and the one chosen to make this machine learning example is Scikit-learn, one of the most used in the data science world. Scikit-learn has a rich environment to provide newest implementations of many well-known machine learning algorithms with an easy and direct syntax, indeed it is an easy-to-use interface tightly integrated with the Python language. Since it relies on the scientific Python ecosystem, Scikit-learn can easily be integrated into other kind of applications that are not build in a traditional scientific, statistical data analysis environment. Indeed, Scikit-learn is an answer to an increasing demand for machine learning by professionals that do not have computer science or statistics technical background [14].

Scikit-learn has a support vector machines package, the method chosen for this practical example. Support vector machines (SVMs) can be defined as group of supervised machine learning methods used for classification, regression and outlier detection. This article only covers support vector machine applied to classification problem (SVC). As shown in the Figure 1 there are different SVC methods supported by Scikit-learn, the objective of this article does not include deeply discussing the slight mathematical differences between the methods. This Article does a SVC model with linear kernel, SVC with RBF kernel and SVC with polynomial kernel and check which of those 3 fits better to the data collected [11].

The machine learning methods are very popular in other engineering field. [19-23] The data collection and optimization methodologies are essential approach of the modern scientific areas and data science. Engineers and scientist try to use more softwares (like Matlab) and special control theory methods to optimize measuring results and datas.[24] [25]

DOI: 10.21791/IJEMS.2021.3.2.

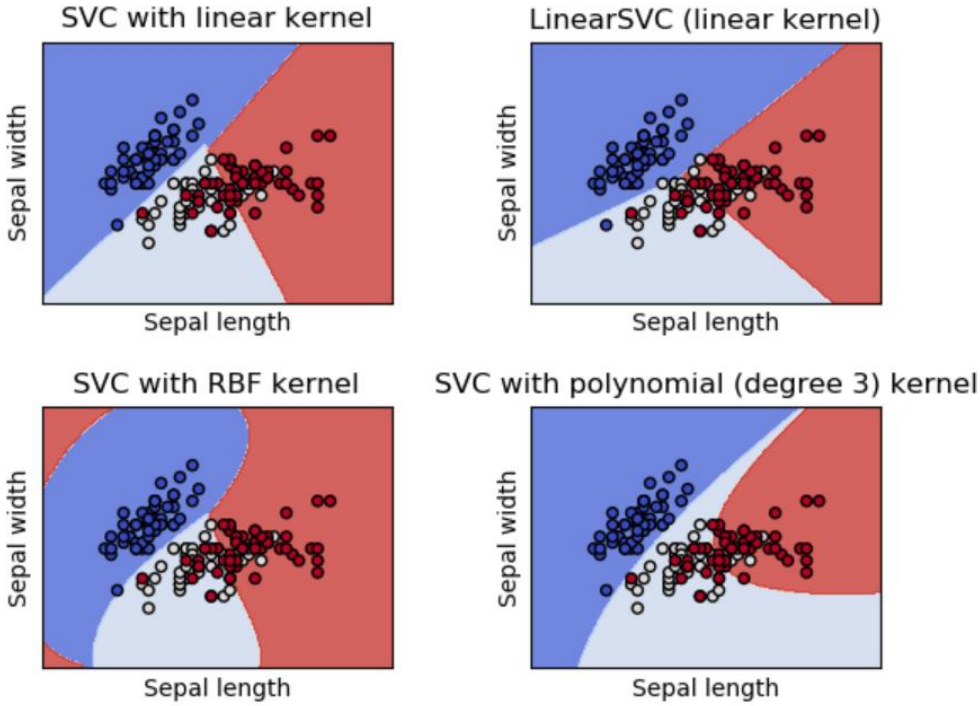

Figure 1. SVC types. Source: https://scikit-learn.org/stable/modules/svm.html#svm

3. Data Collection

First the code needs to get all the data from the Omdb api. This can be done using the python requests library, which allows the user to send get requests to APIs and get its response. Therefore, the algorithm looks for names of movies located in the first column of the first sheet of a excel file and create a URL format according to Omdb api, spaces should be written as %20 according to HTML URL Encoding References [16].

The best approach to export or import data from python to Excel/CSV or from Excel/CSV to python is using the Pandas library. Pandas is easy-to-use data structures and data analysis tool fitting perfectly to these study case requirements. After sending the requests to Omdb api with the proper url, one response is captured. The results captured is only considered if the movie is from one of those kinds; biography, adventure, drama, action, crime, horror, comedy, animation, or documentary, if the movies genre is any different than those the entire response from that movie should be discarded and not considered. The genre filter is needed because there are some specific genres that are not very frequent occurring that the data collected could end up with genres that have few number of quantity, reducing the training set for those specifics genres. One point that should be mentioned is that the movie can have more than one genre, but for this example only its first genre is considered.

Another data cleaning required have to do with the fact that some words from the movie's descriptions do not have describing characteristics, in other words, the presence or non-presence of that word do not have impact in the prediction of its genre. Those kind of words are called stop words. A stop word can be described as a word that has the same likelihood of appearing in any kind of text and are not

DOI: 10.21791/IJEMS.2021.3.2.

relevant to the algorithm. The python library NLTK has a list of default stop words, importing this list to the code and removing this words from the descriptions has a relevant positive impact in the final results [17].

Finishing the text cleaning, the paragraphs "\n", double spacing, triple spacing, equal sign, parentheses, periods, commas and any digits should be removed from the text so it can contain a list a of meaning of words, capable of being transformed in features that describes its genre, replace function of python is good approach to remove the undesired texts, but for the case of the digits replacement, regular expression replacement seems to be the best way of reaching the result desired. Regular Expression are a very powerful way of processing text while looking for recurring patterns, instead of doing 9 replacements with just one regex replacement it is possible to remove all the digits from the text [5].

The JSON file returned by the API should be read and parse to a Python dictionary, that way it becomes possible to manipulate, filter and clean the data in any way it is required to do so. JSON files and Python dictionaries have similar structures, for that reason python has another library, simply called "json", which deals with this data transformation, making the JSON string resulted from the API response become a Python dictionary. Finally, the results containing the movie's description and its genre (only movies with the genres mentioned before) should be sent to a CSV file. Pandas is again useful, since it has a great way to store data frames. Calling the method "DataFrame" of Pandas, it becomes possible to create well-structured tables and later with the method "to_csv" of the "DataFrame" object created, the data can be exported to a csv file. The entire process is shown in the Algorithm 1.

Algorithm 1:

```
import pandas as pd
import requests
import json
import re
import pandas as pd
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
def Generate_list(path): # receives a path of a excel that has a column of movies name and returns a python list
    df=pd.read_excel(path)
    listOfMovies=df.iloc[:,0].tolist()
    return listOfMovies
# initiate empty lists and list of genres movies types that should be considered
typesIncluded=["Biography","Adventure","Drama","Action","Crime","Horror","Comedy","Animation","Documentary"]
docs=[]
genreList=[]
MoviesName=[]
```

DOI: 10.21791/IJEMS.2021.3.2.

```
stop=set(stopwords.words('english')) # get a set of stop words
Movies=Generate_list(path="C:\\Users\\CavalcantiJ\\Desktop\\IA\\moviesNames.xlsx") # generate list from excel file
for movie in Movies: # loop thru each movie
   name=str(movie).replace(" ","%20") #create omdb url replacing spaces for %20
   url= "http://www.omdbapi.com/?t="+name+"&plot=full&apikey=40a283e" #create omdb url for api use
   response=requests.get(url) #send get request to omdb API
   json_data = json.loads(response.text) #load response to python dictionary
   # try to see if the response of the API was successful
   try:
      #check if first genre of movie is in the list of types included and check if there is a movie description available
      if json_data["Genre"].split(",")[0] in typesIncluded and json_data["Plot"]!="N/A":
         #get the movie description
         Content=json_data["Plot"]
         #remove line breaks "\n", remove parenthesis and remove equal sign from description
         Content=Content.replace("\n"," ").replace("(","").replace(")","").replace("=","")
         #remove periods and commas description
         Content=Content.replace(".","").replace(",","")
         Content=re.sub(r'[0-9]+','',Content) #remove all digits from description
         genreList.append(json_data["Genre"].split(",")[0]) # get the movie's first genre and append to list
         MoviesName.append(movie) get the movie's name and append to list
         for word in stop:
            Content=Content.replace(" " +word+" "," ") #remove stop words
         Content=Content.replace("  "," ").replace("  "," ") #remove double spacing and triple spacing from description
         docs.append(Content) #append the cleaned movie's description to a list of descriptions called docs
   #print any url that failed to give an response from the omdb api
   except:
      print(url)
      print("erro")
#create a DataFrame object containing movie's name, genre and descrption
df = pd.DataFrame({'name': MoviesName, "genre":genreList, "content":docs})
#export DataFrame created to a csv file
df.to_csv(r'C:\\Users\\CavalcantiJ\\Desktop\\IA\\data.csv')
```

The results of the algorithm above are all stored in a csv that contains movies names, genres, and description without stop words and without special chars that should not be considered for the

DOI: 10.21791/IJEMS.2021.3.2.

algorithm training, as mentioned before. In other words, the csv has a cleaned dataset ready to be analyzed, the next step is to create the machine learning model that receives this cleaned dataset and returns an artificial intelligence capable of predicting movie's genre from any description provided.

4. Creating the Model

The model creation had as the first step importing all the data generated from the previous step to Python. Again this can be done with Pandas, with the command "read_csv" receiving as parameter the path to the csv that was created by the preceding algorithm. After reading the csv file, for easy use and understanding the data frame columns are transformed into python lists, using the property "loc" of the data frame, to localize each column that should be extracted chained by the method "tolist" which transform the column selected into one python list.

Frequency-inverse document frequency (TF-IDF) is a statistics method which determines weights for each word in each document that in this case are movie's descriptions. Natural language processing (NLP) and text mining often make use of TF-IDFs to determine something similar to a DNA of the text analyzed. In other words, the method measures the importance of the words in the provided text and the importance of the word increases according to the number of times it appears in the document [2].

In python the library Scikit-learn has a function called "TfidfVectorizer" which can transform a list of text into a frequency-inverse document frequency, making it a great tool for use in this article example. "ngram_range" parameter of "TfidfVectorizer" determines how many word's combinations should be taken into consideration, in these example, "ngram_range=(1,2)" Is used to allow the code to consider not just words but pairs of words, hence if there are words that appear together in the text frequently, not just the words individually will be taken into account but also the pairs will be part of the text DNA. It is good to mention that "TfidfVectorizer" have a parameter called "stopwords" which a list of stop words could be assigned resulting in replacements without the need of a manual replacement like shown before, but for better understanding of how the model works, the replacements were done manually in this article [12].

"TfidfVectorizer" returns a sparse matrix as result, sparse matrices refer to the matrices in which most entries of it are zero, that's the case since many of the words in one movie description won't be in the others descriptions, generating a frequency matrix with many zeros values representing those words that are in some of the descriptions but not in the one it refers to. Sparse matrix is more easily compressed, therefore it requires less storage resulting in consequence of that a desirable reduce in computational cost [9].

After creating a sparse matrix with "TfidfVectorizer", the data should be split into test and training datasets. Machine learning datasets often are split into training data and testing data, the training dataset is used for fitting the data to the model, in other words the algorithm creates a function to predict labels of new unknown data. In contrast, the test dataset is used to measure the quality of the machine learning model created by using it as an unknown test dataset and checking the capacity of the model to properly predict correct labels. As the test dataset was not fit to the model, testing the fitted model with it can provide an unbiased analysis of how good the fitted model is [1].

DOI: 10.21791/IJEMS.2021.3.2.

Splitting the data in python can be done with Scikit-learn function called “train_test_split”. The parameters “test_size” and “stratify” of “train_test_split” should be set to 0.2 and “genre” respectively, the first one determines that the size of the testing data is equal to 20 percent of the total dataset and the training data 80 percent consequently. On the other hand, stratify is equal to “genre”, which is a list containing all the genres extracted from the csv file. This stratify configuration makes the data split in a stratified fashion proportionally to the original dataset, that way both test and train datasets will be an exact reduced picture of the main dataset.

In scikit-learn, model optimization can be done in two ways, using “GridSearchCV” or “RandomizedSearchCV”. Both cases take as input an estimator and hyper-parameter settings to search through. The hyper-parameters must be optimized by cross-validated search over hyper-parameter settings. “GridSearchCV” do an exhaustive search over specified parameter values for an estimator looking for the best combination of parameters for the model. “RandomizedSearchCV” is a smarter algorithm that avoids the combinatorial explosion that leads to timeout problems. It gets a random fixed number sample from the parameter distributions chosen and optimizes then with cross validation [6][12]. To keep the model simpler, “GridSearchCV” was used with a list of possible values of each hyper-parameter of the SVC, C and gammas parameter can be 0.001, 0.01, 0.1, 1, 10 or 100, in other hand, kernel can be of 3 different types, 'linear', 'rbf' or 'poly'.

“TruncatedSVD” performs linear dimensionality reduction by means of truncated singular value decomposition (SVD) and it works with sparse matrices efficiently. If data has a large number of features, that’s mostly what happens in texts analyses since there are many words in each text to be analyzed, it can happen that some important feature loses it power being diluted in a large quantity of meaningless features. Hence, “TruncatedSVD” reduces the data into a subset of features that are the most relevant, which results in an improvement to the model’s accuracy [18] [10]. Scikit-learn’s “TruncatedSVD” has one parameter called “n_components”, that determines the size of the reduction, in other words, it tells the dimensionality of the output data, which contains the features that will be considered for the model fitting. For probabilistic latent semantic indexing a value of 100 is recommended [15]

All the steps mentioned before should be executed in the right order, going thru a pipeline of steps, this can be accomplished creating a “make_pipeline” object from Scikit-learn library, followed by its fit, and predict methods on x, y train dataset and x test dataset respectively. Finally, after fitting the model and predicting with x test dataset, the model’s performance was measured using the Scikit-learn classification report to print a text report with the main classification metrics. All the steps mentioned in this section are in the Algorithm 2.

Algorithm 2:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.decomposition import TruncatedSVD
import pandas as pd
from sklearn.model_selection import GridSearchCV
```

DOI: 10.21791/IJEMS.2021.3.2.

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report
df=pd.read_csv(r'C:\\Users\\CavalcantiJ\\Desktop\\IA\\data.csv') #read the data previously generated
names=df.loc["name"].tolist() #transform the DataFrame column "name" into a python list
genre= df.loc["genre"].tolist() #transform the DataFrame column "genre" into a python list
content=df.loc["content"].tolist() #transform the DataFrame column "content" into a python list
def generate_sparse_matrix(docs): #function to generate sparse matrix using TfidfVectorizer
   tfidf = TfidfVectorizer(ngram_range=(1,2))
   csr_mat = tfidf.fit_transform(docs)
   return csr_mat
sparse_matrix = generate_sparse_matrix(content) # generate sparse matrix using function created
X_train, X_test, y_train, y_test = train_test_split(sparse_matrix, genre, test_size=0.2, stratify=genre) #split data
#create GridSearchCV object with set of possible parameters
Cs = [0.001, 0.01, 0.1, 1, 10, 100]
gammas = [0.001, 0.01, 0.1, 1, 10,100]
kernel = ['linear','rbf','poly']
param_grid = {'C': Cs, 'gamma': gammas,'kernel': kernel}
Grids = GridSearchCV(SVC(), param_grid=param_grid)
#create TruncatedSVD object with n_components=100
svd=TruncatedSVD(n_components=100)
#create a pipeline that receives the model steps: TruncatedSVD and GridSearchCV with a SVC
pipeline=make_pipeline(svd,Grids)
pipeline.fit(X_train, y_train) #fit the model
genreResult = pipeline.predict(X_test) #predict the model
print(classification_report(genreResult, y_test )) #print classification report
```

5. Results

The first result of the classification report is shown in the Table 1. It can be observed that Adventure, crime and horror genres could not be predicted properly, in the other hand, documentaries were easy for the model to predict, this effect happened because most of documentaries have only one genre and also have words that are exclusively used for its descriptions, such as the word "documentary" that appears in almost all descriptions, same happens with biographies. The bad performance for adventure, crime and horror happened because most of the movies of this kind have multi-genres, they have common words with other movies descriptions causing confusion to the algorithm, using a multi-label classification model could improve performance of this genders significantly, as well as adding more learning data before fitting the model.

DOI: 10.21791/IJEMS.2021.3.2.

Genre	Precision	Recall	f1-score	Support
Action	0.55	0.44	0.49	101
Adventure	0.04	0.20	0.07	10
Animation	0.26	0.52	0.35	25
Biography	0.66	0.40	0.49	111
Comedy	0.56	0.39	0.46	103
Crime	0.07	1.00	0.13	3
Documentary	0.70	0.64	0.67	90
Drama	0.41	0.29	0.34	100
Horror	0.15	0.50	0.23	14

Table 1. Classification report stratified

While measuring the performance of the model globally, as it is shown in the Table 2, the macro and weighted average performance has relatively satisfactory results, considering the lack of data and the inconsideration of the existence of the multi-genres movies. Global precision was 0.54, recall 0.43, f1-score 0.46 and support 557.

Metric	Precision	Recall	f1-score	Support
macro avg	0.38	0.49	0.36	557
weighted avg	0.54	0.43	0.46	557

Table 2. Classification report

6. Conclusion

There are some premises that should be taken into consideration while analyzing the results, firstly a machine learning model has as one of its definition, a model that learns when more data is applied to the model, for this article the quantity of around 3,000 movie's data was used to fit the model. This is relatively a small quantity when talking about machine learning and data science, considering this, it is understandable that the model does not reach outstanding levels of accuracy. Moreover, as the genres of the movies can be multiple, and the model created can only predict one genre for each movie, it is expected that the learning does not go as smooth as it should, hence reducing the correctness of the model's predictions.

For further studies it is recommended that more tests with different quantities of data are applied to the model, that way it is possible to know how is the learning curve of the model and what is the quantity of data that gives the best trade off regarding running time and accuracy. It is also recommended that Multi-label classification could be tested that way not just the first movies genre would be taken into account by the algorithm.

References

[1] Adi Bronshtein. Train/Test Split and Cross Validation in Python. Understanding Machine Learning (2017).

DOI: 10.21791/IJEMS.2021.3.2.

[2] B. Trstenjak, S. Mikac, D. Donko, KNN with TF-IDF based Framework for Text Categorization, Procedia Engineering, Volume 69, 2014, Pages 1356-1364, ISSN 1877-7058, https://doi.org/10.1016/j.proeng.2014.03.129.

[3] Caron, M., Bojanowski, P., Joulin, A. and Douze, M., 2018. Deep clustering for unsupervised learning of visual features. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 132-149).

[4] Conneau, A., Kiela, D., Schwenk, H., Barrault, L. and Bordes, A., 2017. Supervised learning of universal sentence representations from natural language inference data. arXiv preprint arXiv:1705.02364.

[5] Hunt J. (2019) Regular Expressions in Python. In: Advanced Guide to Python 3 Programming. Undergraduate Topics in Computer Science. Springer, Cham

[6] J. Bergstra and J. Bengio. Random search for hyper-parameter optimization. JMLR, 13:281–305, 2012

[7] K.J. Milmann and M. Avaizis, editors. Scientific Python, volume 11 of Computing in Science & Engineering. IEEE/AIP, March 2011.

[8] Machine learning: Trends, perspectives, and prospects M. I. Jordan1,T. M. Mitchell,Science 17 Jul 2015:Vol. 349, Issue 6245, pp. 255-260 DOI: 10.1126/science.aaa8415).

[9] Otazo, R. , Candès, E. and Sodickson, D. K. (2015), Low-rank plus sparse matrix decomposition for accelerated dynamic MRI with separation of background and dynamic components. Magn. Reson. Med., 73: 1125-1136. doi:10.1002/mrm.25240

[10] Saleem, G., Ahmed, N. and Qamar, U., 2016. TEXT MINING THROUGH LABEL INDUCTION GROUPING ALGORITHM BASED METHOD. Science International, 28 (1).

[11] Scikit-learn.org. (2019). 1.4. Support Vector Machines — scikit-learn 0.21.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/svm.html#svm [Accessed 26 Nov. 2019].

[12] Scikit-learn.org. (2019). sklearn.feature_extraction.text.TfidfVectorizer — scikit-learn 0.21.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html [Accessed 26 Nov. 2019].

[13] Stack Overflow. (2019). Stack Overflow Developer Survey 2019. [online] Available at: https://insights.stackoverflow.com/survey/2019 [Accessed 26 Nov. 2019].

[14] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[15] Scikit-learn.org. (2019). sklearn.decomposition.TruncatedSVD — scikit-learn 0.21.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html [Accessed 28 Nov. 2019].

DOI: 10.21791/IJEMS.2021.3.2.

[16] W3schools.com. (2019). HTML URL Encoding Reference. [online] Available at: https://www.w3schools.com/tags/ref_urlencode.asp [Accessed 26 Nov. 2019].

[17] Wilbur, W. J., & Sirotkin, K. (1992). The automatic identification of stop words. Journal of Information Science, 18(1), 45–55.

[18] Yavuz, M.C., 2019. Analyses of Literary Texts by Using Statistical Inference Methods.

[19] D. Huri, T. Mankovits: Parameter Selection of Local Search Algorithm for Design Optimization of Automotive Rubber Bumper, APPLIED SCIENCES-BASEL 10: 10 pp. 1-16. Paper: 3584 , 16 p. (2020)

[20] D. Huri, T. Mankovits: Automotive rubber part design using machine learning, IOP CONFERENCE SERIES: MATERIALS SCIENCE AND ENGINEERING 659 pp. 1-6. Paper: 012022 , 6 p. (2019)

[21] D. Huri, T. Mankovits: Comparison of the material models in rubber finite element analysis, IOP CONFERENCE SERIES: MATERIALS SCIENCE AND ENGINEERING 393 Paper: 012018 (2018)

[22] I. Kocsis, T. Mankovits: Application of Non-parametric Regression in Engineering Optimization, ANALELE UNIVERSITATII DIN ORADEA FASCIOLA MANAGEMENT SI INGINERIE TEHNOLOGICA / ANNALS OF THE UNIVERSITY OF ORADEA FASCICLE OF MANAGEMENT AND TECHNOLOGICAL ENGINEERING 12: 22 pp. 159-162., 4 p. (2013)

[23] T. Mankovits, T. Szabó: Nemlineáris VEM program gyakorlati alkalmazása gumialkatrészekre, MULTIDISZCIPLINÁRIS TUDOMÁNYOK: A MISKOLCI EGYETEM KÖZLEMÉNYE 2: 1 pp. 103-114., 12 p.(2012)

[24] R. Szabolcsi: Szabályozáselmélet, Budapest, Magyarország : Óbudai Egyetem (2019) , 470 p. ISBN: 9789634491880

[25] R. Szabolcsi: Irányítástechnikai rendszerek tervezése és vizsgálata MATLAB környezetben, Budapest, Magyarország : Óbudai Egyetem (2020) , 396 p., ISBN: 9789634491873