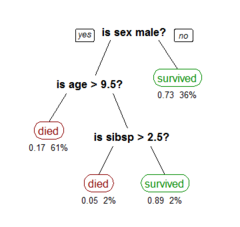
**MACHINE LEARNING – REASEARCH REPORT**

By, Krishna Khurana

1. **Decision Trees**
   1. **Function**

Decision tree learning uses a decision tree to go from observations about an item (branches) to conclusions about the target variable (leaves). It is one of the predictive modelling approach used in statistics , data mining and machine learning .



The goal is to create a model that predicts the value of a target variable based on several input variables

Each leaf represents the possible values of target variables i.e. (died or survived) and the input variables is the path from the root to leaves.

Pruning- The performance of a tree can be enhanced by pruning , it involves removing the branches that make use of features having low importance. In this way we reduce the complexity of the tree and thus increasing its predictive power by reducing overfitting.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| 1. Decision Trees are easy to interpret and explain.  Nonlinear relationships b/w parameters do not affect tree performance. | 1. Trees can be very non-robust i.e. a small change in training data can result in a big change in the tree, and thus a big change in final predictions. |
| 2. No need for variable transformation as the tree structure will remain same.  Also missing values will not prevent splitting of data, and decision trees are not sensitive to outliers since splitting happens based on samples within the split ranges. | 2. They can cause over fitting of data i.e. instead of learning from the training data it just memorizes it (i.e. It produces an analysis that corresponds too closely or exactly to a particular set of data and may therefore fail to fit additional data or predict future observations.) |
| 3. It can handle both numerical and categorical data. |  |

* 1. **Best practises (and use case adoption)**

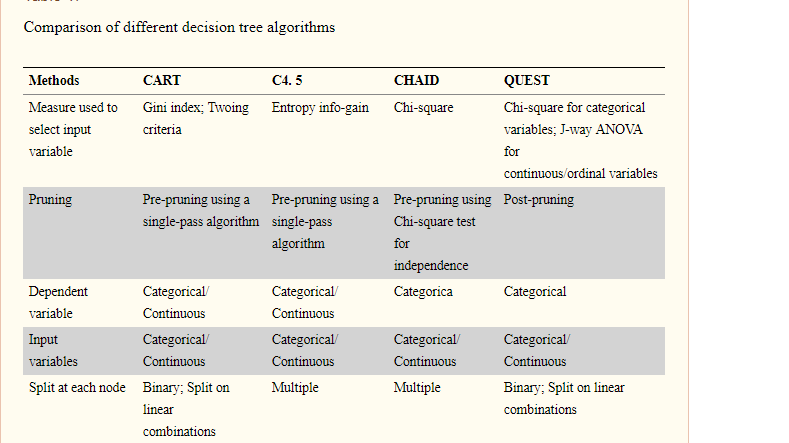
Decision trees is commonly used data mining method for establishing classification systems based on multiple covariates or for developing prediction algorithms for a target variable.

Titanic Dataset for predicting whether a passenger will survive or not (based on 3 features sex, age, sibsp)

* 1. **Available library’s/ tools**

Several statistical algorithms for building decision trees are

CART,C4.5,CHAID(Chi-Squared Automatic Interaction Detection),QUEST(Quick, Unbiased, Efficient, Statistical Tree)



CART

Tree models where the target variables can take discrete values are called **Classification Trees**, here leaves represent tree labels (i.e. which class it belongs to)

Tree models whose target variables can take continuous values are called **Regression Trees**.

1. **Random Forests**
2. **Function**

Random forests build multiple decision trees and merge them together to get a more accurate and stable prediction. The random forest algorithm brings extra randomness into the model. Instead of searching the best features while splitting a node (i.e. while taking a decision) it searches among a random subset of features. So random forests prevents over fitting by selecting a random subset of features and building trees from these subsets.

|  |  |
| --- | --- |
| Advantages | **Disadvantages** |
| 1. RANDOM FOREST IS SO EFFECTIVE … because its “accuracy on Unseen data is upper bounded by both how Accurate each Tree is & how Uncorrelated these are”. | 1. A large a number of trees would make the algorithm slow and ineffective for real world predictions. In general these algorithms are fast to train but slow to predict. |
|  | **2.**  PREDICTION THROUGHPUT   * # of predictions per second of RF < Linear Model (given same resources) * if model needs to predict in real-time e.g. for large websites, this may be problem (throwing more resources should solve it) |

1. **Best practises (and use case adoption)**

Ensembles are a divide-and-conquer approach used to improve performance. The main principle behind ensemble methods is that a group of “weak learners” can come together to form a “strong learner”. Each classifier, individually, is a “weak learner,” while all the classifiers taken together are a “strong learner”.

Random Forest is intrinsically suited for multiclass problems. For a classification problem Random Forest gives you probability of belonging to class. SVM gives you distance to the boundary, you still need to convert it to probability somehow if you need probability.

One application is population structure analysis, which aims to group individuals into subpopulations based on shared genetic variations, such as single nucleotide polymorphisms. RF-derived proximity measure combined with a clustering technique may be well suited for determining the underlying structure of unlabelled data.

[**Paper on RF Unsupervised Learning**](https://biodatamining.biomedcentral.com/articles/10.1186/s13040-017-0156-2)

1. **Available Library and Tools –**

3.2.4.3.1. [sklearn.ensemble](http://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble).RandomForestClassifier

[**Link for Random Forest**](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

1. **SVM**

**a.) Function**

It looks at the extreme of data sets and draws a Decision Boundary also known as hyper plane. To draw the optimal decision boundary only support vector (extreme i.e. shortest distance to closest point) is important whereas other Training examples are ignorable.

In case of non-linear decision boundary we use a kernel trick to transform to higher dimension vector space to make it linear.

Popular Kernels:

1. Polynomial Kernel
2. RBF Kernel
3. Sigmoid Kernel

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| 1.Effective in high dimensional spaces | 1. If # of features > # of samples , it is likely to give poor performance. |
| 2. Also effective if no. of dim > no. of samples | 2. They do not directly provide probability estimates , it’s is calculated using K fold cross validation |
| 3.They use a subset of training points in the decision functions , so its also memory efficient. |  |

**Link for SVM reading-** [**SVM**](https://community.alteryx.com/t5/Data-Science-Blog/Why-use-SVM/ba-p/138440)

**b.)Best practises (and use case adoption)**

In general terms SVMs are very good when you have a huge number of features. For example for text classification in a bag of words model.  
  
SVMs with non-linear kernels perform quite well in most cases and are usually head to head with random forests, sometimes RFs work slightly better and sometimes SVMs win.  
  
SVMs are also particularly useful when you want ordinal classification (ranking), they are widely used in "learning to rank" algorithms.

SVM can be used for classification as well as pattern recognition purpose.

SVMs can be used in medical imaging , another example is SVM based Regression model to study the air quality in urban areas in the city of Oviedo in Spain , also used in Image interpolation , medical classification tasks and in financial industry in time series prediction as well as financial analysis .

It is also used In Pattern Recognition and Page Ranking algorithm and text and object recognition.

**c.)Available Library and Tools –**

1. Scikit-learn SVM implementation based on SMO based LibSVM
2. Calling SVMLight Binaries from Python using subprocess
3. Creating a small implementation for Pegasos (in Python) which solves in the primal form

The IPython notebook doing the comparison can be found at <http://nbviewer.ipython.org/5153583>

**Alternatives of Random Forests**-

* Gradient Boosting Algorithms (XGBoost)-Training and Testing By RF but for prediction we can use any algorithm.

**Gradient Boosting Trees Method**

*Gradient-boosted trees generally perform better than a random forest, although there is a price for that: GBT have a few hyperparametres to tune, while random forest is practically tuning-free.*

* **Boosting** is based on weak learners (high bias, low variance).

In terms of decision trees, weak learners are shallow trees, i.e. (Trees with 2 leaves). Boosting reduces error by reducing bias (and also to some extent variance, by aggregating the output from many models).

GBM is a boosting method, which builds on weak classifiers. The idea is to add a classifier at a time, so that the next classifier is trained to improve the already trained ensemble. Notice that for RF each iteration the classifier is trained independently from the rest.

* On the other hand, **Random Forest** uses fully grown decision trees (low bias, high variance). It tackles the error reduction task by reducing variance. The trees are made uncorrelated to maximize the decrease in variance, but the algorithm cannot reduce bias.

RF uses decision trees, which is prone to over fitting. In order to achieve higher accuracy, RF decides to create large number of trees based on bagging. The basic idea is to resample the data over and over and for each sample train a new classifier. Different classifiers over fit– the data in a different way, and through voting those differences are averaged out.

**Click on the link to learn more ->** [**RF vs. GBM**](https://arxiv.org/pdf/1506.03410.pdf) **.**

[**When to use GBM**](https://www.quora.com/When-would-one-use-Random-Forests-over-Gradient-Boosted-Machines-GBMs)

**RECOMMENDER SYSTEMS**

These are the systems that help select out similar things whenever we select something online. E.g. Netflix, Amazon, Facebook (friends you might wanna know).

How do they work?

There are 2 basic kinds of algorithms for generating recommendations-

* 1. **Content Based Filtering-**It relies upon similarities b/w the items themselves i.e. b/w two movies or 2 songs.

**E.g.** List of movies –INSIDE OUT, Minions, Age Of Ultron,

ANT- MAN.

Suppose someone saw –INSIDE OUT, now we want to know based on what he/she saw out of 3 which he will go to next.

We will generate list of features or quality for all of these movies-

* + - * AnimatedOrNot
      * MarvelOrNot
      * HasASuperVillain
      * PassBechdelTest
      * ParksRecAlumini

We’re going to consider two movies i.e. INISIDE OUT (WHICH I HAVE WATCHED) and any other movie I haven’t watched and by seeing how many features match, So for the movie which has most number of matches with INSIDE OUT, that is the most likely movie that he/she will watch.

So Content Based Filtering systems rely just on the properties of the movies, so I can build the recommender systems just by knowing about the product. **The content based filtering systems rely upon the item in consideration and nothing about any of the user preferences.**

* 1. **Collaborative Filtering –** It relies upon not the quality of the object itself but rather how other users have responded to same object.

So I will take bunch of different users and survey them whether or not like they like a particular movie. So either we can directly survey them or see which movies did they watch.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Inside Out | Minions | Age Of Ultron | Ant-Man |
| **Jason** | **Yes** | **Yes** | **Yes** | **Yes** |
| **Andy** | No | Yes | No | Yes |
| **Sarah** | **Yes** | **No** | **yes** | **No** |
| **Sam** | No | No | Yes | Yes |
| **New User** | **Yes** | ?? | ??**YES** | ?? |

Now suppose we got a new user, so I will ask suppose he liked Inside Out so which other movies is he/she likely to enjoy, based on similar did??

i.e. I will go through/ filter through the list and see those users which also like INSIDE OUT i.e. **the users which matched his/her preference** and for those users I will see which movies were liked by both that movie is likely to be enjoyed by the new user i.e. Age Of Ultron.

**So here I am using the data that was not about the movie itself but data about other users’ preferences.**

So more data we collect better will be the recommendations.

* 1. Hybrid Recommender Systems

[Python Code for Recommender Systems](https://www.youtube.com/watch?v=39vJRxIPSxw)

**Libraries Available-**Apache Mahout (It combines 3 machine learning algorithms Collaborative Filtering, Clustering Algorithms and Classification algorithms)

[Github Link for ML training models like Google, Azure ML](https://github.com/grahamjenson/list_of_recommender_systems)

## Software as a Service Recommender Systems

The benefits to using a SaaS recommender system is that you can pay for value with a low overhead rather than having a large upfront investment, they generally have a clear integration path for you to use, and they provide continual development and improvement while you use it.

The SaaS recommender systems are:

* Google Cloud Prediction API - [Google Cloud ML API](https://cloud.google.com/ml-engine/) – It offers training and prediction services, which can be used together or individually.
* Parallel Dots – A text classifier that may serve our purpose

[Text Classifier](https://www.paralleldots.com/custom-classifier)

* Amazon Machine Learning – From Building the model, to training it and then finally deploying it , it also makes the predefined ML algorithms 10X faster

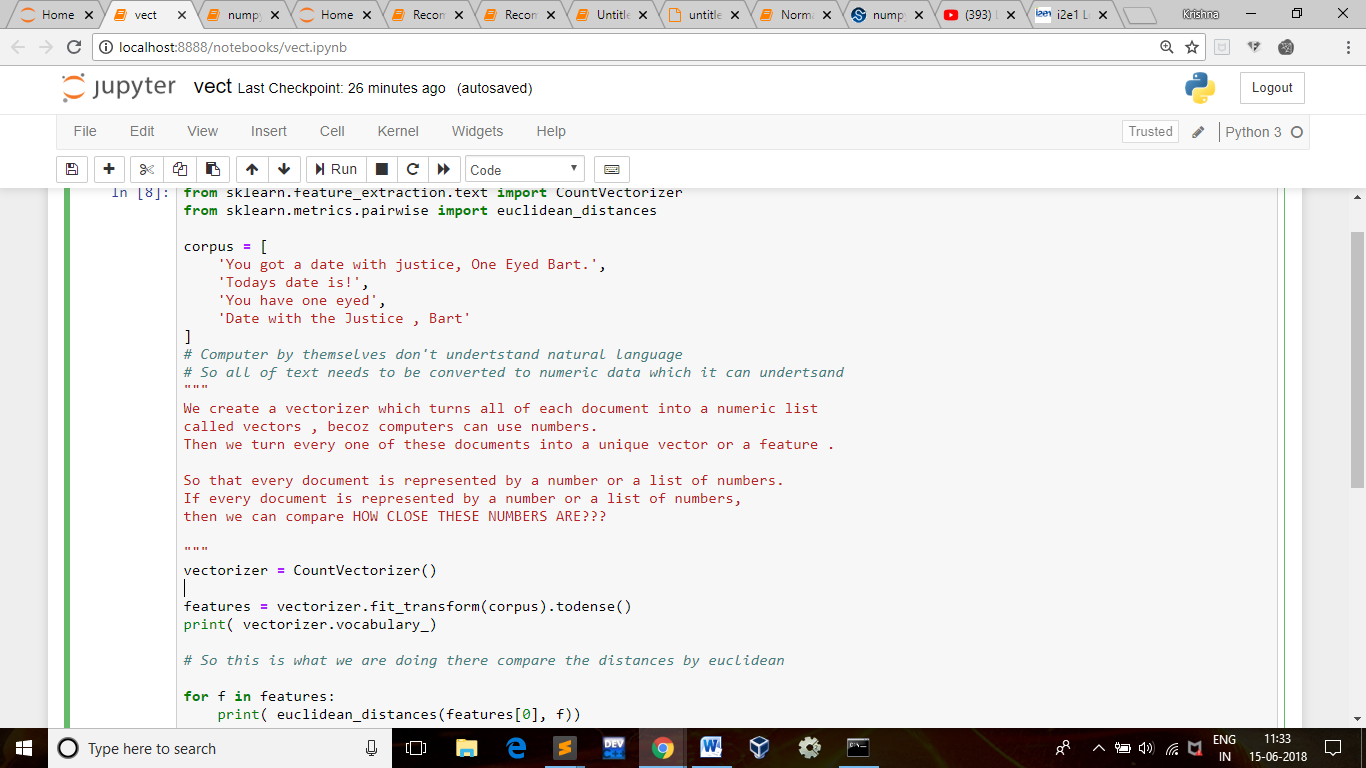
[Amazon SageMaker](https://aws.amazon.com/sagemaker/features/)

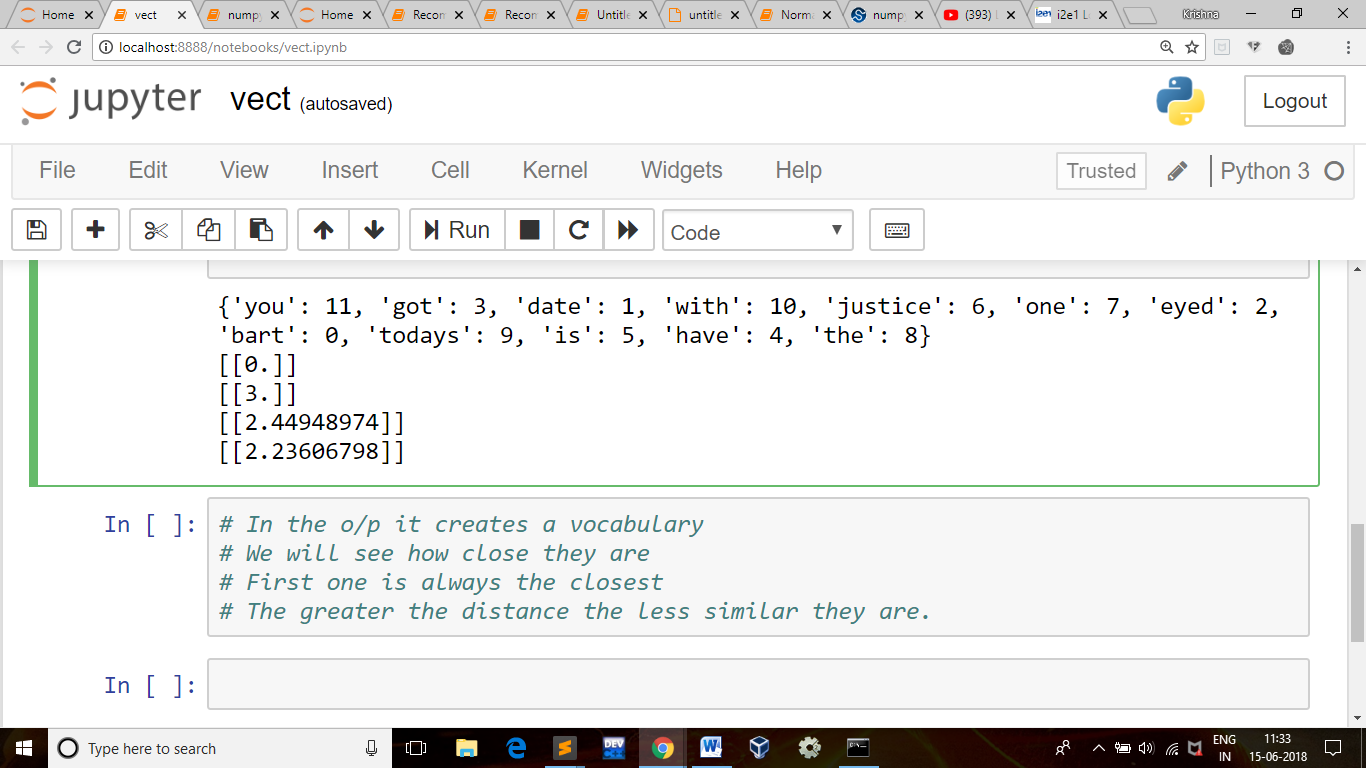
* Microsoft Azure ML machine learning platform to model data and create predictions. [Azure Text Classifier](https://microsoft.github.io/ml-server-text-classification/visualize.html)
* IBM Watson - [Watson Discovery](https://www.ibm.com/watson/services/discovery/)

Incident Accuracy Percentage

Some Links

1. [Incident matching](https://datascience.stackexchange.com/questions/23969/sentence-similarity-prediction)
2. [Super-Fast String Matching in Python](https://bergvca.github.io/2017/10/14/super-fast-string-matching.html)
3. [Python algorithms for text matching](http://stackabuse.com/levenshtein-distance-and-text-similarity-in-python/)





NLP with text matching

1. Count Vectorizer
2. tfidf
3. [NLP Naive Bayes](https://medium.com/syncedreview/applying-multinomial-naive-bayes-to-nlp-problems-a-practical-explanation-4f5271768ebf)
4. gensim model

sklearn.feature\_extraction.text.TfidfVectorizer

**Tf-idf term weighting:**

Tf-idf(t,d) = tf(t,d) X idf(t)

* **term frequency**tf(*t*,*d*), the simplest choice is to use the *raw count* of a term in a document, i.e. the number of times that term *t* occurs in document *d*. If we denote the raw count by *ft*,*d*, then the simplest tf scheme is tf(*t*,*d*) = *ft*,*d*.

tf(t,d) = 0.5 + 0.5 \*( ft,d)/(max{ft’,d : t’ E d })

* **inverse document-frequency**

\text{idf}(t) = log{\frac{1 + n_d}{1+\text{df}(d,t)}} + 1

where n_d is the total number of documents, and \text{df}(d,t) is the number of documents that contain term t.

TfidfVectorizer(input=’content’, encoding=’utf-8’, decode\_error=’strict’, strip\_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer=’word’, stop\_words=None, token\_pattern=’(?u)\b\w\w+\b’, ngram\_range=(1, 1), max\_df=1.0, min\_df=1, max\_features=None, vocabulary=None, binary=False, dtype=<class ‘numpy.int64’>, norm=’l2’, use\_idf=True, smooth\_idf=True, sublinear\_tf=False)

Parametres:

Input: string {‘filename’, ‘file’, ‘content’}

If ‘filename’, the sequence passed as an argument to fit is expected to be a list of filenames that need reading to fetch the raw content to analyze.

If ‘file’, the sequence items must have a ‘read’ method (file-like object) that is called to fetch the bytes in memory.

Otherwise the input is expected to be the sequence strings or bytes items are expected to be analyzed directly.

**encoding** : string, ‘utf-8’ by default.

If bytes or files are given to analyze, this encoding is used to decode.

**decode\_error** : {‘strict’, ‘ignore’, ‘replace’}

Instruction on what to do if a byte sequence is given to analyze that contains characters not of the given encoding. By default, it is ‘strict’, meaning that a UnicodeDecodeError will be raised. Other values are ‘ignore’ and ‘replace’.

**strip\_accents** : {‘ascii’, ‘unicode’, None}

Remove accents during the preprocessing step. ‘ascii’ is a fast method that only works on characters that have an direct ASCII mapping. ‘unicode’ is a slightly slower method that works on any characters. None (default) does nothing.

**analyzer** : string, {‘word’, ‘char’} or callable

Whether the feature should be made of word or character n-grams.

If a callable is passed it is used to extract the sequence of features out of the raw, unprocessed input.

**preprocessor** : callable or None (default)

Override the preprocessing (string transformation) stage while preserving the tokenizing and n-grams generation steps.

**tokenizer** : callable or None (default)

Override the string tokenization step while preserving the preprocessing and n-grams generation steps. Only applies if analyzer == 'word'.

**ngram\_range** : tuple (min\_n, max\_n)

The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min\_n <= n <= max\_n will be used.

**stop\_words** : string {‘english’}, list, or None (default)

If a string, it is passed to \_check\_stop\_list and the appropriate stop list is returned. ‘english’ is currently the only supported string value.

If a list, that list is assumed to contain stop words, all of which will be removed from the resulting tokens. Only applies if analyzer == 'word'.

If None, no stop words will be used. max\_df can be set to a value in the range [0.7, 1.0) to automatically detect and filter stop words based on intra corpus document frequency of terms.

**lowercase** : boolean, default True

Convert all characters to lowercase before tokenizing.

**token\_pattern** : string

Regular expression denoting what constitutes a “token”, only used if analyzer == 'word'. The default regexp selects tokens of 2 or more alphanumeric characters (punctuation is completely ignored and always treated as a token separator).

**max\_df** : float in range [0.0, 1.0] or int, default=1.0

When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

**min\_df** : float in range [0.0, 1.0] or int, default=1

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

**max\_features** : int or None, default=None

If not None, build a vocabulary that only consider the top max\_features ordered by term frequency across the corpus.

This parameter is ignored if vocabulary is not None.

**vocabulary** : Mapping or iterable, optional

Either a Mapping (e.g., a dict) where keys are terms and values are indices in the feature matrix, or an iterable over terms. If not given, a vocabulary is determined from the input documents.

**binary** : boolean, default=False

If True, all non-zero term counts are set to 1. This does not mean outputs will have only 0/1 values, only that the tf term in tf-idf is binary. (Set idf and normalization to False to get 0/1 outputs.)

**dtype** : type, optional

Type of the matrix returned by fit\_transform() or transform().

**norm** : ‘l1’, ‘l2’ or None, optional

Norm used to normalize term vectors. None for no normalization.

**use\_idf** : boolean, default=True

Enable inverse-document-frequency reweighting.

**smooth\_idf** : boolean, default=True

Smooth idf weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. Prevents zero divisions.

**sublinear\_tf** : boolean, default=False

Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).