HW7

Compare MNB and SVMs

Bing-Je Wu

IST 736 – Text Mining

Table of Contents

[Introduction 2](#_Toc40519569)

[Analysis and Models 2](#_Toc40519570)

[About the Data 3](#_Toc40519571)

[Data Processing 3](#_Toc40519572)

[Models 7](#_Toc40519573)

[Results 21](#_Toc40519574)

[Conclusion 22](#_Toc40519575)

[Reference 22](#_Toc40519576)

# Introduction

Movies, also called motion picture, play a crucial part in daily life, especially in the 20 century. Everyone loves watching movies. Movies are full over the places. People can rent a movie from the supermarket such as Walmart, Shoprite…etc. And with the improvement of the technology, the way to get the access to a movie has become easier and simpler. People can watch a movie through internet. Many companies, such as Netflix, Hulu, Amazon, Disney and Apple, has provided the streaming services for people to watch movies at home without going out to pick up a DVD or the old fashion of VHS. With that being said, it comes to another issue. Since people get to watch a movie easily, the volume of comments, reviews or critics about a movie has also been increasing exponentially this day. Having the ability to decipher the thought of the movie review becomes one of the major tasks in text mining. This is why more and more applications are built to solve the question.

Analysis and Models  
Three algorithms, ’MultinomialNB’ (Multinomial Naive Bayes) algorithm and ‘LinearSVC ‘ (Linear Support Vector) and ‘SVC’ (Support Vector with kernels) from scikit learn package, were used to build models. Two transformers, CountVectorizer and TfidfVectorizer, were utilized for model building. Holdout test technique (train\_test\_split) and k-fold cross-validation approach (cross\_val\_score) were implemented for model selection. A pipeline (make\_pipeline) was built to get a smooth workflow.

An advanced feature engineering technique, combining the document term matrix and manual features, was also included for having more options on model selection. The technique utilized the ‘transformer’ class (FunctionTransformer) that is provided by sklearn library. The concept of the technique is to create a union (make\_union) with parallel transformers, a sequence of a pipeline (vectorizing text after a text feature transformer) and manual feature transformer. The union can be combined with an algorithm to build a pipeline in high level for model training and model evaluation. In this case, the recurrent preprocessing steps for building a model are vastly reduced. Figure 1 shows the example of Pipeline and FeatureUnion. Figure 2 exhibits the workflow of the data transformation in text with manual features.

**Figure 1**. *Transformer Example*

A picture containing photo, sitting, black, dark

Description automatically generated

**Figure 2.** *Transformation Workflow*

A picture containing dark

Description automatically generated

## About the Data

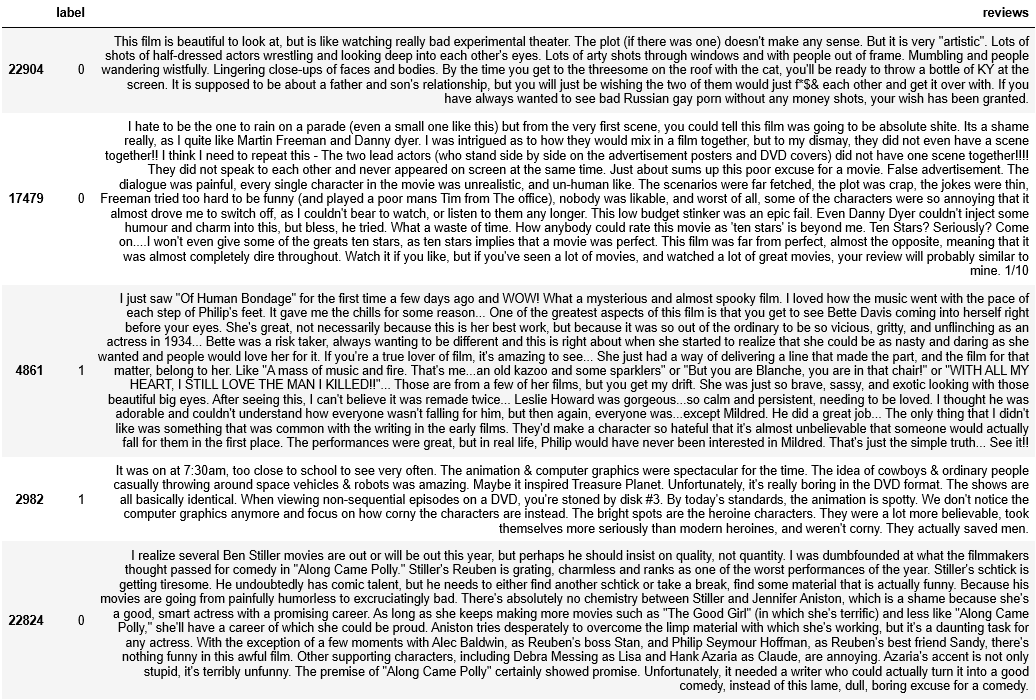
Data were sampled from Large Movie Review Dataset (<https://ai.stanford.edu/~amaas/data/sentiment/> ) as a part of work on course project. The movie reviews were originally collected from IMDB, an online database of information related to films, television programs. Any review with the stars rating greater than or equal to 7 is labeled as positive review. If the rating is less than or equal to 4, it is labeled as negative review. The reviews with stars rating between 5 stars and 6 stars were left out. The sampled dataset contains 25,000 movie reviews in total. 125,000 reviews were labeled as negative review and the other 125,000 reviews were labeled as positive review.

**Table 1.** *Dataset*

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Data Type |
| label | Label of sentiment (‘0’ : negative; ‘1’: positive) | String |
| review | The text of movie reviews | String |

Data Processing  
Data were cleaned and transform into a CSV file. Some feature engineering techniques were implemented for the advanced approach, helping classification task along with document term matrix. 10 extra features were added. A ‘str\_length’ column was created by using ‘len()’ built-in function applied on the ‘reviews’ column. The ‘n\_tokens’ column was added with two functions, ‘len()’ and ‘split()’ on the ‘reviews ‘column. The other columns, such as ‘exclamat’, ‘question’, ‘plus’, ‘period’ and ‘minus’, were created to count the number of the special marks appeared in the review content. Two list, positive words and negative words (<https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html> ), were used to create two columns, ‘n\_pos’ column and ‘n\_neg’’, to count the number of positive words and the number of negative words appeared in the review content.

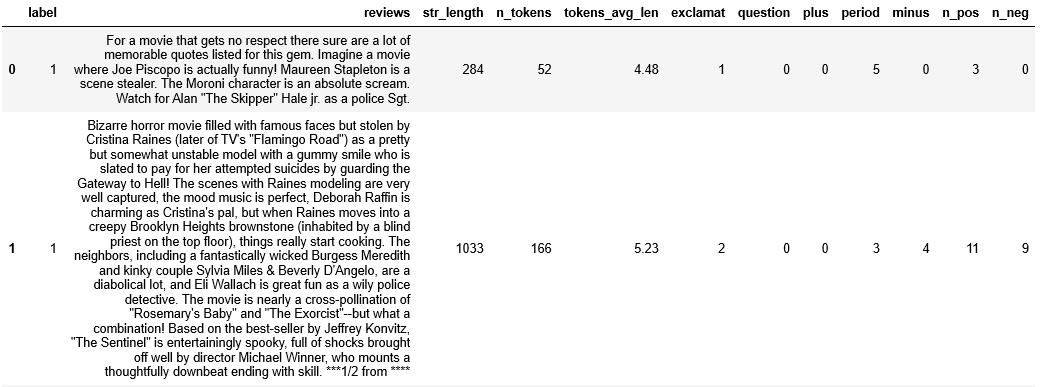
**Figure 3**. *Sample of data before data pre-processing*

**

**Table 2.** *User defined functions for data pre-processing*

|  |  |  |
| --- | --- | --- |
| Function | Type | Explain |
| N\_length | lambda | Count the length of the each of document (review) |
| n\_tokens | lambda | Count the number of words after applying ‘split()’ on each of document |
| exclamat | lambda | Count the number of ‘!’ in each of document (review) |
| question | lambda | Count the number of ‘?’ in each of document (review) |
| plus | lambda | Count the number of ‘+’ in each of document (review) |
| period | lambda | Count the number of ‘.’ in each of document (review) |
| minus | lambda | Count the number of ‘-’ in each of document (review) |
| n\_pos | lambda | Count the number of positive words used in each of document (review) |
| n\_neg | lambda | Count the number of negative words used in each of document (review) |

**Figure 4**. *Sample of cleaned data after data pre-processing*



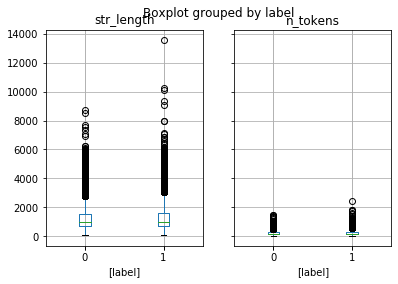
**Figure 5.** *Data output distribution*

A screenshot of a cell phone

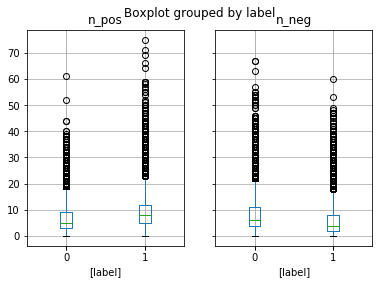
Description automatically generated

After finishing the data pre-processing, an initial data exploratory analysis was conducted to find the useful features for classification task. Figure 5 and Figure 6 show the 'str\_length' column, 'n\_tokens' column, ‘n\_pos’ column and ‘n\_neg’ column have some distinction between negative review and positive review. Especially ‘n\_pos’ column and ‘n\_neg’, they have less overlaps comparing ‘str\_length' column and ‘n\_tokens' column. Four columns will be used as the manual features for the advanced modeling building approach.

**Figure 6.** *Boxplot for 'str\_length' column and ‘n\_tokens' column*



**Figure 7.** *Boxplot for ‘n\_pos’ column and ‘n\_neg’ column*



Models  
The prediction task here is sentiment prediction, predicting a review is a positive or negative review. Naïve Bayes models were built for those tasks by implementing the ‘multinomialNB’ algorithm from sklearn. The ‘LinearSVC ‘ (Linear Support Vector) and ‘SVC’ (Support Vector with kernels) algorithms were also used to build models. All the algorithms were implemented with two transformers, CountVectorizer and TfidfVectorizer. Because the dataset is a huge dataset, it tends to have many tokens when a vectorizer transformer is applied. Both vectorizers were set with the base parameter setting, min\_df=5, stop\_words='english', to eliminate some unnecessary words as feature and to reduce the computation cost dering model building process. The advanced approach was applied with desired features along with document term matrix to build another series of models by using ‘make\_prediction\_manual\_ft’ function to compare the metrics.

**Table 3**. *User-defined functions for model building*

|  |  |
| --- | --- |
| Function | Explanation |
| make prediction | The function takes four input arguments, ‘X’, ‘y’, ‘vect’ and ‘clf’. The ‘X’ argument takes a pandas series that include text. ‘y’ is the label of the data. ‘vect’ is the object name of a vectorizer. ‘clf’ is the name of a classifier. The workflow of the function is as below:   1. Apply train\_test\_split for holdout test; X\_train, X\_test, y\_train, y\_test 2. Create a pipeline; pipe = vectorizer + classifier 3. Use Cross-Validation on pipe with X\_train 4. Validate with X\_test using ‘predict’ method on pipeline 5. Plot and show the information of top 10 and bottom 10 features 6. Create confusion matrix and classification report |
| get\_manual | The function takes a pandas dataframe as an input. It returns a pandas dataframe that is a subset of the input. The filter condition is based on the feature that is desired to be used in model training process. |
| get\_text | The function takes a pandas dataframe as an input. It returns a pandas series that contains text as the input. |
| make\_prediction\_manual\_ft | The function takes four input arguments, ‘X’, ‘y’, ‘vect’ and ‘clf’. The ‘X’ argument takes a pandas series that include text. ‘y’ is the label of the data. ‘vect’ is the object name of a vectorizer. ‘clf’ is the name of a classifier. The workflow of the function is as below:   1. Apply train\_test\_split for holdout test; X\_train, X\_test, y\_train, y\_test 2. Create a pipeline; pipe = [(get\_text\_ft + vectorizer) + get\_manual\_ft]+ clf 3. Use Cross-Validation on pipe with X\_train 4. Validate with X\_test using ‘predict’ method on pipeline 5. Create confusion matrix and classification report |

**Table 4**. *Models performance comparison for Naïve Bayes– Part 1*

|  |  |  |
| --- | --- | --- |
| *Model* | NB model 1 | NB model 2 |
| Setting | MultinomialNB() CountVectorizer(min\_df=5, stop\_words='english') | MultinomialNB() CountVectorizer(min\_df=5, binary=True, stop\_words='english') |
| Training | 85.15% | 85.34% |
| Testing | 85.04% | 85.11% |
| Features | 22716 | 22716 |
| Run Time | 23.3 s | 25.8 s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

**Table 5**. *Models performance comparison for Naïve Bayes– Part 2*

|  |  |
| --- | --- |
| *Model* | NB model 3 |
| Setting | MultinomialNB() TfidfVectorizer(min\_df=5, stop\_words='english') |
| Training | 85.81% |
| Testing | 85.75% |
| Features | 22716 |
| Run Time | 25 s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated |
| Report | A screenshot of a cell phone  Description automatically generated |

**Table 6**. *Models performance comparison for linearSVC – Part 1*

|  |  |  |
| --- | --- | --- |
| *Model* | linearSVC model 1 | linearSVC model 2 |
| Setting | LinearSVC(C=10) CountVectorizer(min\_df=5, stop\_words='english') | LinearSVC(C=1) CountVectorizer(min\_df=5, stop\_words='english') |
| Training | 84.85% | 85.27% |
| Testing | 84.64% | 85.07% |
| Features | 22716 | 22716 |
| Run Time | 32.4 s | 34.7 s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A picture containing bird  Description automatically generated | A picture containing bird  Description automatically generated |

**Table 7**. *Models performance comparison for linearSVC – Part 2*

|  |  |  |
| --- | --- | --- |
| *Model* | linearSVC model 3 | linearSVC model 4 |
| Setting | LinearSVC(C=10) CountVectorizer(min\_df=5, binary=True, stop\_words='english') | LinearSVC(C=1) CountVectorizer(min\_df=5, binary=True, stop\_words='english') |
| Training | 84.35% | 84.67% |
| Testing | 84.37% | 84.79% |
| Features | 22716 | 22716 |
| Run Time | 33.2 s | 33.4 s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A picture containing bird  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

**Table 8**. *Models performance comparison for linearSVC – Part 3*

|  |  |  |
| --- | --- | --- |
| *Model* | linearSVC model 5 | linearSVC model 6 |
| Setting | LinearSVC(C=10) CountVectorizer(min\_df=5, stop\_words='english', ngram\_range=(1, 2)) | LinearSVC(C=1) CountVectorizer(min\_df=5, stop\_words='english', ngram\_range=(1, 2)) |
| Training | 86.54% | 86.66% |
| Testing | 86.51% | 86.51% |
| Features | 56191 | 56191 |
| Run Time | 1min 17s | 1min 18s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A picture containing bird  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

**Table 9**. *Models performance comparison for linearSVC – Part 4*

|  |  |  |
| --- | --- | --- |
| *Model* | linearSVC model 7 | linearSVC model 8 |
| Setting | LinearSVC(C=10) CountVectorizer(min\_df=5, stop\_words='english', ngram\_range=(1, 3)) | LinearSVC(C=1) CountVectorizer(min\_df=5, stop\_words='english', ngram\_range=(1, 3)) |
| Training | 86.51% | 86.58% |
| Testing | 86.55% | 86.64% |
| Features | 58797 | 58797 |
| Run Time | 2min 28s | 2min 9s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A picture containing bird  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

**Table 10**. *Models performance comparison for SVC – Part 1*

|  |  |  |
| --- | --- | --- |
| *Model* | SVC model 1 | SVC model 2 |
| Setting | SVC(C = 10, kernel='rbf') CountVectorizer(min\_df=5, stop\_words='english') | SVC(C = 1, kernel='rbf') CountVectorizer(min\_df=5, stop\_words='english') |
| Training | 86.82% | 86.36% |
| Testing | 86.92% | 86.45% |
| Features | 22716 | 22716 |
| Run Time | 47min 5s | 28min 10s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A picture containing bird  Description automatically generated | A picture containing bird  Description automatically generated |

**Table 11**. *Models performance comparison for SVC – Part 2*

|  |  |  |
| --- | --- | --- |
| *Model* | SVC model 3 | SVC model 4 |
| Setting | SVC(C=10, kernel='poly') CountVectorizer(min\_df=5, stop\_words='english') | SVC(C=1, kernel='poly') CountVectorizer(min\_df=5, stop\_words='english') |
| Training | 79.12% | 80.07% |
| Testing | 79.05% | 83.03% |
| Features | 22716 | 22716 |
| Run Time | 45min 47s | 23min 34s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A screenshot of a cell phone  Description automatically generated | A picture containing bird  Description automatically generated |

**Table 12**. *Models performance comparison for SVC – Part 3*

|  |  |  |
| --- | --- | --- |
| *Model* | SVC model 5 | SVC model 6 |
| Setting | SVC(C=10, kernel='sigmoid') CountVectorizer(min\_df=5, stop\_words='english') | SVC(C=1, kernel='sigmoid') CountVectorizer(min\_df=5, stop\_words='english') |
| Training | 71.97% | 74.75% |
| Testing | 73.92% | 74.95% |
| Features | 22716 | 22716 |
| Run Time | 7min 51s | 9min 53s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

**Table 13**. *Models performance comparison for SVC – Part 4*

|  |  |  |
| --- | --- | --- |
| *Model* | SVC model 7 | SVC model 8 |
| Setting | SVC(C = 10, kernel='rbf') TfidfVectorizer(min\_df=5, stop\_words='english') | SVC(C = 1, kernel='rbf') TfidfVectorizer(min\_df=5, stop\_words='english') |
| Training | 85.81% | 85.81% |
| Testing | 85.75% | 85.75% |
| Features | 22716 | 22716 |
| Run Time | 21.9 s | 22.1 s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A screenshot of a cell phone  Description automatically generated | A picture containing bird  Description automatically generated |

**Table 14**. *Models performance comparison for SVC – Part 5*

|  |  |  |
| --- | --- | --- |
| *Model* | SVC model 9 | SVC model 10 |
| Setting | SVC(C = 10, kernel='poly') TfidfVectorizer(min\_df=5, stop\_words='english') | SVC(C = 1, kernel='poly') TfidfVectorizer(min\_df=5, stop\_words='english') |
| Training | 85.81% | 85.81% |
| Testing | 85.75% | 85.75% |
| Features | 22716 | 22716 |
| Run Time | 22.3 s | 22.1 s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A picture containing bird  Description automatically generated | A picture containing bird  Description automatically generated |

**Table 15**. *Models performance comparison for SVC – Part 6*

|  |  |  |
| --- | --- | --- |
| *Model* | SVC model 11 | SVC model 12 |
| Setting | SVC(C = 10, kernel='sigmoid') TfidfVectorizer(min\_df=5, stop\_words='english') | SVC(C = 1, kernel='sigmoid') TfidfVectorizer(min\_df=5, stop\_words='english') |
| Training | 85.81% | 85.81% |
| Testing | 85.75% | 85.75% |
| Features | 22716 | 22716 |
| Run Time | 22.5 s | 23.1 s |
| Confusion Matrix | A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Report | A picture containing bird  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

**Table 16**. *Models performance for manual features*

|  |  |  |
| --- | --- | --- |
| *Model* | Manual feature model 1 | Manual feature model 2 |
| Setting | LinearSVC(C=1) CountVectorizer(min\_df=5, stop\_words='english', ngram\_range=(1, 3)) | LinearSVC(C=1) CountVectorizer(min\_df=5, stop\_words='english', ngram\_range=(1, 3)) |
| Manual feature | 'n\_pos' ,'n\_neg' | 'n\_pos' ,'n\_neg', 'str\_length', 'n\_tokens' |
| Training | 86.9% | 78.36% |
| Testing | 86.61% | 85.73% |
| Features | 22718 | 22720 |
| Run Time | 2min 20s | 2min 49s |
| Confusion Matrix |  |  |
| Report |  |  |

# Results

From Table 4 to Table 5, it shows that NB model 3 has the best performance of 85.75% accuracy rate among all Naïve Bayes models. From Table 6 to Table 9, it states that the ‘linearSVC model 8’ is the best model with the performance of 86.64% accuracy rate and acceptable computation cost. From Table 10 to Table 12, it shows the model performance of SVM models with different kernels. Although the ‘SVC model 1’ has the highest accuracy rate, 86.92%, it is not the best model because of the high computation cost. From Table 13 to Table 15, it is easy to see that Tfidf transformer has the same performance with different SVM kernels. Those models have the accuracy rate at 85.75%.

Traditional model building and advanced model building technique were both implemented. From Table 9, the best model from the traditional method (linearSVC model 8) has the accuracy rate at 86.64%. From Table 16, the advanced approach with two extra columns, ‘n\_pos’ and ‘n\_neg’, has the performance at 86.61%. Although both models have very close accuracy rate, the traditional model is still the preferred model since it is simpler than advanced model.

The top 10 and bottom 10 features words were also show as the table below. Both Naïve Bayes and SVM have their own set of top and bottom feature words. The top 10 words are the indicative words for the most positive category and the bottom 10 words are the indicative words for the most negative category. Both models showed different results. However, the result from the SVM model seems to make more sense since it has some subjective words in both top 10 list and bottom 10 list. The result from the Naïve Bayes has less sense in this case.

**Table 14**. *Top 10 and bottom 10 feature words for Naïve Bayes and SVN*

|  |  |  |
| --- | --- | --- |
|  | Naïve Bayes | SVM |
| Setting | MultinomialNB( ) CountVectorizer(min\_df=5, stop\_words='english') | LinearSVC(C=10) CountVectorizer(min\_df=5, stop\_words='english') |
| Result |  |  |

# Conclusion

Deciphering text content could be hard. Text are everywhere around the world. And, needless to say, extracting the information from tons of movie reviews requires a lot of works. If the tasks are performed manually, it could cost badly in time and money.

However, if the technique is well developed, many can get benefits from it. For example, production companies can make a movie or tv series based on the trend of the popularity on the movie reviews. Investor can decide to make a sequel, or a spinoff based on the level of the discussion on the review platform.

The same technique can be applied not only on the movie review but also other industries, such as restaurant review, product review…etc. This technique can provide different marketing strategy from the traditional method. Because the spread of the internet, companies in the modern age need to adapt the new concept and take advantage of it in order to keep growing.

# Reference

SVC, from scikit-learn [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) ; 2 LinearSVC, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC)  
3 CountVectorizer, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) ; 4 TfidfVectorizer, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)  
5 FeatureUnion, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.FeatureUnion.html) ; 6 make\_union, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_union.html)  
7 FunctionTransformer, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.FunctionTransformer.html?highlight=transformer#sklearn.preprocessing.FunctionTransformer) ; 8 cross\_val\_score, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html?highlight=cross_val#sklearn.model_selection.cross_val_score)  
9 Pipeline, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline) ; 10 make\_pipeline, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_pipeline.html?highlight=make_pipeline#sklearn.pipeline.make_pipeline)  
11 MultinomialNB, from scikit-learn. [Web Link](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html)