University Jean Monnet

Project

Data Mining for Big Data

Authors:
Tu-My Doan
Sali Dauda Mohamnmed

Academic Advisors:
Prof. Baptiste Jeudy
Prof. Christine Largeron

January, 2019

Contents

Abstract

The main objective of this project is to achieve scientific analysis on the financial news dataset using machine learning and data mining techniques. We will first analyze the dataset to understand it more and then apply several machine learning algorithms in order to fulfil the classification task which is to predict the labels for the news articles.

1 Introduction

Our company IT Corp. provides various IT related solutions to customers in different areas. One of our customers has approached us to ask for help in order to improve their business through decision making process for financial news articles provided by Reuters news agency. Our task is to develop for them a prediction system that can help categorize the news articles which is currently done manually. In this case they are only interested in the financial earnings of the companies so they can make useful plans for investment. The task has been assigned to us - data scientists of the company.

The following parts of the report will come in this order: first, we will explain about our data, do some before-hand analysis then in the methodology we will apply various machine learning techniques that are used to complete the prediction task along with their results. Finally, we will provide our conclusion and suggestions for the customer in order to pick the right model for their benefits.

All of the implementations in this report are available in the Jupyter Notebook files for easier understanding.

2 Data Understanding

2.1 Items counts

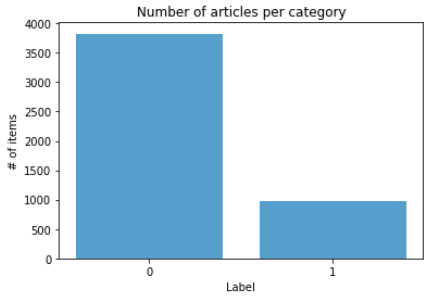

Figure 1: Item counts per category

The dataset comes in a form of XML files and there are 5,000 news articles where only 4,800 of them have labels which will be used as the training set and for the test set we have 200 remaining items.

To understand data more, we did some analysis and we can easily see that our data for the training set is imbalanced at ratio of 80:20 for labels 0 and 1, Figure 1 shows a bar chart of item counts in two categories (this is a binary classification problem).

Moreover, there are 416 items without any content that come from both train (398 items) and test set (18 items), we will drop all of empty content items in the training set because they don't bring any useful information for us.

2.2 WordCloud

Figure 2 shows us WordCloud [6] of the most frequent words that we usually see in the training data where the articles are related to earnings of a company. As we can see that words like mln, vs, ct, dlr, net... appear many times, the more times they appear, the bigger the word in the graph. We used the same technique for the non-related to company earning category (label 0) and we observed a different pattern, their common set of frequent words are not the same as in other category. Words like said, will, year, pct appear many times (Figure 3).

Figure 2: WordCloud for articles that are related to the earnings of a company [6]

Figure 3: WordCloud for articles that aren't related to the earnings of a company [6]

2.3 Word Frequency

Listing 1 displays the 20 most common frequent words in category label 1 and Listing 2 shows result for category label 0. This is inline with our WordCloud results also.

```
[('vs', 3189), ('mln', 2715), ('cts', 1963), ('dlrs', 1604), ('net',
    1220), ('loss', 1092), ('said', 880), ('shr', 862), ('profit', 749),
     ('year', 727), ('revs', 545), ('share', 516), ('1986', 510), ('
    billion', 408), ('oper', 360), ('company', 345), ('note', 321), ('
    sales', 307), ('quarter', 292), ('march', 292)]
```

Listing 1: 20 most common words for True label articles

```
[('said', 11155), ('pct', 3890), ('mln', 3592), ('dlrs', 3484), ('will',
     3118), ('billion', 1889), ('us', 1706), ('year', 1702), ('company',
     1576), ('inc', 1482), ('new', 1361), ('bank', 1307), ('last', 1238)
    , ('corp', 1221), ('market', 1127), ('one', 982), ('shares', 967), (
    'stock', 959), ('debt', 859), ('two', 854)]
```

Listing 2: 20 most common words for False label articles

Figure 4 and 5 represent the frequency patterns where the y axis shows us the frequent of words and the x axis displays the word rank by their frequency. As a result, we can say that our data follows the power-law distribution [9].

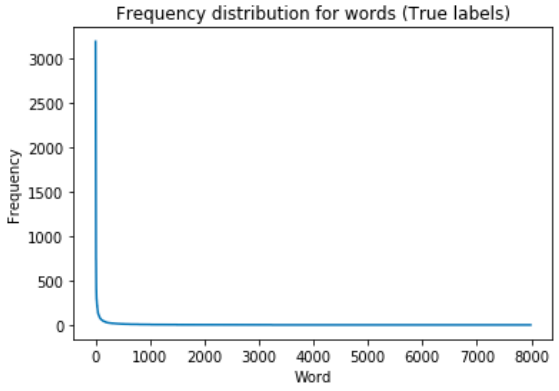

Figure 4: Word Frequency (True labels) [9]

Details about the implementation can be found in notebook Data Analysis.

3 Methodology

In the methodology, we will apply several algorithms for the prediciton problem. The first one is the word embeddings with Deep Learning model, followed by some popular algorithms namely LinearSVC, AdaBoost, and LogisticRegression.

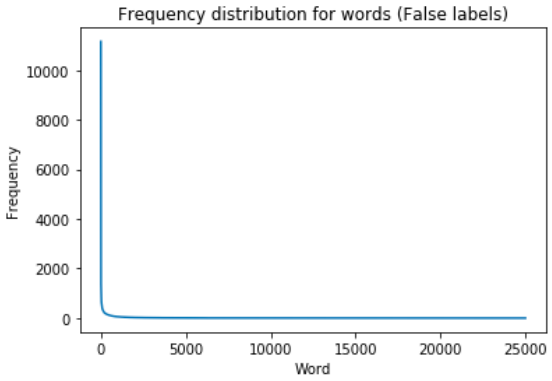

Figure 5: Word Frequency (False labels) [9]

3.1 Word Embeddings

3.1.1 Motivation

Currently, there are many NLP techniques that treat words as atomic units but there is no notion of similarity between words and they are represented as indices in vocabulary. Those techniques are very popular for its simplicity and robustness. One example of these techniques is the N-gram model which is widely used in statistical language modelling [2].

There are two main categories for word embeddings models namely the counted-based (using count vector, TF-IDF vector, Co-occurence vector...) and the predictive based models (using Neural probabilistic language model, `word2vec, GloVe` and `fastText`...). In this project we will apply the word embeddings technique named `word2vec` in order to learn good vector representations for words which will be used later in building deep learning model for text classification problem.

3.1.2 `word2vec` Model

The `word2vec` technique was developed by Tomas Mikolov et. al in 2013. The main objective of the technique is not only to learn high-quality vector presentation for words where similar words tend to stay close together in high dimensional vector space but also to reduce training time on huge dataset [2].

In the paper [2], the author proposed two model architectures for learning distributed representations of words and minimizing the computation complexity namely CBOW (Continuous bag-of-words) and Skip-gram model.

- CBOW (Continuous bag-of-words): In this method, the author predicted the target word w_t by using n-words before and after it. It used continuous representations whose order doesn't matter (Figure 6).
- Skip-gram: This model is reversed from the CBOW as it used the context word to predict its surrounding words (Figure 7).

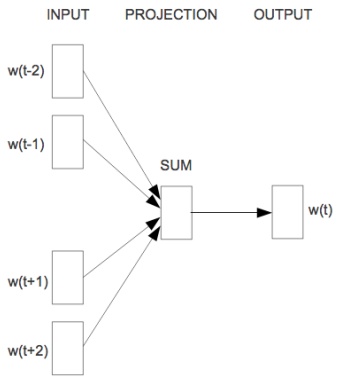

Figure 6: Continuous Bag of Words architecture.[Mikolov et. al., 2013][2]

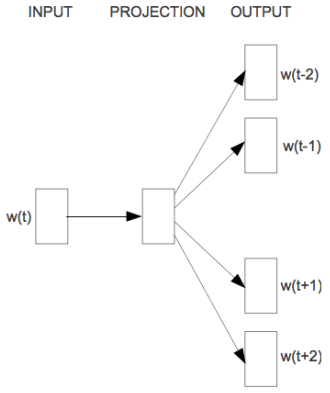

Figure 7: The Skip-gram model architecture [Mikolov et. al., 2013][2]

3.1.3 Data preparation for `Gensim`

In this section, we will learn word embeddings using `Gensim` package.

First, we have to read our training data and their contents in XML format and merge them into one dataframe (saved under file name `news_datafile.csv` for later use). Then we will drop all `NaN` values which will result in a new dataset of around `4,402` rows with four columns as in Figure 8.

Next, we will convert the contents of all training articles into lowercase, replace many spaces by single space, remove special characters and the word `reuter` which always appear at the end of the article content, and make use tokenizer from NLTK package so that at the end we have a set of sentences for each article where each sentence is a set of words. This will be the input data for our training word embedding model.

Listing 3 shows an example output of first 5 sentences for article at index `12` in our dataframe.

```
['oper', 'shr', 'loss', 'two', 'cts', 'vs', 'profit', 'seven', 'cts']
```

```
['oper', 'shr', 'profit', '442000', 'vs', 'profit', '2986000', 'revs', '
    2918', 'mln', 'vs', '1511', 'mln']
['avg', 'shrs', '517', 'mln', 'vs', '434', 'mln']
['six', 'mths']
['oper', 'shr', 'profit', 'nil', 'vs', 'profit', '12', 'cts']
```

Listing 3: Sample result

id	file	label	content
1	1.xml	0	Showers continued throughout the week in\nthe ...
2	2.xml	0	Standard Oil Co and BP North America\nInc said...
3	3.xml	0	Texas Commerce Bancshares Inc's Texas\nCommerc...
4	4.xml	0	BankAmerica Corp is not under\npressure to act...
5	5.xml	0	The U.S. Agriculture Department\nreported the ...

Figure 8: Sample data after merging contents and labels and removing empty contents.

3.1.4 Training `word2vec` with Gensim

After having all the data in `Gensim` [5] format, we now can begin the training process. Let's discuss about the parameters that we will set for the model [4, 5].

- `num_feature`: This is the dimension of our word vector, the more dimension the better the our representation will be. However, large dimension vector will also result in longer training time. Here we use the default dimension of 100.
- `min_word_count`: This is the minimal appearance of a word to be considered in the training. Here we also use default value 5.
- `window_size`: This is the number of words to be taken into account around both the left and the right of the target word. Here we use a window size of 10.
- `down_sampling`: The threshold for configuring which higher-frequency words are randomly downsampled, useful range is (0, 1e-5).
- `num_thread`: Number of parallel processes to run. We set it to 5.
- `iteration`: Number of iterations (epochs) over the corpus. Higher number will result in better vector representation. We set it to 30.
- Training algorithm: We will pick CBOW as the default value as this method works well with small dataset and in our case we don't have large number of data.

The training time took around 434.5s (∼7.3 min) to complete and there are `7,651` words in our result model.

After having the model, we will save the vocabularies and their corresponding vectors into two different lists (`word_lst_gensim_w2v, word_vec_gensim_w2v`) so later we can reuse them. There is one thing to note is that we add the character `"-"` as the padding at the beginning and `"unk"` for unknown word at the end of the word list and they will have corresponding zero vectors in the vector list. The reason behind this will be discussed more in the implementation of our deep learning model.

For more details about this implementation, please refer to the notebook `Learning word2vec embeddings with Gensim`.

3.1.5 Deep Learning Model with Keras

Before training our Deep learning model for text classification problem, we have to pre-process our training data to feed into the training model.

- Trainig data:
  - First, we will load again our file `news_datafile.csv` and drop all `NaN` values.
  - Then we will clean the data by removing special characters, extra spaces, etc.. This is very similar to the cleaning process we did when learning `Gensim` word embeddings.
  - The requirement for training the model is that all input data must have the same length which means that we have to set the maximum length for our news articles. Figure 9 show the distribution of the length and its frequency in our training dataframe. The average number of words per article is 130 words. Normally, we can set the maximum length to this number but here we want to keep more information in the article so we set the maximum length to 200 words.

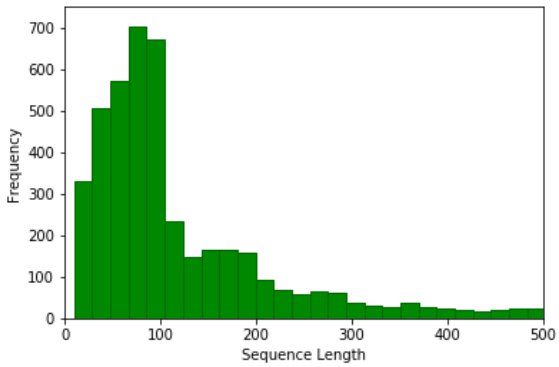

Figure 9: Frequency of news article length

- Here, we don't break down each article into a set of sentences but instead we will format the data by converting them into array of index corresponding to its position in the vocabulary list we got from `Gensim` model. The result output will be article array with the same `max_seq_len` containing index for each word. Any article with the length less than 200 words will be padded with zero values at the beginning of the array whereas for articles with longer length will be trim at the beginning. After formatting, our result data will be as in Listing 4.
- For any words that can't be found in our vocabulary list, it will be replaced by the `"unk"` index (which is `7652`). For padding zero that we add at the beginning of the array for short article, it will take index of padding index in our vocabulary (which is `0`). Both of those characters will take vector of zero values.

- Test data: For test data we apply the same formatting technique to get the output arrays which will be used for prediciton later.

```
Article at index 10 in words:

['shr', '34', 'cts', 'vs', '119', 'dlrs', 'net', '807000', 'vs', '
    2858000', 'assets', '5102', 'mln', 'vs', '4797', 'mln', 'deposits',
    '4723', 'mln', 'vs', '4403', 'mln', 'loans', '2992', 'mln', 'vs', '
    3272', 'mln', 'note', '4th', 'qtr', 'not', 'available', 'year', '
    includes', '1985', 'extraordinary', 'gain', 'from', 'tax', 'carry',
    'forward', 'of', '132000', 'dlrs', 'or', 'five', 'cts', 'per', 'shr'
    ]

Same article in index values where each word is corresponding to its
    position in vocabulary word list:
[    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0   65  943   29   17
  2518   11   41 7652   17 7652  323 7652    8   17 7652    8  789 7652
     8   17 7652    8  264 7652    8   17 7652    8  177 1374  451   36
   263   25  277   95  617  306   15  123 1413 1942    2 7652   11   46
   115   29   89   65]
```

Listing 4: Sample index of news article after formatting

After finishing pre-process our data, we can start the training model.

- To train model we use `Keras` framework.
- We will split training data into train and validation set with ratio of `80:20`.
- Listing 5 shows the summary of our deep learning model in `Keras`. Basically, we will have input layer of size `[batch_size, article_max_legth]` (200 is the length of each article), an `Embedding` layer that accept input of `[batch_size, article_max_legth, word_vector_dimension]`, one `Flatten` layer and finally the `Dense` layer with `sigmoid` activation function as our problem is a binary classification problem.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
main_input (InputLayer)      (None, 200)               0
_________________________________________________________________
embedding_3 (Embedding)      (None, 200, 100)          765300
_________________________________________________________________
flatten_6 (Flatten)          (None, 20000)             0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 20001
=================================================================
Total params: 785,301
Trainable params: 20,001
Non-trainable params: 765,300
_________________________________________________________________
```

Listing 5: Model summary

- For the training, we use `Adam` optimizer and `binary_crossentropy` loss with early stopping to prevent over-fitting. Fortunately, the training went really fast and it stopped after 3 epochs thanks to early stopping.

Details about the implementation is available in Jupyter Notebook `[word2vec] Text_Classification`.

3.1.6 Result

After training, the final model returned the accuracy on validation set at around `93.64%` (Listing 6).

```
Train on 3521 samples, validate on 881 samples
Epoch 1/30
3521/3521 [==============================] - 2s 535us/step - loss:
    0.0045 - acc: 0.9997 - val_loss: 0.6894 - val_acc: 0.9376
Epoch 2/30
3521/3521 [==============================] - 0s 104us/step - loss:
    0.0045 - acc: 0.9997 - val_loss: 0.6957 - val_acc: 0.9353
```

```
Epoch 3/30
3521/3521 [==============================] - 0s 95us/step - loss: 0.0045
    - acc: 0.9997 - val_loss: 0.6965 - val_acc: 0.9364
```

Listing 6: Training Log

If we use the trained model to predict our validation data we can get confusion matrix as in Figure 10 with accuracy of `93.75%`.

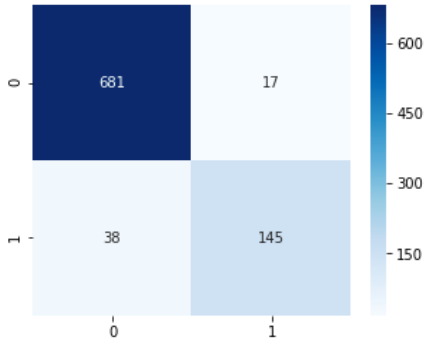

Figure 10: Confusion matrix on validation set for Deep Learning model using `word2vec` embeddings

3.2 `LinearSVC`

Here we use Linear Support Vector Classification `LinearSVC` from `sklearn` [10] which is similar to `SVC`.

3.2.1 Data Preparation

Before starting with the training, we will prepare our data. Data will be read from file `news_datafile.csv` which was the output from our pre-processing with `word2vec` model. After dropping all `NaN` values, it will be splitted into ratio `80:20` for train and validation set. This setting will be the same for `AdaBoost and LogisticRegression` classifiers.

3.2.2 Training

To train the model, we create a common function that can be re-used for other algorithms. We apply the `Pipline` class [8] from `sklearn` package. First, we will vectorize input data with `CountVectorizer` then use a transformer `TfidfTransformer` followed by a classifier of our choice, here we use `LinearSVC`.

For the `CountVectorizer` we set up same parameters for other algorithms:

- `analyzer='word'`: this will apply tokenization based on `word`.
- `stop_words='english'`: We also apply technique to remove stopwords for input data. Here we use pre-defined set of stopwords in `sklearn`.
- `max_features=1000`: This is used to limit the number of features in our training process. Because we use vectorize technique, the result of this will be a very large dimension vector. As a result, we will limit this for better training.

For `LinearSVC` training parameters we use the default settings with `random_seed=123`.

3.2.3 Result

After training, we got accuracy of `96.935%` on the validation set. Figure 11 shows a confusion matrix for classification results.

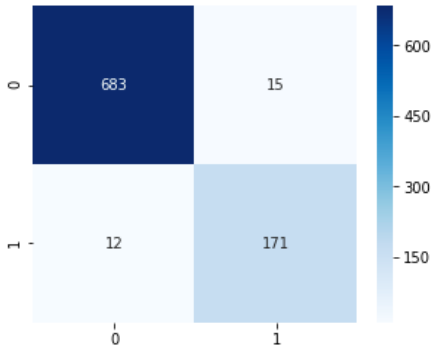

Figure 11: Confusion matrix on validation set for `LinearSVC` classifier

Details about the implementation can be found in notebook `[Other models] Text Classification`

3.3 `AdaBoostClassifier`

`AdaBoost` classifier [11] is a boosting algorithm that is used to convert a set of weak learners into a stronger one by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

3.3.1 Training

For training this classifier, we use the same data processed in the previous algorithm and do some parameter settings as follows:

- `n_estimators=30`: the maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.
- `random_state=123`: this is the random seed.

For other training parameters we use the default settings.

3.3.2 Result

The model accuracy on validation set is `95.80%`. Listing 12 shows confusion matrix on validation prediction results.

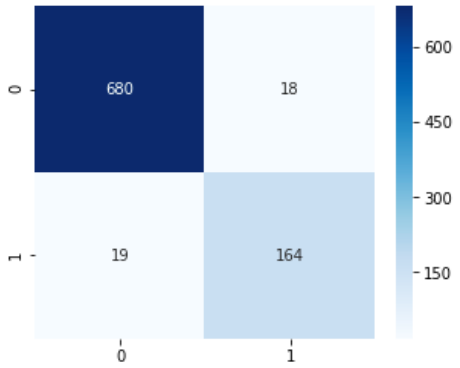

Figure 12: Confusion matrix on validation set `AdaBoost` classifier

3.4 `LogisticRegression`

Logistic Regression is a very popular classifier for binary classification problem which utilized the `sigmoid` activation function [12].

3.4.1 Training

Same processed data was used for this model with below settings for parameters:

- `solver='lbfgs'`: this solver supports L2 regularization with primal formulation.
- `random_state=123`: this is the random seed.

For other training parameters we use the default settings.

3.4.2 Result

Accuracy on validation returned `96.595%`. Figure 13 shows confusion matrix for the validation results.

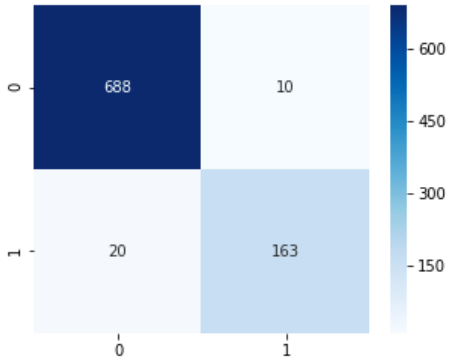

Figure 13: Confusion matrix on validation set for `LogisticRegression` classifier

4 Conclusion and Suggestions

Table 1 shows the summary of accuracy on four different algorithms that we implemented above. As we can see that `LinearSVC` has the highest accuracy.

Algorithms	**Accuracy (%)**
Linear SVC	**96.93**
Logistic Regression	96.59
Adaboost	95.80
`word2vec` embeddings with Deep learning	93.75

Table 1: Summary of accuracy

It can been seen that among our implemented models, the most complicated one is the `word2vec` embeddings with Deep Learning. Unfortunately, this accuracy is not as high as its counterparts and data preparation for this model is not as simple as the others, so we recommend to pick `LinearSVC` to be our final model. Hence we will use this model to predict on our test data. The prediction result can be found in the file `svc_test_results.csv`.

References

[1] Tomas Mikolov, Wen-tau Yih and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In Proceedings of NAACL HLT, 2013.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013).

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. arXiv:1310.4546 (2013).

[4] Doan Tu My, Learning Word Embeddings - Year-end project. https://github.com/doantumy/Word-Embeddings

[5] Gensim package - https://radimrehurek.com/gensim/models/word2vec.html

[6] Andreas Mueller, WordCloud package, https://github.com/amueller/word_cloud

[7] Jovian Lin, Embeddings in Keras: Train vs. Pretrained, https://jovianlin.io/embeddings-in-keras/

[8] Working with Text data from sklearn - https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

[9] Brendan Martin Nikos Koufos, Sentiment Analysis on Reddit News Headlines with Python's Natural Language Toolkit (NLTK)- https://www.learndatasci.com/tutorials/sentiment-analysis-reddit-headlines-pythons-nltk/

[10] Linear Support Vector Classification from sklearn - https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[11] AdaboostClassifier - sklearn - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[12] Logistic Regression - sklearn - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html