Project Report

Classification of the Office Quotes NLP Project

Babiak Chiara (if20b061), if20b061@technikum-wien.at
Führer Angelika (if20b090), if20b090@technikum-wien.at
Rinner Clemens (if20b279), if20b279@technikum-wien.at

Content

Summary

Natural Language Processing, short NLP, is a powerful branch of Artificial Intelligence and refers to a discipline which strives to make computers able to understand spoken as well as written words, meaning giving the computer the power to process human languages.

In this Project we used this technology to create an NLP model which classifies which character of our chosen show, the Office, said that specific sentence and if it was a reply or monologue.

The project environment chosen is a jupyter notebook written in the programming language python (ver. 3) and it includes and utilizes the following libraries:

- Pandas
- Nltk
- Re
- Sklearn
- Genism
- Tensorflow

Following Links are important for the Project:

- GitHub
- Kaggle Dataset

Task and Data Description

Task

The main Goal of this project is to classify a quote from the show The Office. The classification is either to be done by main character or by quote-class, which includes a reply or a monologue (from a talking head interview) quote.

Data

The Office is an American TV-Show which is filmed in a mockumentary format, meaning it is filmed if it was a documentary and follows some of the conventions. These include characters being aware of the camera, capturing candid moments with a Spy-Cam and the talking head interview, which only include a singular character responding to questions or sharing their thoughts. The main characters of this show are Michael, Dwight, Jim, and Pam.

The dataset that we have chosen consists of 2 csv files:

- Parent.csv
  This file contains quotes of main characters replying to others.
- Talking_head.csv
  This file contains quotes of main characters while they were holding a talking head interview.

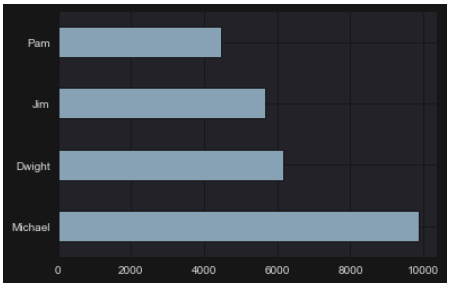

Figure 1: Bar Chart showing the distribution of character quotes in the parent.csv file

The parent.csv holds a total of 26150 datapoints. Each datapoint consists of an id, the reply quote, the character, and the parent. The parent here describes the sentence that one of the characters was replying to.

The second csv, talking_head.csv, consists of 1749 quotes. In here each datapoint only consists of character and quote.

One can already see that there is an imbalance present between monologue- and reply-quotes. The monologue quotes only take up 6.27% of the dataset.

To get a greater insight of the dataset we did some descriptive coding work. The results were that from the 27899 quotes (replies and monologues) 10602 quotes were from Michael. Therefore,

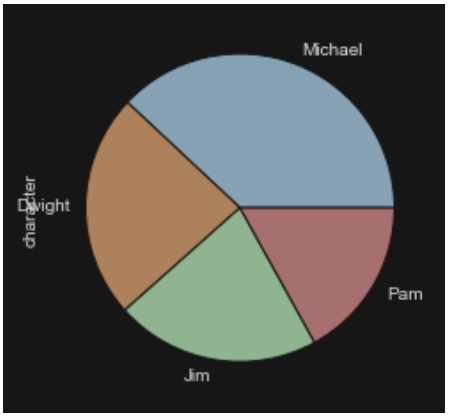

Figure 2: Pie chart showing distribution of characters in all the datapoints

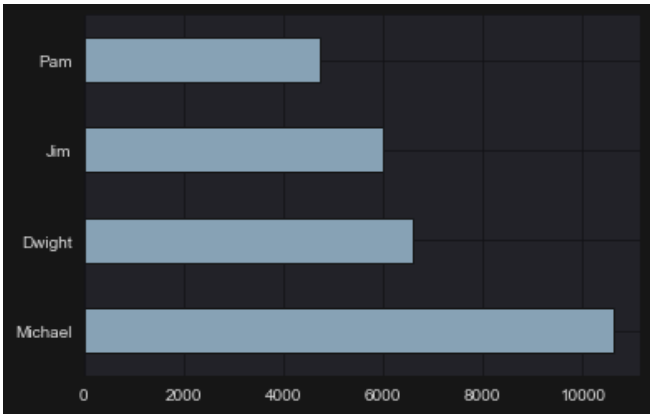

Figure 3: Bar chart presenting the distribution of characters over the entire dataset

Michael takes 38% of the Dataset, around 10% more than the other three main characters.

This This character imbalance is due Michael having the most prominent speaking role and as a result is present in nearly all scenes and holding the most talking head interviews. Alone in Season One Michael Scott has a percentage of 60.97 in Screen Time. [1]

How does the project relate to real applications?

The Office is not the only big TV-Show on the market with a huge following and fanbase supporting it for years. There are many more such shows present (Game of Thrones, Sherlock, Hannibal, etc.) which fanbase is obsessed with the content and would love to know more. Sometimes I catch myself asking how present was Pam's talking role in a specific episode?

Without much manual work, such an AI could analyse the transcript of a given episode and tell the End-User the percentage of a certain character's speaking role.

When generalizing this idea, the NLP model could be able to return and analysis of the entire transcript of a certain episode for each character. Not only that could be done, but the text of a certain character could be extracted and presented.

Meaning that this Model has similarities with Content analysis and could further be used to extract e.g.: Michael Scott Quotes from transcripts to use as input/training data for a Michael Scott Chatbot, especially since we could train the AI with replies as well as independent statements.

Problems encountered and their Solutions

Imbalances/Class weights

As mentioned before we can see a certain imbalance being presented in our Dataset, especially visible when focusing on Michael (highest occurrence) and Pam (lowest occurrence). Such imbalances can lead to a model having a worse performance, than when training and testing with balanced data.

To check on that we did train the Models with both balanced and imbalanced data. To balance the dataset that we had we used a module from the python library sklearn called compute_class_weight. The class weights that were calculated here are later used in the model fit process.

Choosing Classifier

The difficulty when we started working on the model was the decision about which classifier to use. Such Classifiers would be the LSTM, Linear regression and Naïve Bayes. The first Model we tried out was the LSTM. This decision was made based on the fact that the LSTM has a long short term memory, which is important in NLP.

This was also the model that was worked on the most, but the curiosity caught the best of us and we decided to also implement and try out other models to check and compare the accuracies for our dataset.

LSTM

Working with LSTM was a new challenge for our team. It required loads of trial and error work especially when talking about the input needed for this kind of classifier.

To solve the problem between what our available data is and the actual needed input we had to transform our data (The way we transformed it will be discussed in the next chapter.).

Embedding Layer

With the embedding layer we had the exact same issues: Again, working with embedding layers was a new challenge for us and required a lot of research as well as trial and error work.

Description of Code

In this chapter a better insight on the steps and an explanation on why something has been implemented that way. The descriptive part of the code has been left out, since the results have been presented in the first chapter.

Pre-processing

Pre-processing is a really crucial step when wanting to build a model. This is due to the fact that the algorithm learns based on the data provide and thus the input has to be suitable for a machine and increases the accuracy of the resulting model.

Typical pre-processing steps are:

- Tokenization
- Lower casing
- Stop word removal
- Stemming
- Lemmatization

And these are also the steps we went through with our dataset. For describing the dataset a Data-Frame was created which was later also utilized here. Since we have a DF and not a simple text file, where we would just iterate through the given text functions were created for each step for a more simple implementation. To illustrate the architecture behind it, we will take a closer look at the Tokenization.

In [126…
```
def tokenize(column):
    tokens = word_tokenize(column)
    return [w for w in tokens if w.isalpha()]
```

In [127…
```
# we will not change the quote column we will be adding a new table to the merged dataframe
df_merged['tokenized'] = df_merged.apply(lambda x: tokenize(x['lower']), axis = 1)
df_merged.head()
```

Out[127…

	quote	character	lower	tokenized
0	So you've come to the master for guidance? Is ...	Michael	so you've come to the master for guidance? is ...	[so, you, ve, come, to, the, master, for, guid...
1	All right. Well, let me show you how it's done.	Michael	all right. well, let me show you how it's done.	[all, right, well, let, me, show, you, how, it...
2	If you think she's cute now, you should have s...	Michael	if you think she's cute now, you should have s...	[if, you, think, she, s, cute, now, you, shoul...
3	Any messages?	Michael	any messages?	[any, messages]
4	Oh! Pam, this is from Corporate. How many time...	Michael	oh! pam, this is from corporate. how many time...	[oh, pam, this, is, from, corporate, how, many...

Figure 4: Illustartion of architecture behind preprocessing steps

Here we can see how the function tokenize is being called for each row of the dataframe using lamda and inserted into an extra column. The columns are used for the next step in the process and are deleted afterwards. This does grow the runtime fo the program, but makes it easier to look at the process step by step instead of creating a blackbox.

Models

Before moving on to train and test models, the quotes had to be vectorized. This is due to the fact. That machine learning models need numerical data and we have strings instead. Commonly used methods to vectorize data are Bag-of-Words (BOW) and Word Embedding.

Therefore we have decided to use the CountVectorizer and the Tf-Idf. Why did we use both? We have decided to train and test multiple types of machine learning models, including Linear Regression, which needs the conversion of text to a matrix of token counts as an input concerning NLP.

Clustering

First we worked on the k-Means/Clustering Model. This Model takes the Tf-idf vectorization as input. The most important decision one had to make here is the amount of clusters, since these have to be pre-specified before training the model.

In our case we have chosen 8 clusters.

Linear Regression

Linear Regression is the second model. Here we just used the CountVectorizer Results as an input. More decision did not have to be made. What we did for a better insight was to print a confusion matrix of y_test and y_pred, as well as printing out the classification report, since that is another way of showing us the quality of our predictions .

Figure 5: Confusion Matrix of Linear regression

	precision	recall	f1-score	support
Dwight	0.46	0.31	0.37	1338
Jim	0.35	0.23	0.28	1206
Michael	0.45	0.73	0.55	2093
Pam	0.35	0.17	0.22	943
accuracy			0.43	5580
macro avg	0.40	0.36	0.36	5580
weighted avg	0.41	0.43	0.40	5580

Figure 6: Classification Report of Linear Regression

To see if there are any differences, we tested out the Linear Regression again with the Tf-Idf vectors. We put the solver to libnear, which is limited to one-versus-rest schemas [2], exactly what is needed in our example. Since we have chosen that soler we are only able to use either l1 or l2 as penalty parameters. We have chosen the l1 penalty, since it is more robust and the l2 regularization increases the cost of outliers in the data exponentially due to the fact, that inl2 we use the square of weights, while in l1 the sum of absolute values of the weights is taken.

Figure 7: Confusion Matrix of Linear Regression with tf-idf vectors

	precision	recall	f1-score	support
Dwight	0.45	0.33	0.38	1338
Jim	0.34	0.26	0.29	1206
Michael	0.44	0.68	0.54	2093
Pam	0.34	0.17	0.23	943
accuracy			0.42	5580
macro avg	0.39	0.36	0.36	5580
weighted avg	0.41	0.42	0.40	5580

Figure 8: Classification Report for Linear Regression with tf-idf vectors

Naive Bayes

Here not a lot of decisions had to be made.

Figure 9: Confusion Matrix for Naive Bayes

	precision	recall	f1-score	support
Dwight	0.56	0.17	0.26	1338
Jim	0.42	0.07	0.11	1206
Michael	0.40	0.94	0.56	2093
Pam	0.47	0.02	0.04	943
accuracy			0.41	5580
macro avg	0.46	0.30	0.24	5580
weighted avg	0.45	0.41	0.30	5580

Figure 10: Classification Report of Naive Bayes

LSTM

LSTM, short for Long short-term memory, is very effective in memorizing important information. It can use multiple word string to find out the class to which it belongs. The first hyperparameters of LSTM is units and defines the number of neurons that work a the memory of the model.

Here again we had to change the data shape before using the quotes in the sequential mode. In the next step the class names (here: character names) had to be converted into numerical data and the representation in our DF had to be changed accordingly (Michael: 0, Dwight: 1, Jim: 2, Pam: 3). Next we used the DF, which only includes the numerical data, to convert the text into so called sequences. All the sequences have the same length by filling them with 0s.

Embedding Layer

The Embedding Layer of Keras takes the previously calculated integers of teh sequences and maps them to a dense vector of the embedding.The following parameters are defined:

- input_dim: the size of the vocabulary
- output_dim: the size of the dense vector
- input_length: the length of the sequence, in our case the max_length used while pad_sequences

Dense Layer

The neuron of a dense layer in a model receives output from every neuron of its preceding layer, where neurons of the dense layer perform matrix-vector multiplication. The keras dense layer can take a set of hyperparameters of which we used the following:

- Units: defines the size of the output from the dense layer = dimensionality of the output vector
- Activation: activation function used for the transformation of the input values of neurons -> introduces non-linearity into neral network (necessary for learning th relationship between input and output values)

The last Dense layer of the model must have the number of classes as unit parameter

For the first Sequential Model we used 2 LSTM layers with 64 units and 3 Dense Layers, 2 with 64 units and one with 32 units. The Dropout Layers in between are used to not overfit our model. The activation function here was the Softmax.

Since we have worked on a Sequential Model for the first time we have used these parameters, since they are the most standard ones alongside 128, which was used in the second sequential model. To experiment a bit we have also changed the activation function in the model to sigmoid. Changing the activation function did not change anything though. More dense layers were added to the second Sequential Model as well, to dense the dimensions even more, in total there are 5 dense layers instead of only 3.

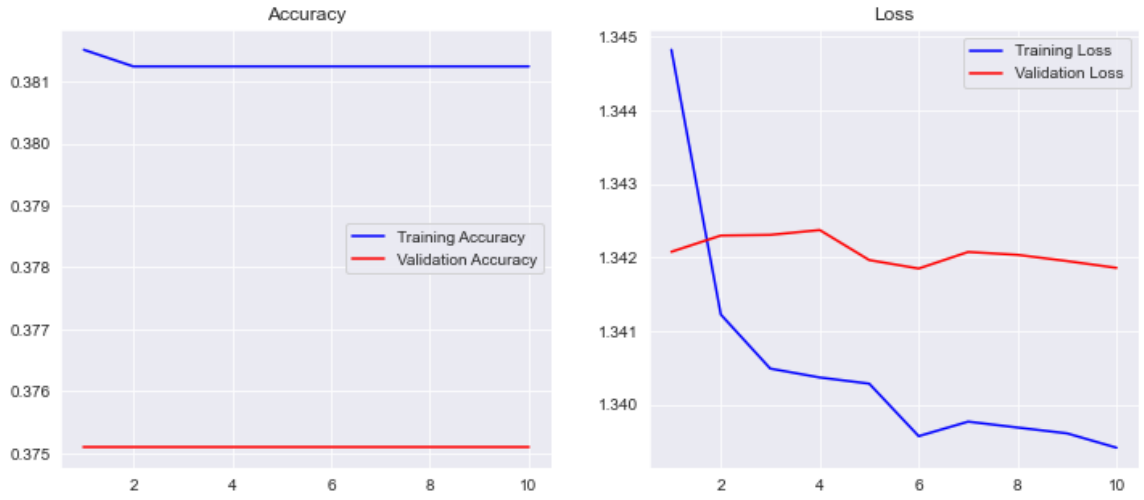

Figure 11: Accuracy and Lost of first Sequential Model

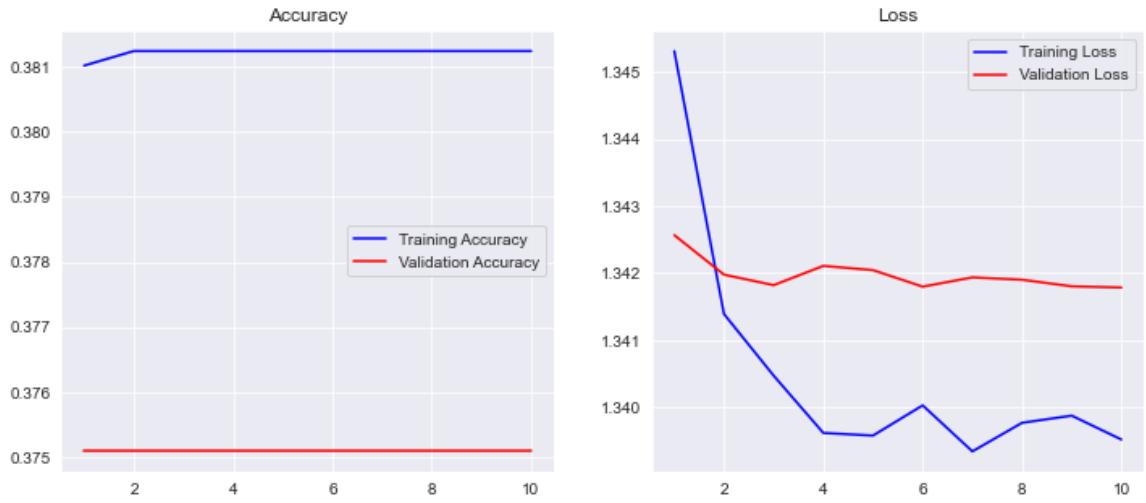

Figure 12: Accuracy and Loss of second Sequential Model

As said before, our data is imbalanced and we did calculate the class weights to see if they would impact the accuracy. Therefore they were used to implemented two other Models.

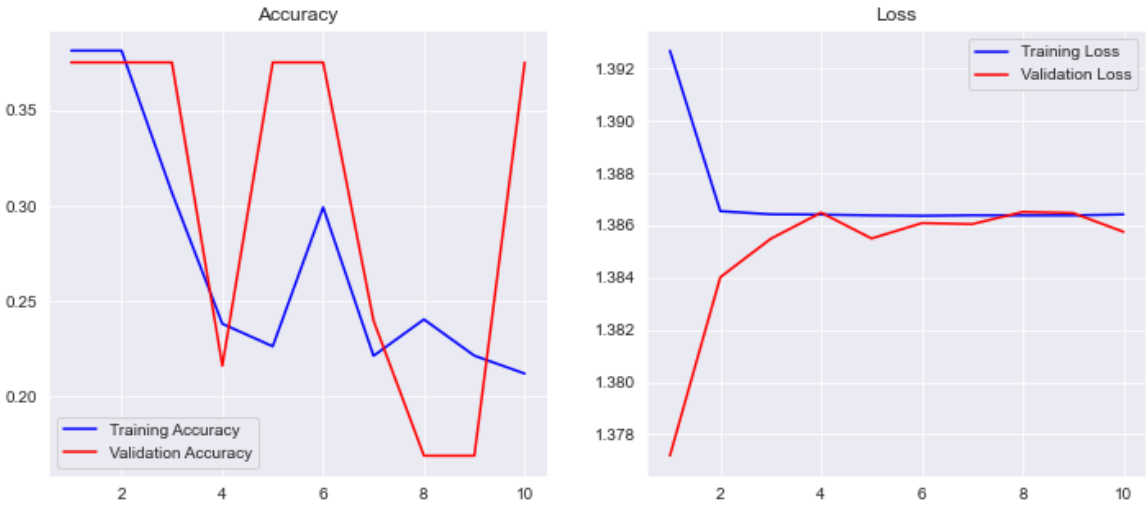

Figure 13: Accuracy and Loss of second Sequential Model with Balanced data

The other Model was a basic LSTM with only one Dense Layer. This Model was chosen, since the accuracy does not change when tuning the hyperparameters.

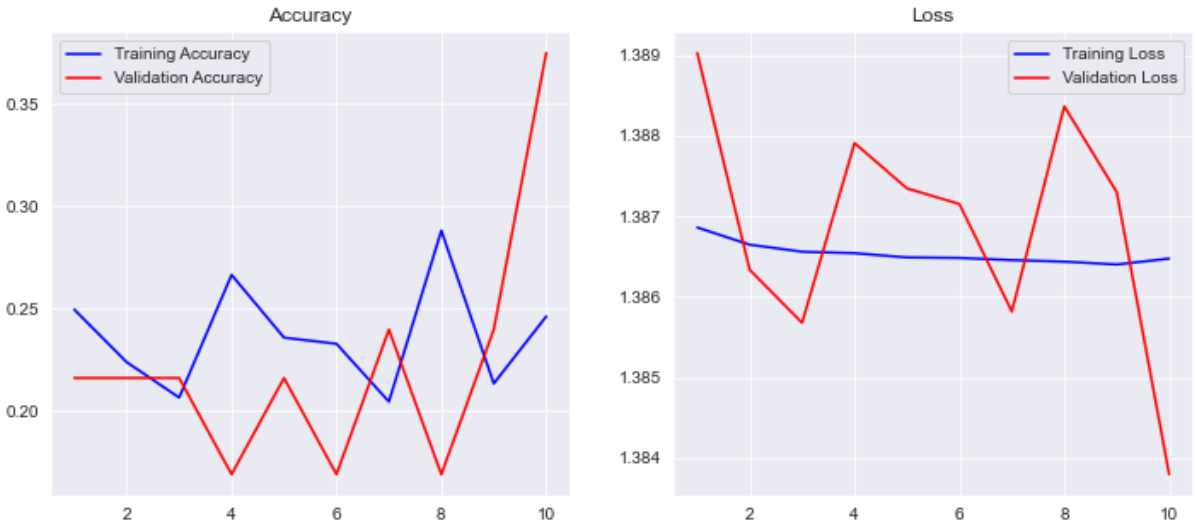

Figure 14: Accuracy and Loss for basic LSTM with one dense layer and balanced data

LDA

The LDA Model was also used, or at least implemented since we were curious which topics would be the output on our dataset. But since the words are not really meaningful for specific topics it was not the biggest success.

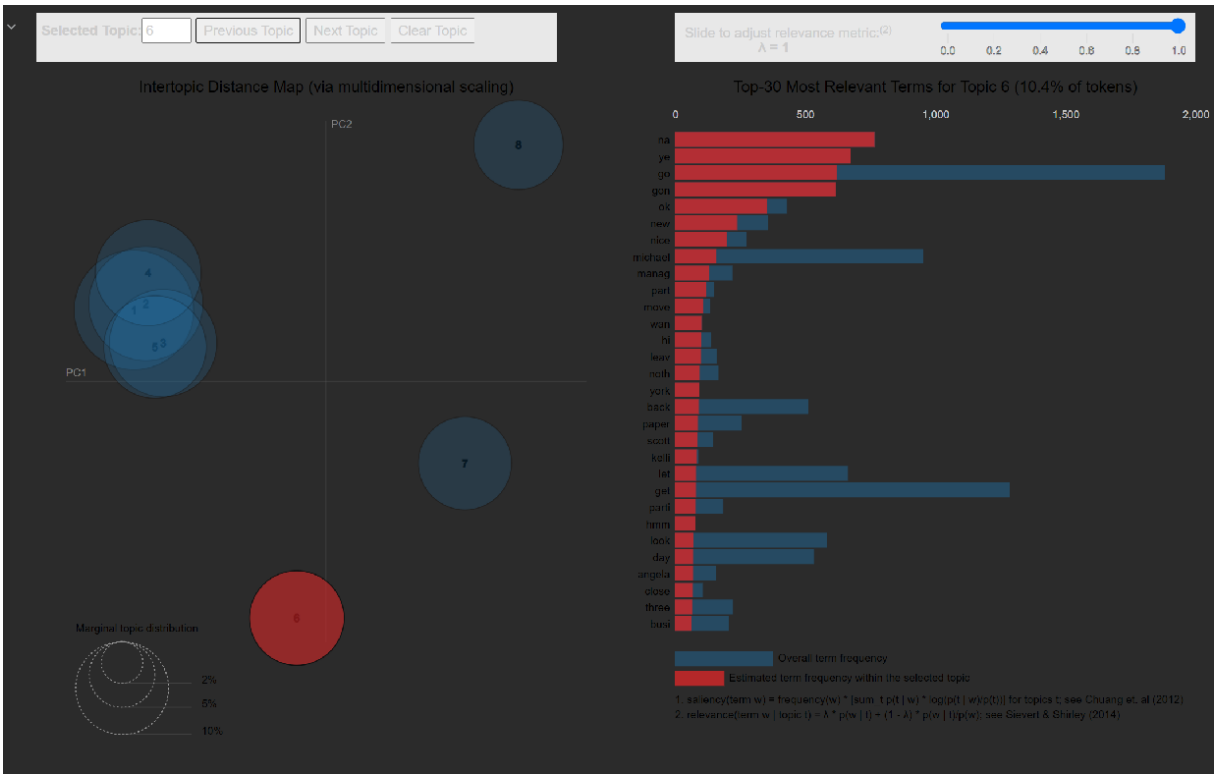

Figure 15: LDA interactive graph

Source Index

[1] @digg, “The ‘Office’ Cast Members Who Had The Most Screen Time In Each Season, Charted | Digg,” digg.com. https://digg.com/2020/the-office-actor-screentime-data (accessed Nov. 09, 2022).

[2] “sklearn.linear_model.LogisticRegression — scikit-learn 0.21.2 documentation,” Scikit-learn.org, 2014. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html