Classifying character sentences from the Simpsons

Benedikt Óskarsson
Reykjavik University
benedikto16@ru.is

Steinar Þorláksson
Reykjavik University
steinart16@ru.is

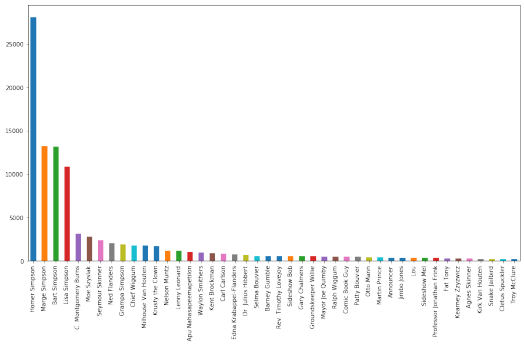
October 8, 2018

Abstract

In TV shows and movies the fundamental basis for most characters is what they say or their script lines. Characters often have unique dialog and can be identified from their script lines. In this experiment text classification is used to predict what character from a famous TV show could have said a given line. Classification methods that are implemented and compared are Naive Bayes, Linear Support Vector Machine, Logistic Regression and K-Nearest Neighbor. The most influential parameters for each classification model are tweaked and compared to get successful result.

I. Introduction

Text classification is the problem of automatically assigning zero, one or more of a predefined set of labels to a given segment of free text. The labels are to be chosen to reflect the "meaning" of the text. A widely used research approach to this problem is based on machine learning techniques [1].
The problem we are trying to solve in this paper is to classify characters from a TV show with the script line they have using supervised learning algorithms. The TV episode we chose was the simpsons [2], since it has a-lot of characters and usually relatively short lines they say. The simpson dataset[4] we used was created from 27 seasons of the episodes. Consisting of nearly 160 thousand rows of sentences or sequence of words, and 6732 characters. We quickly realized that predicting who said what sentence with over six thousand characters was not feasible, and therefor decided to work only with the characters who have at least two hundred lines in our dataset, reducing the number of characters to 44, which should give us a fighting chance. Here we can see the distribution of word sequences said, where Homer says the most, and then the simpsons family and so on.



The attributes of the dataset we are going to use are the text said in stemmed normalized form and classifying how said it. The algorithms we considered were Naive Bayes, Support Vector Machines, Logistic Regression and K-Nearest Neighbor, which are all known to work pretty decently with text classification.

II. Process

i. Vectorizers

To be able to classify data on text some kind of prepossessing is necessary and the most common way is to represent the data as a vec-

tor space model [3]. Two common word vectorizer used are Count-vectorizer and TFIDF-Vectorizer . The difference between the two is the CountVectorizer [6] counts the word frequencies but the TFIDF-Vectorizer increases proportionally to count and offset by the frequency of the word, known as inverse document frequency [5]. For the reasons mention the TFIDF-Vectorizer was used. Often it would be necessary to stem words to some normalized form, but for the data used this had already been done and it was possible to use the *normalized text* attribute out of the box.

ii. TFIDF-Vectorizer

- The vectorizer does weight calculation for each token. [7]

$$tfidf_t = f_{t,d} * log\frac{N}{df_t}$$

$tfidf_t = weigth\ of\ t$
$f_{t,d} = frequency\ of\ token\ t\ in\ document\ d$
$N = total\ document$
$df_t = documents\ that\ contain\ token\ t$

- Token length is set to be either single word, two words or three words with the *ngram_rnage* parameter.
- Stop words are eliminated with the stop words parameter. Stop words are word that do not have much meaning and are used more for grammar, words like *a*, *the*, *etc.*.
- To be able to make some kind of classification on a given token it has to appear at least few times so the minimum appearances required for each token is set to 5.
- A logarithm term frequency with sublinear tf scaling. For smoother scaling.
- Norm L2 is used because of the stability that the euclidean distance provides. [8]

iii. Splitting the data and Grid Search

Next the data was randomly split into test and training data with 80% training data and 20% test data, and vectorizer used to process the data. Each of the classification methods used has different key parameters that change how the model behaves and performs, to be able to identify what values suit this data we have to make an educated guess to try and guess possible values that are good. To do this we use a method called *Grid Search*. *Grid Search* takes in training data, classification model and a dictionary of parameters to try and train a new model for each combination of parameters and returns the best parameters values based on accuracy from cross validation of the training data.

In the experiment a lot of parameters where tried out and experimented with. After reading a lot up on different possibilities and trying them we decided one or more hyper-parameter for each model that were the most important ones in decided to to a organized grid search for the best values for those parameters.

iv. Models tried

Naive Bayes

Naive Bayes is a probabilistic classifier based on applying *Bayes' theorem* . Naive Bayes takes linear time which compared to many other classifier is very fast and that makes it good for bigger datasets. [10] Multinomial Naive Bayes is variation of Naive Bayes that uses multinomial distribution with the number of occurrences of a token and weight as classification feature. The ability to do this with each token makes it good fit for text classification. [7]
In Multinomial Naive Bayes the alpha parameter is a smoothing parameter used to make less difference between probabilities. For example if the alpha parameter is 0 there is no smoothing at all when multiplying independent probabilities together this can lead to the equation getting the value of zero because of single zero probability which is bad. So the idea of smoothing is to prevent this, but with more smoothing the calculations become more imprecise. With the grid search of the smoothing parameter the aim is to find the sweet spot between smoothing and precision. We also have a fit prior parameter that determines if

the classifier learns the class distribution probabilities beforehand or not.

Linear Support Vector Machine

Support vector machines are models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one category or the other an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space mapped so that examples of the separate categories are divided by a clear gap that is as wide as possible. To classify new examples we map them into the same space and predict the category based on which side of the gap they fall. [9]

When working with text we usually have a lot of features, and since SVMs use overfitting protection, which does not necessarily depend on the number of features, they have the potential to handle these large feature space, and therefor a good candidate for text classification. Since SVM's only have a binary output.To get an M-class classifier we must train M binary classifiers, each separating one class from the rest. Real-valued output of a SVM (distance to decision boundary) can be interpreted as confidence value. Then we choose the class with the highest confidence value. In support vector machines the C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

Logistic Regression

Logistic regression is a statistics model that uses a logistic function to model variables. It calculates probabilities for each instance it the dataset and finds the decision boundaries. This is done in an iterative process with new coefficients to that are based on error predictions. [11] Similar to the Naive Bayes the Logistic Regression uses probabilities for calculations for each token the difference is that the error is minimized with the iterative error predictions which makes it more precise but it takes more computation time.
The C parameter in the logistic regression is the a constant parameter that applies a regularization to decrease a magnitude of specific values to reduce over-fitting. Of course if the parameter becomes too large the model can easily become under-fitted. Using the grid-search the aim is to find the best value for C to not over-fit or under-fit the model. [12]

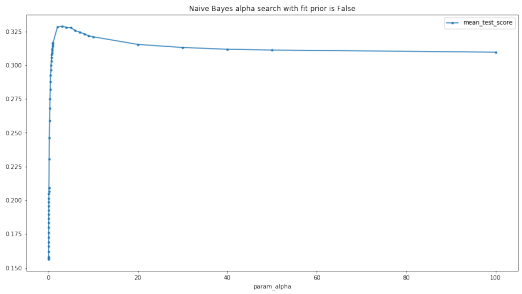
K-Nearest Neighbor

K-Nearest Neighbor is quite simple classification algorithm where the unknown data point is compared to the k closest points and then uses majority voting to decide which class the data point belongs too. K-Nearest Neighbor can be good for text classification if the data for each category is a lot different than the data in the other categories, for example each character in the Simpsons would always say unique different words from other characters. However for the Simpsons data this is not really the case for most characters so K-NN performs under average.
For the K-NN the main parameter is the K. If K is 1 the test data point is always voted to be same as the closest point in the training data, resulting in over-fitting, if K is too big way too many points will be used in the voting group resulting in under-fitting. The aim with the grid search is too find the k that gives us the best results.

III. Results

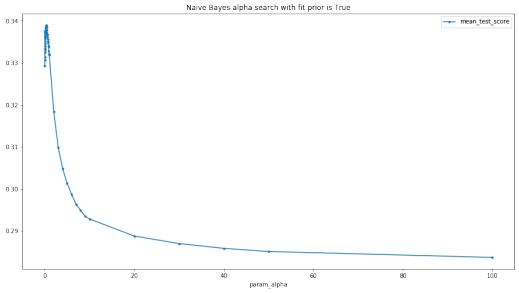
Naive Bayes

Parameter grid search for the alpha parameter where fit prior parameter is set as false.



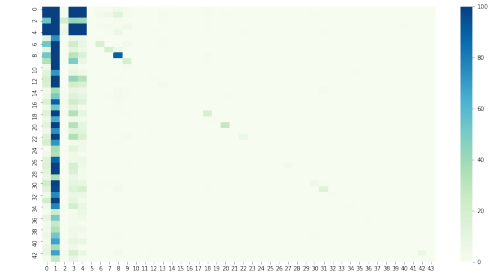
We can see here that if the alpha parameter is 0 or very low the accuracy is very bad, but as soon as it starts to go a bit up the accuracy gets much better and then slowly decreases again.

Parameter grid search for the alpha parameter where fit prior parameter is set as True.



We can see here that when the model that has learned the class distribution before hand we get a much better result with little or no smoothing at all, but as soon as the smoothing becomes to much the accuracy goes down really fast. We found that the best alpha parameter was 0.3368

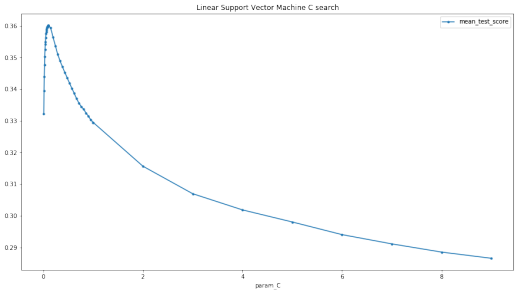
Confusion matrix for test predictions for Naive Bayes using chosen parameter.



This graph shows how dominating the majority classes are for the probabilities the Naive Bayes uses are and a few other classes are predicted right. The accuracy with the Naive Bayes model with top parameters was 0.352.

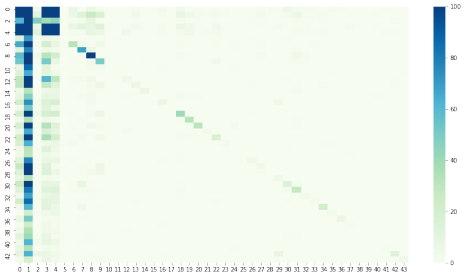
Linear Support Vector Machine

Parameter grid search for the C parameter.



We can see from the parameter grid above that we reach the best average accuracy with the C value as 0.0952, and if we increase C more than that the average accuracy of our model drops fairly quickly.

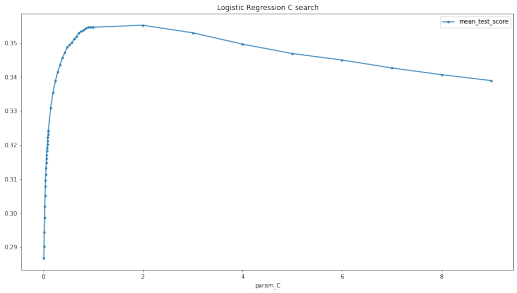
Confusion matrix for test predictions for Linear Support Vector Machine using chosen parameter.



Like we expected the linear SVM's misclassify a lot as the simpsons family when it is not the simpsons family. The accuracy we got with the support vector machine was: 0.369.

Logistic Regression

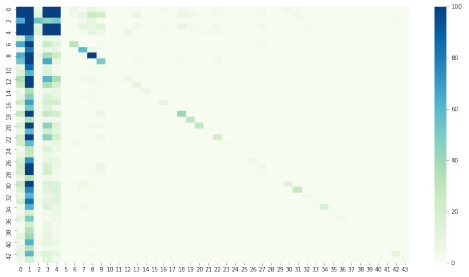
Parameter grid search for the C parameter.



In the parameter grid above we can see that by changing the C parameter the model gets better accuracy until we reach C=2, then we

go down again, which could be because of overfitting to the training data.

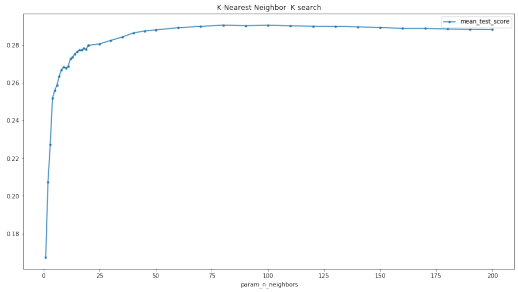
Confusion matrix for test predictions for Logistic Regression using chosen parameter.



As expected the confusion matrix above shows that we are misclassifying people that are not in the simpsons family as the simpsons family. The accuracy we got with the logistic regression model was 0.364.

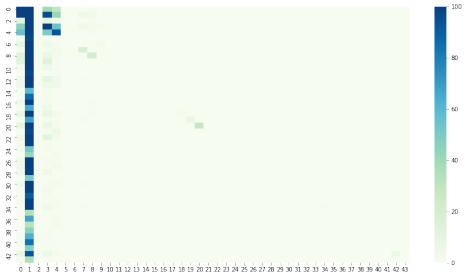
K-Nearest Neighbor

Parameter grid search for the K parameter.



As we can see from the parameter grid above, the KNN model peaks at K=80, and then slowly evens down.

Confusion matrix for test predictions for K-Nearest Neighbor using chosen parameter.



As expected the KNN model misclassifies the data most with people who are not Marge(Id 1) and classifies them as Marge. The reason for this is probably because Marge's data is spread out, and therefor has points all over the place and therefor could have a high count of points.

IV. Conclusions

Model	Accuracy
Naive Bayes	0.3515
Linear Support Vector Machine	0.3685
Logistic Regression	0.3644
K-Nearest Neighbor	0.2984

In this table it is clear that after tweaking the hyper-parameters that the accuracy of all the models is a bit similar. K-NN still has the worst accuracy by far and Linear Support Vector Machine has the best. However if we look at the confusion matrices we can see that both Naive Bayes and K-NN are predicting almost all predictions too the most popular character and some predictions too the other popular characters but almost nothing too others. The reason for this is likely too be that written text lines in TV shows taken out of context tend to be similar for many characters and hard to predict, so these models naturally predict most to the most popular classes. If the data would be more unique for each class these models would have the opportunity to perform much better classification. Looking at confusion matrices for Linear Support Vector Machine and Logistic Regression it is clear that they perform much better classifications for smaller characters as well, and this has likely a lot to do with how theses models perform a more detailed classification where the dominating classes are not as dominating like in the other models.
It is also clear by looking at the grid search plots that choosing the right parameters for the models has a lot to do with the performance and is the biggest factor in avoiding over- and under-fitting.

To sum it up we can see that we are getting an accuracy of up to 37% which is an decent result for a text classification problem with 44 different classes and a lot of short sentences

taken out of context.

V. Future work

We could continue working with the Simpsons dataset [4], and find a way to also include the location and episode-ID attributes into the training data of the classifiers. Each episode and each location is likely to have a lot of dialog from the same characters so this could easily improve the accuracy of models. Another approach to improve the models further is to use semantics for natural language processing.

With a decent classifier it would also be a fun idea to make a mobile application that can take in any sentence in English and predict which Simpsons character would be most likely to say it.

References

[1] Machine learning," Wikipedia, 04-Oct-2018. [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning. [Accessed: 08-Oct-2018].

[2] "The Simpsons," Wikipedia, 01-Oct-2018. [Online]. Available: https://en.wikipedia.org/wiki/The_Simpsons. [Accessed: 08-Oct-2018].

[3] Ø. L. Garnes. Feature Selection for Text Categorization. Oslo, 2009.

[4] W. Cukierski, "The Simpsons by the Data," RSNA Pneumonia Detection Challenge | Kaggle, 28-Sep-2016. [Online]. Available: https://www.kaggle.com/wcukierski/the-simpsons-by-the-datasimpsons_script_lines.csv. [Accessed: 08-Oct-2018].

[5] "TfidfVectorizer," Scikit-learn documentation. [Online]. Available: http://scikit-learn.org/stable/modules/generated/-sklearn.feature_extraction.text.-TfidfVectorizer.html. [Accessed: 08-Oct-2018].

[6] "CountVectorizer," Scikit-learn documentation. [Online]. Available: http://scikit-learn.org/stable/modules/generated/-sklearn.feature_extraction.text.-CountVectorizer.html. [Accessed: 08-Oct-2018].

[7] Bayu Yudha Pratama, Riyanarto Sarno, "Personality Classification Based on Twitter Text Using Naive Bayes, KNN and SVM". International Conference on Data and Software Engineering, 2015.

[8] "L1 Norms versus L2 Norms," Aleksey Bilogur. [Online]. Available: https://www.kaggle.com/residentmario/l1-norms-versus-l2-norms. [Accessed: 08-Oct-2018].

[9] "The Simpsons," Wikipedia, 01-Oct-2018. [Online]. Available: https://en.wikipedia.org/wiki/The_Simpsons. [Accessed: 08-Oct-2018].

[10] "Naive Bayes classifier," Wikipedia, 23-Sep-2018. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier. [Accessed: 08-Oct-2018].

[11] Augmented Startups, Udemy. [Online]. Available: https://www.udemy.com/machine-learning-fun-and-easy-using-python-and-keras/learn/v4/t/lecture/7696076?start=0. [Accessed: 08-Oct-2018].

[12] "Naive Bayes classifier," Wikipedia, 23-Sep-2018. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier. [Accessed: 08-Oct-2018].