Investigating The Use of Classical Machine Learning Techniques in Text Categorisation

Andrej Liadov Isobel Mahon Caroline Liu

Abstract

The problem of text categorisation has been an issue in searching and navigating text data on websites and other application. Text categorisation will allow for better organisation of indexes and overall more precise search results. The challenges faced in this project include getting raw text data, cleaning the text data and encoding the text data. The use of Gaussian kernelised SVM and logistic regression for categorising the text data was investigated. In doing so hyper-parameters of all the models used were tuned with k-fold cross validation.

Keywords: SVM, Gaussian Kernel, Text Categorisation, Logistic Regression, Wikipedia Text Data, Text Encoding, Word Embedding

1 Introduction

One of the difficulties in working with language processing is representing words or text as features that can be modelled mathematically. There are many different methods for encoding words. For example, one-hot encoding is used to create features based on whether a word in a corpus exists. This doesn't take into the term frequency or inverse document frequency.

In general texts can be categorised into certain topics. From a mathematical perspective, certain words or phrases are more likely to show up in a given category. For example, the phrase "string theory" is much more likely to appear in a physics category than a non-physics category. Classical machine learning methodologies can be used to generalise these patterns in text.

There are several classical machine learning algorithms that are used for the purpose of text categorisation. These include but are not limited to kernelised SVM, k-nearest neighbours and logistic regression. K-nearest neighbours and Naive-Bayes have the benefit of being scalable "big-data" techniques. Kernalised SVM on the other hand starts to slow down as the amount of data is increased.

2 Methodology

2.1 Acquiring the Data

The data we used was from Wikipedia. This data comes with categories, and the Wikipedia API allowed us to get a given number of pages from each category. We started with 20 pages from three categories as a test, and then got up to 500 pages from three categories, and finally up to 500 pages from ten categories.

Wikipedia categories have a tree-like structure, with each category having some number of pages and some number of subcategories. We were only interested in predicting the top level category. We started by getting all of the pages from each top level category, but there generally weren't very many, so we then got the pages from their subcategories. We tried to avoid favouring one subcategory over another by dividing up the number of pages yet to be gathered evenly between the subcategories. This technique, while better than getting all of the pages from one subcategory, does mean that smaller subcategories are likely to be over represented.

Because pages can be in multiple categories, we then removed any pages that appeared more than

once in our data.

2.2 Cleaning the Data

There were several Filtering methods used to categorise the text:

1. *Tokenisation*: The first language process applied was a tokeniser. Tokenisers transform texts into standalone words called tokens. The NLTK library offers a tokeniser.
2. *Stop Word Filter*: Tokens are then passed through a stop word filter with the default NLTK English stop word set. Stop words filters remove the words that do not contribute to the categorisation of the input texts. These words are typically very common, for example "the".
3. *Lemmatisation*: This is the process of taking a word and transforming it into a base form in contrast to a morphological inflection. This is different to stemming because stemming utilises simple rules that rarely output a word. Lemmatisation uses sophisticated techniques that take context into account and consequently return a real word. Often this results in better results for many applications(Edda Leopold). The downside of lemmatisation is that it takes much longer to run.
4. *N-gram Filter*: This was used to split the words into standalone tokens and the N adjacent words. This increases the amount of tokens greatly. N-grams allow the model to get a slight grasp of "context".

2.3 Creating Feature Vectors

The following methods were chosen to create features:

1. *Bag of Words*: Bag of words uses term frequency to encode words. This means that a word that appears often in a document will be encoded with a higher number than one that appears fewer times. This allows the model to give greater weight to words that appear more than once. It does not take into account whether the word is common though, so common words may mistakenly be given greater weight than words that give a better indication of the topic of the text.
2. *TF-IDF Vectorisation*: This is the process of applying the TF-IDF formula to the a bag of words vector. TF-IDF rewards terms for higher frequency but it also penalises terms if they feature in many documents. This prevents common, irrelevant terms like "and" from scoring well.
3. *Hashing Vectorisation*: Hash converts a collection of text to a matrix of token of occurrences, it implements the hashing trick to find the token string name to feature index mapping i.e. applies a hash function to term frequency counts. This is a very memory and time efficient method as this vectoriser does not store the resulting vocabulary, which also means that once hashed, the feature names are not retrievable.
4. *One-hot Encoding*: One hot encoding is a representation provides a method of vectorising categorical data as binary vectors that can be later utilised. This requires for the categorical values to be mapped to integer values. Each of the integer values converted are then transferred to a binary vector where the index of the integer is one and the rest are all zeros. As the dataset does not consist of categorical data, this method would not yield an accurate result.

2.4 Learning Methods

The main learning methods utilised were the kernelised SVM and logistic regression was used as a baseline. A dummy classifier was employed to create a point of comparison with legitimate machine learning algorithms. Kernelised SMV will try to create a largest minimum margin between two catgories. The reason for adding the kernel is to allow for non linearity in the decision boundary. In this project

we used a Gaussian kernel with an cross-validation optimised shape.

2.5 Cross-validation

Cross validation is done using k-folds cross validation. K-folds ensures that all the data is used when the hyper-parameters are tuned. The mean precision, f1 score and accuracy is calculated for each hyper parameter value. The mean precision values are plotted and the plots are used to determine the best values for hyper-parameters.

2.6 Analysis of The Model

Analysis of the models is done with the best models. Performance was quantified using precision on a test data split. A range of tables including metrics like AUC-ROC and precision were employed to ascertain the performance of the proper method and the baseline model.

3 Results

3.1 Metrics Used

3.1.1 Precision

Precision is the number of true positive divided by the sum of true positive and false positives.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

3.1.2 Recall

Recall is the proportion of total true positives returned.

$$Recall = \frac{TP}{TP + FP} \tag{2}$$

3.2 F1 Score

F1 Score is the harmonic mean of the precision and recall. This is a good combination of both metrics.

$$F1 = \frac{2TP}{TP + FN + FP + FN} \tag{3}$$

3.2.1 Area Under the Curve (AUC)

The AUC-ROC curve is the performance measurement for a classification problem, where ROC (Receiver Operating Characteristics) is the curve and AUC, the area under. The area under the curve indicates the capability of the model of distinguishing between classes, a higher value AUC (that is, nearer to 1) value means that the model is better at its predictions.

3.3 Cross Validation Analysis

3.3.1 TF-IDF Vectorisation

A dummy classifier with TF-IDF as an input vector got a precision of 0.09.

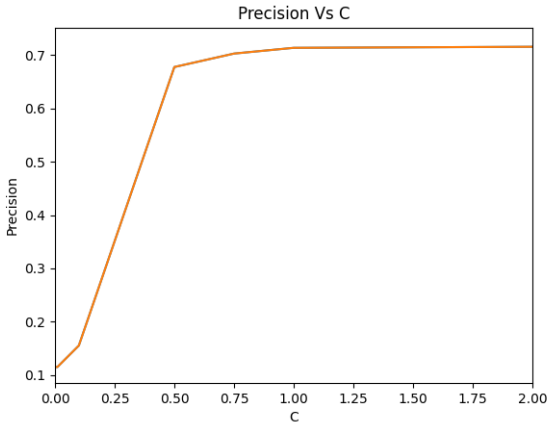

FIGURE 1: The optimal value for C is 0.75

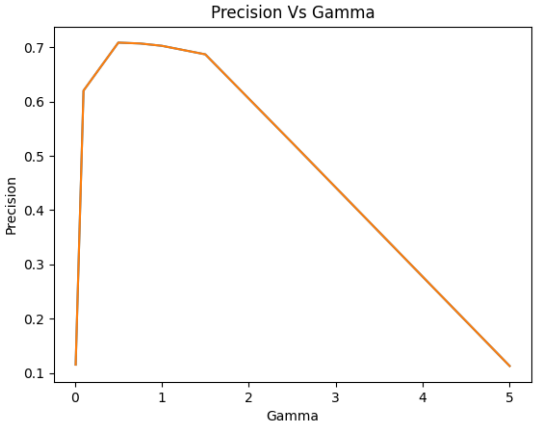

FIGURE 2: The optimal value for gamma is 1.0

3.3.2 Bag of Words

With Bag of Words encoding, an SVM with a C of 1 got a precision of 67%. On data with duplicate pages removed it had a precision of 73%. Since this was worse than the Tf-IDF encoding results, we decided to continue with TF-IDF encoding. The Dummy Classifier with Bag of Words encoding had an accuracy of 10.6%

3.3.3 Hashing Vectorisation

	precision	recall	f1-score	support
0	0.39	0.53	0.45	88
1	0.67	0.70	0.68	115
2	0.66	0.61	0.63	119
3	0.73	0.75	0.74	103
4	0.57	0.52	0.54	125
5	0.42	0.46	0.44	126
6	0.54	0.51	0.53	105
7	0.72	0.56	0.63	95
8	0.81	0.80	0.81	123
9	0.53	0.49	0.51	113
accuracy			0.59	1112
macro avg	0.60	0.59	0.60	1112
weighted avg	0.60	0.59	0.60	1112

FIGURE 3: The accuracy of hashing vectorisation is 0.59

With hashing vectorisation, the resultant accuracy on 10 categories of data yield a percentage of 59% and a precision of 60% with a C value of 1, which is much lower than projected and therefore TF-IDF was the optimal vectorisation method for the purpose of this scenario.

3.4 Performance

3.4.1 Kernelised SVM Model with Optimised Hyper-parameters

	precision	recall	f1-score	support	pred	AUC
0	0.505618	0.483871	0.494505	93.0	89.0	0.905503
1	0.813187	0.725490	0.766839	102.0	91.0	0.958637
2	0.931034	0.757009	0.835052	107.0	87.0	0.982243
3	0.815789	0.885714	0.849315	105.0	114.0	0.982337
4	0.631579	0.685714	0.657534	105.0	114.0	0.929714
5	0.612403	0.658333	0.634538	120.0	129.0	0.935122
6	0.585366	0.727273	0.648649	99.0	123.0	0.924924
7	0.794521	0.617021	0.694611	94.0	73.0	0.958976
8	0.886792	0.862385	0.874419	109.0	106.0	0.990053
9	0.710744	0.761062	0.735043	113.0	121.0	0.972314
avg / total	0.730335	0.720153	0.721874	1047.0	1047.0	0.958925

FIGURE 4: These are the results for all the categories on the test data.

	precision	recall	f1-score	support	pred	AUC
0	0.967846	0.943574	0.955556	319.0	311.0	0.996408
1	0.962319	0.973607	0.967930	341.0	345.0	0.999485
2	0.982332	0.948805	0.965278	293.0	283.0	0.999433
3	0.971338	0.977564	0.974441	312.0	314.0	0.999369
4	0.961310	0.925501	0.943066	349.0	336.0	0.997240
5	0.934718	0.943114	0.938897	334.0	337.0	0.996492
6	0.907738	0.962145	0.934150	317.0	336.0	0.996516
7	0.955823	0.940711	0.948207	253.0	249.0	0.998912
8	0.985207	0.982301	0.983752	339.0	338.0	0.999892
9	0.948454	0.975265	0.961672	283.0	291.0	0.999065
avg / total	0.957784	0.957325	0.957365	3140.0	3140.0	0.998424

FIGURE 5: These are the results for all the categories on the training data.

3.4.2 Logistic Regression with Optimised Hyper-parameters

	precision	recall	f1-score	support	pred	AUC
0	0.625000	0.485437	0.546448	103.0	80.0	0.912210
1	0.809524	0.817308	0.813397	104.0	105.0	0.963996
2	0.840426	0.797980	0.818653	99.0	94.0	0.980341
3	0.834783	0.834783	0.834783	115.0	115.0	0.979362
4	0.656250	0.711864	0.682927	118.0	128.0	0.939118
5	0.575472	0.535088	0.554545	114.0	106.0	0.911049
6	0.582734	0.764151	0.661224	106.0	139.0	0.931496
7	0.762500	0.743902	0.753086	82.0	80.0	0.974295
8	0.909091	0.918367	0.913706	98.0	99.0	0.996882
9	0.762376	0.712963	0.736842	108.0	101.0	0.980614
avg / total	0.732121	0.729704	0.728455	1047.0	1047.0	0.961960

FIGURE 6: These are the results for all the categories on the test data.

	precision	recall	f1-score	support	pred	AUC
0	0.903571	0.818770	0.859083	309.0	280.0	0.988723
1	0.910979	0.905605	0.908284	339.0	337.0	0.993614
2	0.950704	0.897010	0.923077	301.0	284.0	0.996777
3	0.892063	0.930464	0.910859	302.0	315.0	0.994876
4	0.874608	0.830357	0.851908	336.0	319.0	0.984955
5	0.851312	0.858824	0.855051	340.0	343.0	0.984737
6	0.787356	0.883871	0.832827	310.0	348.0	0.984953
7	0.910569	0.845283	0.876712	265.0	246.0	0.993200
8	0.943343	0.951429	0.947368	350.0	353.0	0.997966
9	0.844444	0.923611	0.882255	288.0	315.0	0.992399
avg / total	0.887152	0.885032	0.885166	3140.0	3140.0	0.991548

FIGURE 7: These are the results for all the categories on the train data.

This model was able to get 76% precision on data that had had duplicate pages removed.

4 Discussion

4.1 Data Quality

Wikipedia allows each page to be in multiple categories. This makes it harder for us to categorise them. Some pages would only appear in one top-level category, while others would be in multiple, sometimes in unpredictable ways. For instance, the page on the 2018 video game *Among Us* is in the category *Impact of the COVID-19 pandemic on the*

video game industry, which is in the top level category *Health*. *Among us* is also in the top level categories of *Culture*, *Computing*, as well as others. While most people would probably agree that the page on *Among Us* belongs in the categories of *Culture* and *Computing*, it is not directly related to *Health*. Categorisations like these make this a very difficult problem.

Wikipedia does not usually assign pages a main category, so it is difficult to filter pages by what might be considered a good example of it's category.

4.2 Contextual Disambiguation

Determining the actual meaning of the word depending on the particular context comes naturally to people. For instance, the word *band* has several defined meanings:

1. *band, paper band* - a flat thin string or loop of material used as a fastener
2. *band* - a stripe, line or elongated area of different composition to its surroundings
3. *band* - provide or fit something in the form of a strip or ring
4. *band* - to mark with a stripe or stripes
5. *band* - a group of musicians who play a specific genre of music.

Word sense disambiguation is a natural classification problem, which is when given a word and all its possible defined meanings, to classify the occurrence of the word in context into its defined meanings. Currently on English accuracy for homographs are generally above 90 % whilst in the finer-grained sense, lies around 50-60 %. This is taken into consideration as correct analysis of the meanings of words would aid the outputting of the correctly categorising the text in the dataset.

5 Summary

We created a Support Vector Machine with a variety of feature encodings to categorise text from Wikipedia pages. We compared these models to Logistic Regression classifiers trained on the same encodings. We found that TF-IDF was the best method of encoding of the ones we tried for this application. We also found that despite what some literature suggests(2), the Logistic Regression models were just as good as the Support Vector Machines. Logistic regression models also scale a lot better. In summary, the Logistic Regression model using TF-IDF encoding performed best for this application, with up to 76% precision.

References

[Edda Leopold] Edda Leopold, J. K. Text Categorization with Support Vector Machines. How to Represent Texts in Input Space?

[2] Pochet, N. & Suykens, J. (2006). Support vector machines versus logistic regression: Improving prospective performance in clinical decision-making. *Ultrasound in obstetrics gynecology : the official journal of the International Society of Ultrasound in Obstetrics and Gynecology*, *27*, 607–8, https://doi.org/10.1002/uog.2791.

Contributions

5.0.1 Andrej Liadov

1. Wrote the code in tfidf.py.
2. Performed cross validation on the TF-IDF encoded kernelised SVM and logistic regression model. (3.3)
3. Generated the tables in the performance section. (3.4.1, 3.4.2)

4. Wrote the section in the report corresponding to:
   (a) Abstract
   (b) Introduction (1)
   (c) Cleaning The data (2.2)
   (d) TFIDF Feature Encoding (2.3.2)
   (e) Learning methods (2.4)
   (f) Cross validation (2.5)
   (g) Analysis of the model (2.6)

5.0.2 Isobel Mahon

1. Wrote the code for the Bag of Words encoding
2. Wrote the code for the dummy classifier
3. Gathered data from the Wikidata API and wrote code to remove duplicate pages.

5.0.3 Caroline Liu

1. Wrote the code for one hot encoding and relevant sections
2. Wrote the code for hashing vectorisation and relevant sections

Github Link

The code can be found at: https://github.com/andrejliadov/MachineLearningGroupProject