TED Talk Topic Classification

AI Project Report

Jonathan Tooke
University of Cape Town
Cape Town, South Africa
tkxjon001@myuct.ac.za

Shane Weisz
University of Cape Town
Cape Town, South Africa
wszsha001@myuct.ac.za

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Supervised learning by classification**.

KEYWORDS

Text classification, natural language processing, machine learning, deep learning

1 PROBLEM FORMULATION

1.1 Overview

The aim of this project is to develop a machine-learning based text classification model using the TED talks dataset – a dataset containing raw text transcripts and metadata for 2467 TED talks. TED talks are short informative videos by various speakers under the slogan of "ideas worth spreading", originally intended to focus on the topics of Technology, Entertainment and Design. In particular, our aim is to develop a classifier that predicts which of these topics a given TED talk corresponds to, based solely on its raw text transcript.

1.2 Usefulness of this problem

In the context of the TED talk problem, the classifier would be useful to automate the keyword tagging of TED talks based solely on each talk's transcript. This could then be useful for a user that wanted to filter TED talks by the Technology, Education, and Design categories, which would save the user time as they would no longer have to scroll through all of the talks that are not relevant to what they are looking for. Developing a model that is capable of performing well on the TED classification problem would also serve as preliminary evidence that a similarly trained model could be effective on a similar problem in a different context (i.e. a different dataset and different classification task with similar features).

1.3 Dataset Description

The dataset consists of two CSV files – *main.csv* and *transcripts.csv*. *main.csv* consists of 2550 observations, each with a variety of metadata features including a description, a url and a set of keywords describing a given TED talk. *transcripts.csv* contains 2467 observations, each containing a text transcript and a url linking it to a row in *main.csv*. We thus merge the two datasets together using the common url feature, such that we work only with the 2467 TED talks observations that have a transcript associated with them.

1.4 Prediction Task

The machine learning problem that we have chosen to apply to the dataset is a multiclass classification problem. In particular, the prediction task is to predict, given the raw text transcript associated with a given TED talk as input, which of the following topic labels it belongs to as output: TED, TE, TD, ED, T, E, D or Other (where T denotes technology, E denotes entertainment, and D denotes design). These labels are attained by scanning the keywords metadata for a given TED talk and retaining which of these TED topics are present in its keywords. For example, a TED talk which has associated keywords of technology and design, but not entertainment, will be labelled TD.

1.5 Potential Difficulties

The problem faces several potential challenges that we have considered.

1.5.1 Imbalanced Dataset. The number of samples in each of the categories under consideration is highly imbalanced. As seen in Figure 1 below, the largest category is the *Other* category which contains 55% of the observations, while the smallest of the categories is the *ED* category which only contains 1.05% of the observations.

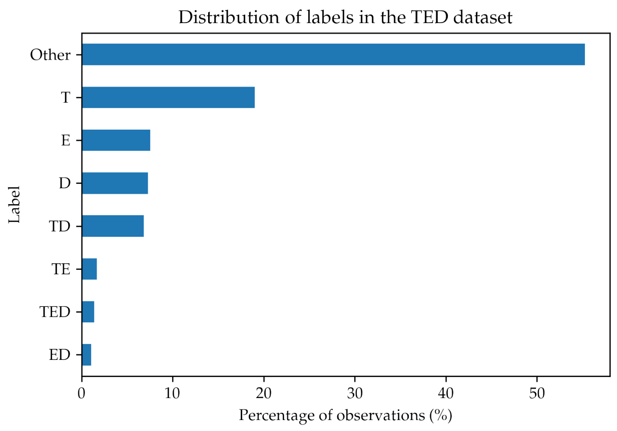

Figure 1: Distribution of labels for the 2467 TED talks

To ensure that this imbalance is taken into account when evaluating the models, we will use the F1-score as an evaluation metric (see Section 5.1 for more explanation in this regard). Moreover, to enable the neural networks to better handle the imbalance and consider the importance of the various classes more evenly during training, the loss function for each class will be weighted proportionally to the inverse of its number of samples.

1.5.2 Multiple Labels. The model may struggle with generalization in some cases as the classes under consideration are not independent of each other – with some of the smaller classes being a subset of the larger classes (e.g. T is in TE, TD, and TED, but each are their own class). Whilst the multi-class approach we adopt is capable of making these distinctions, an alternative approach to this problem could also have been to make use of multi-label classification – where a sigmoid function is used on the output layer, each output node represents one of the possible classes, and the set of output nodes that are activated represent the assigned set of labels.

1.5.3 Number of Samples. The dataset only contains 2467 samples which is not particularly large in the context of many deep learning problems. Each sample does, however, contain a large amount of data in the form of a lengthy text transcript. This means that despite the small number of samples, the explanatory power of each sample may be fairly extensive.

1.6 Ethical Considerations

The development of this artificial intelligence (AI) system does not pose serious ethical issues. If the model were to misclassify a TED talk, the damage caused would be limited to a user wasting time watching a talk that is not fully relevant to them. Moreover, if the model were to perform perfectly, it would not pose a threat to society as the classification of TED talks does not significantly influence the way in which people behave.

2 BASELINE

A baseline for this prediction task – the simplest possible approach to the problem – is to always output the label of the most popular class as the model's predictions. For this dataset, as can be seen in Figure 1 above, the most popular class is the *Other* label. Because the distribution of labels is highly imbalanced, the *Other* class constitutes 55.2% of the observations. Since we apply a *stratified* train-validation-test split (explained in Section 5.2), this class frequency distribution is preserved in the validation and test data – and as a result the baseline accuracy is 55.2%.

However, due to the class imbalance, the F1-score (the harmonic mean of precision and recall) is more pertinent as an evaluation metric (we will elaborate on this in Section 5.1 below). For our multiclass task, we we will apply macro-averaging to treat each class as equally important in the F1 calculation, such that the overall F1-score is evaluated as the arithmetic mean of each individual class's F1-score. Hence, using the baseline approach of predicting the most popular class, we obtain a baseline F1-score of

$$(2 \cdot \frac{1 \cdot 0.552}{1 + 0.552} + 0 + 0 + 0 + 0 + 0 + 0 + 0)/8 = 0.0889 = 8.89\%,$$

since the precision for the *Other* class by always predicting *Other* is 55.2%, the recall is 100%, and all other classes' precision and recall metrics will be 0.

3 MODEL DESIGN

We first outline how the model inputs will be represented, before explaining which models we will consider and comparatively evaluate, along with appropriate architecture choices.

3.1 Bag of n-grams Input Representation

After performing exploratory data analysis, we observed that whilst the number of samples in the dataset is small at 2467 total observations, the median number of words per observation is relatively high at approximately 2028 words per transcript. This results in a low ratio of number of samples to number of words per sample (compared to many typical text classification problems that have a larger number of samples with shorter text per sample). With this ratio being small, using a "bag-of-words" representation (or more generally, a "bag of n-grams" representation) as model inputs has been shown to often outperform or perform at least as well as sequence-based inputs that use word ordering information [3]. A possible explanation for this empirical result is that the bag of n-grams can provide a rich characterization of the sample for long samples, where for short samples sequential information would be more valuable.

As a result, we have chosen to use the bag of n-grams approach to represent the input for our models. In particular, we consider only unigrams and bigrams, and use the TF-IDF (Term Frequency Inverted Document Frequency) approach to assign decreasing weight / importance to more common terms (e.g. "the" and "an" which have little explanatory power). In particular, the procedure used to encode the input is as follows: (i) we first tokenize each transcript into individual words, (ii) then we vectorize each transcript by computing the term frequency for each word in the document, and multiply it by its inverse document frequency – and (iii) finally we perform feature selection by selecting the 20000 most informative n-grams based on their chi-squared association with the response (20000 is suggested as a safe upper bound for an appropriate number of features, based on multiple text classification experiments conducted across multiple datasets in [3]).

Consequently, the model inputs are stored as $n \times 20000$ matrices, where n is the number of samples in the train/validation/test sets, and the 20000 feature columns capture, for each transcript, its TF-IDF weighted frequency for each of the chosen 20000 n-grams.

3.2 Models Considered

There are numerous suitable models that can be applied to the classification task using input in the representation described above. A standard multi-layer perceptron (MLP) or standard feedforward neural network is the most appropriate neural network architecture for the task. More complex neural network architectures, such as recurrent neural networks or convolutional neural networks, would not be suitable for bag-of-words input representations, since this representation does not carry sequential or spatial information. The MLP can be compared, though, to multiple traditional machine learning architectures that also accept input in this format. In particular, we will compare the MLP to a support vector machine (SVM) classifier, a multinomial logistic regression approach, and a Naive Bayes classifier.

3.3 MLP Architecture

The MLPs evaluated in this project have either 1, 2, 3 or 4 hidden layers. Each hidden layer has 2^n nodes where $n \in [3, 8]$. For each model constructed as part of the hyperparameter search, the number of hidden layers and nodes in each hidden layer is selected randomly from those specified above. The output layer consists of 8 output nodes where each node represents a class under consideration. A softmax activation function is used on the output layer to generate probabilities of the observation coming from each of the respective classes.

4 MODEL VALIDATION

Here we outline how we optimize the performance of the models considered through hyperparameter searches, with particular emphasis on the design of the hyperparameter search for the MLP and measures taken to reduce overfitting.

4.1 Hyperparameter Search

The models trained in this project require a number of hyperparameters to be specified. These hyperparameters can be searched over to compare different configurations, with the goal of maximising the evaluation metric, i.e. the macro-averaged validation F1 score (see Section 5.1 for more details about this choice of metric). In the case of the multi-layer perceptron (MLP), the model architecture can also be searched over. To find the best set of parameters for both the hyperparameters and model architecture, a randomised grid search technique is employed. Randomised grid search is employed since it would not be computationally feasible to conduct an exhaustive grid search due to the extremely large number of hyperparameter combinations to search over. The randomised grid search samples from a set of hyperparameters and model architectures to train and evaluate a model. The results for the specific configuration are then recorded and saved for future analysis. Whilst this technique is employed for finetuning and optimizing all the models we consider (i.e. the multinomial logistic regression, Naive Bayes and SVM models), we outline in particular the hyperparameter search space considered for the randomized grid search for the MLP in the section below.

4.2 MLP Search Space

The table below outlines the search space of hyperparameters over which our randomised grid search took place to optimise the MLP's performance.

Hyperparameter	**Search space**
Number of layers	{1, 2, 2, 3, 3, 4}
Number of hidden layer nodes	{8, 16, 32, 64, 128, 256, 512}
Learning rate	{0.01, 0.005, 0.001, 0.0005}
Activation function	{ReLU}
Dropout	{0, 0.1, 0.25, 0.5, 0.5}
L2-regularization penalty	{0, 0, 0.001, 0.01, 0.1}

Table 1: Search space for the MLP hyperparameter randomised grid search

To conduct this procedure, 5000 different models were trained with hyperparameters sampled from the search space table (GPUs through Google Colaboratory were used to facilitate the training). The result of this hyperparameter search (i.e. the architecture and hyperparameters that attained the highest validation F1-score) will be reported in the results section below.

Note that once a number of hidden layers has been randomly chosen, the number of nodes for each layer is independently sampled from the search space as listed in the table.

4.3 Reducing overfitting

From our initial exploration of architectures and hyperparameter choices, we noticed that our MLP models were typically overfitting during training – attaining high training F1-scores that were not generalizing to similarly high validation F1-scores. We thus took numerous measures to mitigate this overfitting effect. In particular, we included various levels of both Dropout and L2-regularization during the hyperparameter search to try identify what level of these quantities would best reduce the effect of overfitting. Additionally, we used early-stopping to stop training once the validation F1-score had not improved for five epochs – and then outputted the model at the number of epochs that had attained the highest validation F1-score. Finally, we ensured small networks were also considered in the architecture search process, in the case of models having too high a capacity being a cause of overfitting.

5 EVALUATION

We now explain our choice of evaluation metrics, how we conducted the train-validation-test split, and report our experimental results.

5.1 Evaluation Metrics

Accuracy is one of the most popular metrics used for evaluation of classifiers, indicating the proportion of correct classifications relative to the total number of predictions made. This metric is a good indication of model performance if the dataset is balanced, however it can give a false representation of the model if the dataset is highly unbalanced. For example, if 99% of a dataset has one label and 1% another, a model that always predicts the first label will achieve 99% accuracy even though it never classifies the second label correctly.

The F1 score is often used to solve this problem, calculated as the harmonic mean of the precision and recall metrics (where precision for a class is computed as the number of true positives divided by the number of predicted positives, and recall is the number of true positives divided by the total number of positives). The F1 score is a value between 0 and 1 and can be interpreted as a percentage. For multi-class classification problems, each measure is computed for each label present in the data, with a weighted average of the labels' metrics then taken as overall measures of the model's performance. To account for class imbalances, macro-averaging can be adopted as the weighting strategy – taking the arithmetic mean of each label's F1 scores – so as not to skew scores in favour of popular classes. Hence we use the F1 score with macro-averaging approach as our key evaluation metric.

5.2 Training/Validation/Test split

We created training, validation, and testing datasets by randomly sampling from the 2467 observations in an 60-20-20 split. The split is stratified by class labels in order to preserve the label proportions in each dataset (this is especially important due to the class imbalance).

The training set is used for training the models, whilst the validation data is used as an estimate of how the models will perform on the test set so as to guide the hyperparameter tuning process. Finally, the unseen test data serves as an unbiased indication of the models' out-of-sample performance and ability to generalize to new data.

5.3 Results

5.3.1 MLP hyperparameter search results. The hyperparameters and associated evaluation metrics results for all models considered in the hyperparameter search process were saved to a CSV file. In particular, the MLP architecture and hyperparameter combination that attained the highest validation F1-score is as follows.

Hyperparameter	**Outcome**
Number of layers	3
Hidden layer nodes per layer	[32, 512, 16]
Learning rate	0.001
Activation function	ReLU
Dropout	0.5
L2-regularization penalty	0

Table 2: Hyperparameters for the MLP that attained the highest validation F1-score

It is interesting to note that this model used a dropout rate of 0.5 as its regularization mechanism – and did not make use of L2-regularization.

This model is then chosen to be compared with the traditional machine learning models in the next subsection.

5.3.2 Comparison of models on validation set. The F1-score on the validation set of the models considered are listed in the table below. The validation accuracy scores are also included, but our focus is on the F1-score due to the class imbalance. Note that NB stands for Naive Bayes classifier and LR for the multinomial logistic regression classifier.

	Baseline	**SVM**	**LR**	**NB**	**MLP**
Accuracy (%)	55.26	63.97	65.18	65.59	69.23
F1 (%)	8.90	23.29	26.98	29.08	42.25

Table 3: Comparison of F1-scores and accuracies on the validation set

The MLP model thus outperforms the SVM, Naive Bayes, and multinomial logistic regression classifiers on the basis of F1-score. As a result, the MLP is chosen as our final model. It will be evaluated on the test set, and its performance analysed in the next section.

We note also that the Naive Bayes classifier was the most successful traditional machine learning model, with multinomial logistic regression the next best approach – both outperforming the SVM. Moreover, we notice that the models' scores are higher than the baselines, indicative of the success of the models 'learning'.

6 ANALYSIS OF MODEL PERFORMANCE

In this section we analyze the performance of our final MLP model on the test set, including its potential shortcomings.

6.1 Final Evaluation on Test Set

We now perform our final evaluation of our chosen MLP model on the test set. The chosen MLP achieves a test F1 score of 35.56% and a test accuracy of 64.17%. This can be interpreted as giving an indication of our model's out-of-sample performance and ability to generalize to completely unseen data. Accordingly, given new TED talk transcripts as input, we would thus expect the model to similarly attain an accuracy and F1-score in the regions of 64.17% and 35.56% respectively. These are both higher than our baseline test accuracy and F1-scores (55.2% and 8.89%), which shows that our model has learnt valuable information to distinguish between classes.

We also note that the test F1 and accuracy scores are both a bit lower than their validation scores. This variation could be explained by the small size of the dataset, which has restricted the extent to which the validation set is fully representative of the test set.

6.2 Prediction Analysis

Figure 2 shows the accuracy, precision, recall, and F1 scores for each of the TED classes as calculated by the predictions on the test set.

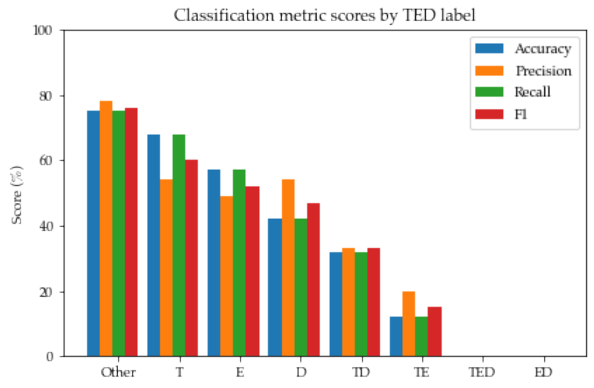

Figure 2: Bar graph comparing accuracy, precision, recall and F1-scores for the final MLP model on the test set

The largest two classes, *Other* and *T* outperform the rest of the classes across almost all metrics. This could be explained by the fact that there was more data to train on, enabling the model to better learn the class patterns. The only exception here, is *D*, which attains the second highest precision score (second to *Other*). This means that the percentage of times we are correct when we predict *Design* is higher than that for any other class but *Other*. We can interpret this as meaning we can be more confident in the model's predictions when it predicts *Design* than for most of the other classes.

The two smallest classes, *TED* and *ED* achieve scores of zero across all metrics. This could be explained by how little data there was to train on, restricting the ability of the model to learn the class patterns.

For each of the classes, the accuracy, precision, recall and F1 metrics produce roughly similar scores. This shows that the main difference between the final aggregated F1 and accuracy metrics (64.17% vs 35.56%) is explained by the equal weighting of the classes in the macro-weighted F1-score calculation, as opposed to accuracy which just compares the final predictions for the whole test set to the true labels (and hence does not account for the class imbalance).

We now more specifically analyse which classes were mistaken for which other classes through the confusion matrix in Figure 3 below. In particular, the confusion matrix shows the difference between the number of times the true label was predicted for each class, and the number of times that each of the other classes were incorrectly predicted instead.

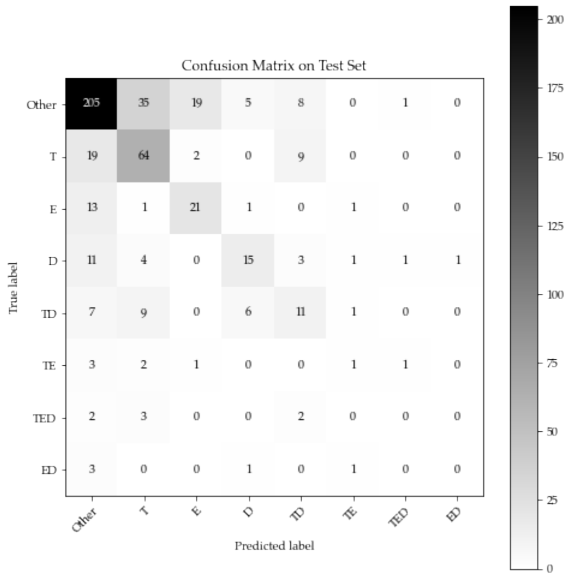

Figure 3: Confusion matrix for the final MLP model on the test set

Our first observation is that the high numbers on the main diagonal depicts when a true class label is correctly predicted by the model. In other words, the values off the main diagonal are the interesting ones to consider since these explain where the model is making mistakes. Many of the common misclassifications pertain to *Other* – either *Other* being predicted when it is not, or *Other* failing to be predicted when it is actually the correct label. This makes sense, given that *Other* was the most frequent class in the dataset.

It is interesting to note that the only misclassification pairs that have values above 5, are those which are only off by one of the classes involved (e.g. *D* predicted when the true class is *TD*, or *TD* being predicted when the true class is *T*). This suggests that the model is not making many big mistakes in terms of being completely wrong about all the categories involved. A classification metric that incorporates this (for example, penalizing a prediction of *T* less heavily than a prediction of *E* when the true label is *TD*) could thus perhaps be useful for future consideration to better capture this idea.

6.3 Potential shortcomings

The bag-of-words approach used as the input representation for the text transcripts results in sequential information being lost by the models considered. A sequential model, such as a Long Short-Term Memory (LSTM) recurrent neural network, would thus be a viable option to consider in order to capture this information, and could potentially result in improved performance. We do note however that we did also consider bigrams (not just unigrams) for the bag-of-words input representation for the model, which does mean that information pertaining to orderings between words pairs were retained.

Additionally, as discussed earlier, treating the problem as a multi-label classification problem could also potentially improve performance.

7 SOFTWARE

The MLP neural networks were implemented in Python through the Keras Sequential API [2] with a Tensorflow 2.0 backend [1], and trained using GPUs through Google Colaboratory. Keras was chosen as our deep learning framework because of its ease of use and modularity for model building, without reducing flexibility [4]. This allowed for rapid model development, and enabled more time to be spent on experimentation. The SVM, multinomial logistic regression, and Naive Bayes classifiers were implemented in Python using the scikit-learn API [5]. Similar to the rationale for choosing Keras for developing the neural networks, scikit-learn was chosen for its simplicity and efficiency in building these machine learning models for multi-class classification. More generally, the Python library NumPy was used for numerical computation, pandas for data manipulation and analysis, and Matplotlib for visualizations.

8 REPRODUCIBILITY

Due to the importance of reproducibility as part of the scientific process, numerous steps have been taken to ensure the experiments and the corresponding results could be reproduced. Firstly, the code used to build and evaluate the models is open-source and has been thoroughly documented – as such, the code can be rerun to replicate the full workflow that we adopted. Random seeds were specified for when random numbers were used (e.g. the train-validation-test split). The final chosen MLP model has been saved as a .pb file such that its reported prediction results and evaluation could easily be re-obtained, whilst its hyperparameters configuration is specified in Section 5.3 above. The procedure for producing these test results has been scripted and can be rerun to replicate the final results. Furthermore, the test data and final model's predictions on the test set have also been saved.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] François Chollet et al. 2015. Keras. https://keras.io. Accessed: 2020-11-06.

[3] GoogleDevelopers. 2019. https://developers.google.com/machine-learning/guides/text-classification/step-2-5 Accessed: 2020-11-06.

[4] Keras. 2020. https://keras.io/why_keras/ Accessed: 2020-11-06.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.