COMSM0089 Introduction to Data Analytics Coursework

QJ22919/ 2334033

|  |  |
| --- | --- |
| Task 1: Sentiment Classification | |
| * 1. **Implement & Train** | |
| **Overview|** | |
| The FiQA dataset consists of JSON files containing Tweets and Headlines along with a label indicating whether the content is mainly positive or negative.  Pre-processing splits the JSON into a list of documents and a corresponding list of labels. The document list is processed to remove common words/ punctuation and add negations to negative word pairs (e.g., ‘*didn’t like’*). These documents are split into tokens (individual words in this case)- this is the ‘Bag of Words’ representation.  A word vector is created- all the unique words in the dataset are combined to create a vocabulary for the document. For each document in the corpus a vector is created which is the same length as the vocabulary with a count of the occurrence of the word correspondingly located in the vocabulary.  A Logistic Regression Classifier is trained by finding the optimum weight for each feature in the training word vectors that best predicts the document class ( gold label), this means it is a discriminative model. Cross validation is used to generate possible values of weighting and then gradient descent is used to find the minimum value of the loss function ( a cross entropy calculation).  Logistic Regression is a binary classification method so Multiple Logistic Regression is performed to produce the Positive, Negative and Neutral models.  The dot product of the unseen word vectors and the weighting vector (from the training/ fit stage) is then classified using a Sigmoid function which scales the probability ( an unbounded real number) to a value between 0 and 1.  The performance of the model is assessed by comparing the predictions to the (human) generated labels in the test set. | |
| **Data Pre-Processing** | |
| The data loader iterates through each individual JSON file (Tweet) in the FiQA dataset and adds each’s text and label to a corresponding list.  The list containing the labels for all tweets is then converted from its continuous value ( negative value for negative sentiment and vice versa) into discrete values.  This initial threshold of -0.2 to 0.2 for neutral gave the following distribution of tweets:   |  |  |  |  | | --- | --- | --- | --- | | **Continuous Label** | **Output Label** | **Sentiment** | **Proportion of Dataset** | | Less than -0.2 | 0 | Negative | 28% | | -0.2 & 0.2 | 1 | Neutral | 18% | | More than 0.2 | 2 | Positive | 54% |   From the initial evaluation of the models, it was clear that the threshold of -0.2 to 0.2 for neutral gave a model that was heavily biased. Experimentation with the threshold showed neutral between -0.25 & 0.32 produced a model with more intuitive performance   |  |  |  |  | | --- | --- | --- | --- | | **Continuous Label** | **Output Label** | **Sentiment** | **Proportion of Dataset** | | Less than -0..25 | 0 | Negative | 25% | | -0.25 & 0.32 | 1 | Neutral | 32% | | More than 0.32 | 2 | Positive | 42% | | |
|  | |
| A function *add\_negation* was created which uses REGEX to identify words preceded by *not* or *never* and words which end with *n’t.* The function appends a *not­\_* prefix to these words so that the negative meaning in captured in the word vector. | |
|  | |
| The tweets are split into test, validation and training sets. The training set is used to train the model by providing both the content of the tweet and its label. A random state value was set for the split so that the evaluation is rep  The validation set is not included when the model is trained so it can be used to give an indication of the model’s performance when predicting the classification. The predicted values (0/ 1/ 2) for each of the tweets in the Val set is compared to its actual label (which is from the gold data classified by humans).  There are hyperparameters in the Logistic Regression Classifier model but the validation set was used to adjust the threshold of the initial conversion from to continuous to discrete labels.  Once the data split was believed to be optimum the test data was given to the model to predict. This ensures that the data has not been optimised to fit the test set which would give an unrealistic impression of the model’s performance on other unseen data. | |
| Tokenize from the NLTK package is used in the CountVectoriser to generate word vectors for the Test, Validation and Training sets. The Tokenize function first creates a vocabulary of all the tokens (individual words, punctuation and special features such as email addresses) in all of the tweets within the data split (or corpus). This is the ‘vectoriser.fit’ stage and is only done against the training dataset as both the validation and test sets are the unseen data for model validation.  The CountVectoriser, for each tweet in a dataset, creates a vector the same length as the vocabulary with a count of the times that correspondingly placed word was seen in the individual tweet. This results in a sparse matrix as wide as the vocabulary and with the same number of rows as tweets in the data set. The output form the CountVectoriser is a Bag of Words (BoW) as the vector for each tweet represents the frequency of a word from the vocabulary being in that tweet but not the order ( and therefore relationship) between the words in the tweet.  Normalisation: There are several techniques that can be applied to standardize the tokens in the vocabulary such as case folding all words to lower case or substituting emoji for text. Only folding to lower case was used here.  Lemmatizing the words to their root form (reading to read etc.) or steming words (use regex to drop prefixes & suffixes such as assuming an ‘s’ indicates a plural) can be useful, particularly in very large datasets, but they also remove some information. I have decided to exclude this step as the data set is small so does not require the simpler processing this gives.  The pre-processing of the 754-document training set reduced the vocabulary from 3349 to 3277 tokens. The compressed sparse row format of storing the word vectors reduced from 10510 to 8351 stored elements. | |
| **1.2 Model Evaluation** | |
| **Performance Metrics** | |
| The sckikit-learn metrics accuracy\_score, precision\_score, recall\_score and f1\_score were used to evaluate the models. The classification\_report was used as this gives all the metrics above and the macro values ( value for each class). The weighted average is also given which is useful as it considers the support ( number of true instances for each label)  Accuracy is misleading in this case as there is an imbalance in the number of classes. Recall is a better metric as it incorporates False negatives,  COnfuseion matrices were also used as this aids the understanddibilty of the model more than just metrics.  The  classification\_report | |
|  | |
| Naïve Bayes was considered initially, this is a generative, probabilistic model which learns the probability of individual tokens being associated with a particular class of document.  Naïve Bayes only considers a single token’s probability so the likelihood of each token is conditionally independent of any other.  As can be seen from the metrics below | |
|  |  |
| Naïve Bayes Without Data Processing |  |
|  |  |
| precision recall f1-score support  0 0.61 0.30 0.40 37  1 0.60 0.12 0.21 24  2 0.60 0.92 0.73 73  accuracy 0.60 134  macro avg 0.60 0.45 0.45 134  weighted avg 0.61 0.60 0.54 134 |  |
|  | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **Logistic Regression Classifier**  No data processing |  |
| precision recall f1-score support  0 0.59 0.46 0.52 37  1 0.38 0.21 0.27 24  2 0.68 0.86 0.76 73  accuracy 0.63 134  macro avg 0.55 0.51 0.52 134  weighted avg 0.60 0.63 0.61 134 |  |
|  |  |
| With Data Processing |  |
|  |  |
| precision recall f1-score support  0 0.79 0.59 0.68 37  1 0.60 0.12 0.21 24  2 0.69 0.96 0.80 73  accuracy 0.71 134  macro avg 0.69 0.56 0.56 134  weighted avg 0.70 0.71 0.66 134 |  |
| Data processing, in particular the negation improved the model for both negative and positive sentiment. The largest error is for neutral statements being classified as positive | |
|  |  |
|  |  |
| With neutral between 0.32 & - 0.25: |  |
| Number of instances: 1111  Number of labels: 1111  Number of negative labels: 283  Number of neutral labels: 359  Number of positive labels: 469 | |
|  |  |
|  |  |
|  |  |
| precision recall f1-score support  0 0.61 0.50 0.55 34  1 0.55 0.51 0.53 43  2 0.56 0.65 0.60 57  accuracy 0.57 134  macro avg 0.57 0.55 0.56 134  weighted avg 0.57 0.57 0.57 134 |  |
|  |  |
| By adjusting the threshold scores the split of the data set is adjusted. The actual proportion of positive, neutral and negative tagged tweets is unknown but the confusion matrix above possibly indicates a realistic estimation.  Rather than the model being biased towards positive sentiment as before, each category has more correctly classified than not.  Also, there is a graduation between values which is more intuitive than the earlier model. | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |