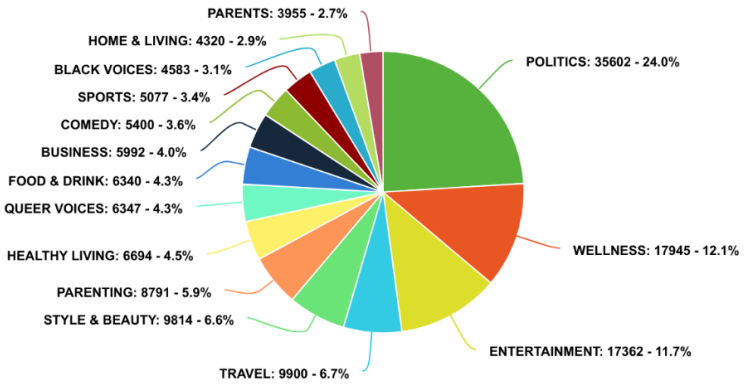
Assignment 2: Text Classification

Kristjan Gamboc

2024-01-14

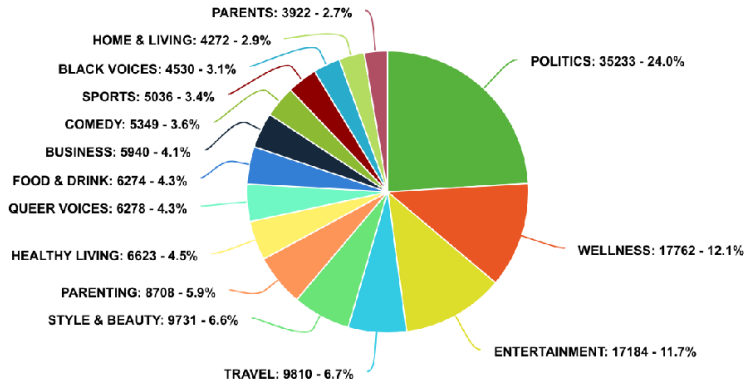
1 Data and preprocessing

The data used in this assignment is the 'News Category Dataset' by Rishabh Misra. It contains information about 148122 articles from the year 2012 to 2022 obtained from HuffPost. The information provided regarding the articles are the link to the article, the headline, the category, a short description, names of the author(s) and the date posted. The goal of the assignment was to train and compare different models for the classification of these articles into the categories provided. In total there are 15 categories. The number of articles for each category and the share of each in relation to the whole is shown in the graph below:



There was some missing data and some noise. The first issue was with the links. Certain articles were published before the HuffPost changed its name from Huffington Post and the link was different. Another issue was that some articles were originally posted somewhere else, and the links were concatenated with the old website link for Huffington Post, for example: https://www.huffingtonpost.comhttp://www.independent.co.uk/voices/munroe-bergdorf-loreal-dropped-anti-white-rant-institutional-racism-trans-a7924196.html. The data was cleaned up by first removing the link to Huffington Post if necessary, and then the website name was extracted from the link, for example the link above was reduced to just independent. This was done since the rest of the link contained either the title of the article or random digits, which would add nothing to the training data.

The next issue was missing data. Three attributes had data in some entries missing: headlines, short description and authors. The entries with no authors were attributed to "Jane Doe" and the entries missing either the headline or short description were removed. This was done because the headline and short description of the article are much more vital to the categorization of the article and the amount of entries with either no headlines or short descriptions amounted to less than 1%, while the entries with no authors comprised over 16% of the total data. To ensure that the removal of articles was mostly even across categories the amounts and shares of categories in the data were calculated again:



After the noise and missing data were dealt with the data was preprocessed for use. First sentences were converted into lowercase and strings were tokenized into separate words. After that punctuation and stopwords were removed entirely. After that the words was stemmed and lemmatized into its base root. After that the words of each attribute were concatenated with spacing to create a string rather than an array. At the start of each model the separate attributes were concatenated together into a single for every article.

2 Classification models

2.1 Vectorization

The first step in each model was to vectorize the data. TF-IDF was used for this purpose. TF-IDF, or Term Frequency-Inverse Document Frequency, is a numerical statistic used to evaluate the importance of a work within the text. First the term frequency (TF) is computed. The TF measures how often a word appears in a document:

$$TF(w,t) = \frac{\text{amount of times word } w \text{ appears in text t}}{\text{total amount of terms in text } t}$$

After that the inverse document frequency (IDF) is computed. This measures the significance of each word across a collection (corpus) of texts. Words that appear more frequently are penalized, while words that are rarer are given more weight:

$$IDF(w,C) = \log(\frac{\text{total amount of texts in corpus } C}{\text{amount of texts containing word } w} + 1)$$

TF-IDF is the product of the above frequencies. This gives a higher weight to terms that are frequent in a specific document while rare in the entire corpus:

$$TF - IDF(w,t,C) = TF(w,t) \cdot IDF(w,C)$$

2.2 Decision tree

The decision tree is the simplest model used in this project. The goal is to generate a tree where each node represents a test based on an attribute. Each leaf represents the decision - the class the specific case gets classified into. The tree was built using the *sklearn* library. First the data for each entry except the category was concatenated into a single string and two arrays are built: the first contains the strings used for training, the second contains the categories for each string. The first step is to vectorize using $TF-IDF$

the training data and after that the data is split into two sections, one for training, which consists of 80% of the data and one for testing, which consists of the other 20%. After that the model is built and tested:

```
X_vector = vectorizer.fit_transform(X)
X_train, X_test, Y_train, Y_test = train_test_split( *arrays: X_vector, Y, test_size=0.2, random_state=42)

decision_tree_classifier = DecisionTreeClassifier()
decision_tree_classifier.fit(X_train, Y_train)

Y_predictions = decision_tree_classifier.predict(X_test)
```

The tree reached a depth of 528. Overall the results for this model are the least accurate and precise. It achieved an accuracy score of only 73% and the model is not very precise, reaching only 49% as its lowest score. The other scores can be found in the table below:

	precision	recall	f1-score
BLACK VOICES	0.54	0.49	0.51
BUSINESS	0.49	0.47	0.48
COMEDY	0.60	0.58	0.59
ENTERTAINMENT	0.71	0.73	0.72
FOOD & DRINK	0.79	0.76	0.77
HEALTHY LIVING	0.56	0.58	0.57
HOME & LIVING	0.81	0.76	0.78
PARENTING	0.68	0.67	0.67
PARENTS	0.63	0.57	0.60
POLITICS	0.80	0.85	0.83
QUEER VOICES	0.78	0.69	0.73
SPORTS	0.63	0.56	0.59
STYLE & BEAUTY	0.82	0.80	0.81
TRAVEL	0.75	0.73	0.74
WELLNESS	0.75	0.79	0.77

Additionally, the model received an accuracy score of 73%.

2.3 K-nearest neighbors

This model finds the k nearest neighbors and classifies each entry into the most common class among those neighbors. The data is first vectorized and split, but this time more than one model is built, changing the hyperparameter k. The values for k used were between 3 and 7:

```
for i in [3, 4, 5, 6, 7]:
    knn_classifier = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    knn_classifier.fit(X_train, Y_train)

    Y_predictions = knn_classifier.predict(X_test)
    accuracy = accuracy_score(Y_test, Y_predictions)

    report = classification_report(Y_test, Y_predictions)
    print(report)
```

The accuracy score did not change much with the increase of k, going from 74% at $k = 3$ to 77% at $k = 7$. The payoff for the increasing time complexity of the model was the increase of precision across classes. The lowest precision at $k = 3$ was 46%, while the lowest precision score at $k = 7$ was 61%.

2.4 Random forest

Random forest is an enseble method that combines the predictions of multiple decision trees train on different subsets of the training data, which reduces overfitting. Once the model is trained the predicted class label is decided by adding the results from each tree and the class assigned is the majority vote. The model was further improved by adding a grid search on the model's hyperparameters. Since the original decision tree reached depth 528 the parameters were set such that the most complex hyperparameters were the same as in the original tree. This was done to see if a simpler model could be found. The hyperparameters tested were max_depth, which limits the depth of the tree, $n\_estimators$, which decides the amount of decision trees in each forest and $max\_features$, which is the amount of features considered when looking for the best split when the trees are being built. The last addition to the model was cross validation. The data was split into three folds at each iteration of the grid search.

```
grid = {
    'randomforestclassifier__max_features': ['sqrt', 'log2', None],
    'randomforestclassifier__n_estimators': [50, 75, 100],
    'randomforestclassifier__max_depth': [200, 400, None]
}

pipeline = make_pipeline( *steps: TfidfVectorizer(), RandomForestClassifier())
grid_search = GridSearchCV(pipeline, grid, cv=3, n_jobs=-1)

grid_search.fit(X_train, Y_train)
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

Y_predictions = best_model.predict(X_test)
```

The parameters of the best model were expected. The maximum depth was unlimited and the number of forests was the highest. Additionally, the best setting for the $max\_features$ hyperparameter was $sqrt$, the default one. The results of this model were better than both the k-nearest neighbors and the single decision tree models.

	precision	recall	f1-score	support
BLACK VOICES	0.84	0.45	0.59	926
BUSINESS	0.84	0.45	0.58	1191
COMEDY	0.79	0.58	0.67	1046
ENTERTAINMENT	0.80	0.81	0.81	3428
FOOD & DRINK	0.87	0.85	0.86	1312
HEALTHY LIVING	0.79	0.55	0.65	1338
HOME & LIVING	0.91	0.79	0.85	811
PARENTING	0.77	0.75	0.76	1804
PARENTS	0.86	0.60	0.71	781
POLITICS	0.78	0.97	0.86	6906
QUEER VOICES	0.92	0.71	0.80	1254
SPORTS	0.86	0.64	0.73	1014
STYLE & BEAUTY	0.88	0.88	0.88	1911
TRAVEL	0.87	0.82	0.84	2014
WELLNESS	0.71	0.92	0.81	3595
accuracy			0.80	29331

The accuracy of this model is the highest so far and the lower precision for a category was 71%, much higher that the other two models.

2.5 XGBoosting

XGBoost is a gradient learning model that combines a number of weaker decision trees into a stronger one. Each tree is trained on a subset of the training data and the predictions are combined into the final prediciton. No hyperparameter optimization was done with this model. Uniquely for this model only the vector of classes had to be numerically encoded for it to work with the *xgboost* Python library.

```
X_vector = vectorizer.fit_transform(X)
Y_encoded = encoder.fit_transform(Y)
X_train, X_test, Y_train, Y_test = train_test_split( *arrays: X_vector, Y_encoded, test_size=0.2, random_state=42)

xgboost_classifier = xbg.XGBClassifier(n_jobs=-1)
xgboost_classifier.fit(X_train, Y_train)

Y_predictions = xgboost_classifier.predict(X_test)
Y_predictions = encoder.inverse_transform(Y_predictions)
Y_test = encoder.inverse_transform(Y_test)
```

Even with no hyperparameter optimization the model achieved a better accuracy score and overall higher precision scores than all the previous models.

	precision	recall	f1-score	support
BLACK VOICES	0.77	0.59	0.67	926
BUSINESS	0.78	0.60	0.68	1191
COMEDY	0.78	0.68	0.73	1046
ENTERTAINMENT	0.85	0.80	0.82	3428
FOOD & DRINK	0.90	0.84	0.87	1312
HEALTHY LIVING	0.78	0.68	0.72	1338
HOME & LIVING	0.90	0.82	0.86	811
PARENTING	0.83	0.79	0.81	1804
PARENTS	0.81	0.74	0.77	781
POLITICS	0.80	0.94	0.87	6906
QUEER VOICES	0.90	0.80	0.85	1254
SPORTS	0.86	0.71	0.78	1014
STYLE & BEAUTY	0.90	0.88	0.89	1911
TRAVEL	0.88	0.82	0.85	2014
WELLNESS	0.78	0.90	0.83	3595
accuracy			0.82	29331

3 Results and improvement

The precision scores for each class are interesting. There seems to be no link between the share of the total data each class takes up and the precision score. For example, Black voices is the third least represented category, but it reached a precision score of 84% with the forest model, while the Wellness category, which is the second most represented category, reached a precision score of only 71%. The precision scores for each category also varied a lot between models. The Black voices category, with which the forest model was very

precise, only reached a 54% precision score. The single decision tree model was however more precise with the Wellness category, reaching 75% over the forest's 71%.

This project could be improved in multiple ways. One way would be to add more models for comparison. Another would be testing each model not only with the $TF - IDF$ but other methods too and compare the effect this has on the results. The forest model could be better trained by further increasing the amount of trees in the forest and splitting the data into more folds would probably result in a better model, however the machine the code was run on would run out of memory with a higher value for the parameter. Lastly, the XGBoost method performed the best out of the four without any hyperparameter optimizaton. Using random search for example could lead to even better results. Data preprocessing could be changed to use either only tokenization or only lemmatization, since using both is mostly redundant.