Capstone Project
Machine Learning Engineer Nanodegree

Roman Levitas
May 6, 2017

Text Classification on Stack Exchange Questions

DEFINITION

Overview

The aim of this investigation will be to build a text classifier, but the twist is that the classifier will be trained on topics independent of the testing set, making this a classic Natural Language Processing (NLP) problem with a transferred learning approach.

The data to be analyzed is a subset of the Stack Exchange Data Dump[1] published on December 15, 2016 focusing on 6 topics in particular: biology, cooking, cryptography, diy, robotics, and travel. The testing set is a 7th topic--physics.

The history of Stack Exchange's question-and-answer format dates back to as recent as 2008 when StackOverflow was created, the originating leg of the Stack Exchange network, which allows users to crowdsource knowledge on the topic of computer science/software engineering.

The data dump contains the titles, text, (the input set) and tags (which is what we will be predicting via the classifier) of Stack Exchange questions in the form of a comma separated value (CSV) document.

The classifier to be designed will have knowledge of 6 seemingly independent topics. The idea of there being a common thread or unifying theme, if looked at from the perspective of physics, among these topics is the goal of this investigation.

Problem Statement

The problem at hand is defined by the Kaggle team's competition title: 'Transfer Learning on Stack Exchange Tags' [2] which aims to "Predict tags from models trained on unrelated topics". Specifically, predicting tags for physics questions after training the classifier on questions provided in the 6 different fields mentioned above.

Underlying in this approach is the presumption that physics is the unifying concept, so the investigation is itself an exercise into a problem without a definite answer, but if there is a correlation that can be found, it can certainly shed light and illuminate the grey area in question: Is Physics at the heart of everything?
Predictions can be compared against the physics questions' actual tags and thus the model can be ranked on the correctness of its categorization using metrics discussed below.

[1] archive.org/details/stackexchange
[2] kaggle.com/c/transfer-learning-on-stack-exchange-tags

Metrics

The evaluation metric for this competition is Mean F_1-Score[3]. The F_1 score measures accuracy using the statistical notions of precision (p) and recall (r). Precision is the ratio of true positives (t_p) to all predicted positives (t_p + f_p). Recall is the ratio of true positives to all actual positives (t_p + f_n).

The F_1 score is given by:

$$F_1 = \frac{2pr}{p + r}$$

where:

$$p = \frac{t_p}{t_p + f_p}$$

and

$$r = \frac{t_p}{t_p + f_n}$$

In the multi-class and multi-label case, Mean F_1-Score is the weighted average of the F_1 score of each class. [4]

To better understand this, an example is given below:
On the left is an array of predicted tags and on the right are the correct, assigned tags.

```
Prediction => Actual

[quantum-mechanics, electron, current] =>[ electron, ampere, quantum-mechanics, voltage]
```

Correct predictions are:

`quantum-mechanics`: t_p
`electron`: t_p

while the rest are either false negatives or false positives:

`current`: f_p *(since it appeared in the prediction but not the actual)*
`ampere`: f_n *(since it was missing in the prediction, but supposed to be present according to actual)*
`voltage`: f_n

2 t_p, $1 f_n$, 2 f_n

p = 2/3 , r = 1/2

F_1= 4/7 = 0.57

[3] kaggle.com/c/transfer-learning-on-stack-exchange-tags#evaluation
[4] scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

ANALYSIS

Data Exploration

The input dataset is 6 separate CSV files for biology, cooking, cryptography, diy, robotics, and travel[5]. Each row of the data contains the title, text, and associated tags of a question. The export of the data from StackExchange supported html markdown for the text column so there will be a prerequisite data cleaning step to take into account markdown formatting and html tags/elements.

The first five rows of the biology CSV are shown:

```
>> biology.head(5)

   id  title
0   1  What is the criticality of the ribosome bindin...
1   2  How is RNAse contamination in RNA based experi...
2   3      Are lymphocyte sizes clustered in two groups?
3   4  How long does antibiotic-dosed LB maintain goo...
4   5        Is exon order always preserved in splicing?

   content
0  <p>In prokaryotic translation, how critical fo...
1  <p>Does anyone have any suggestions to prevent...
2  <p>Tortora writes in <em>Principles of Anatomy...
3  <p>Various people in our lab will prepare a li...
4  <p>Are there any cases in which the splicing m...

   tags
0  ribosome binding-sites translation synthetic-b...
1                                    rna biochemistry
2                 immunology cell-biology hematology
3                                        cell-culture
4            splicing mrna spliceosome introns exons
```

The training set contains ~25,000 entries for diy, ~8,600 for biology, ~10,400 for cryptography, ~2,700 for robotics, ~12,000 for travel, ~15,000 for cooking.

The testing data has ~82,000 entries on physics but lacks the tags, which are to be predicted, submitted to kaggle for verification, and scored.

[5] kaggle.com/c/transfer-learning-on-stack-exchange-tags/data

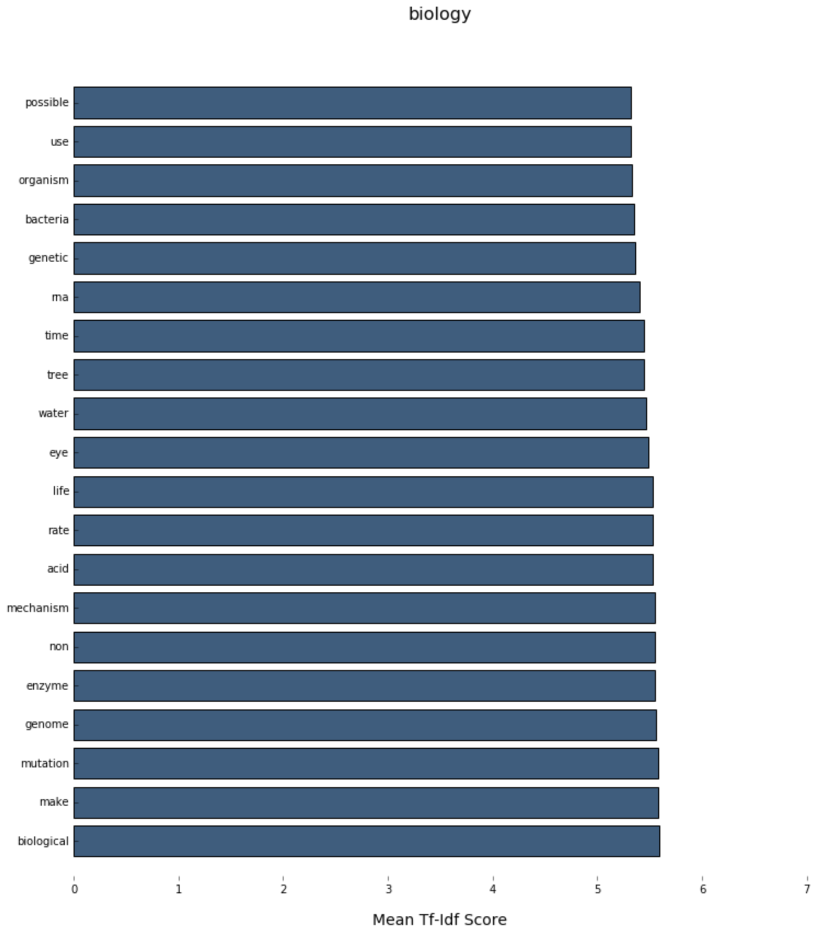
Exploratory Visualization

One measure of how important a word in a text collection is its term frequency (tf), how frequently a word occurs in a document. There are, however, words in a document that occur many times but may not be significant; in English, these are words like “the”, “is”, “of”, etc.

Another approach is to look at a term’s inverse document frequency (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. This can be combined with term frequency to calculate a term’s $tf\text{-}idf$, which is a way to score the importance of words in a document based on how frequently they appear across multiple documents, or in this case, rows/questions.

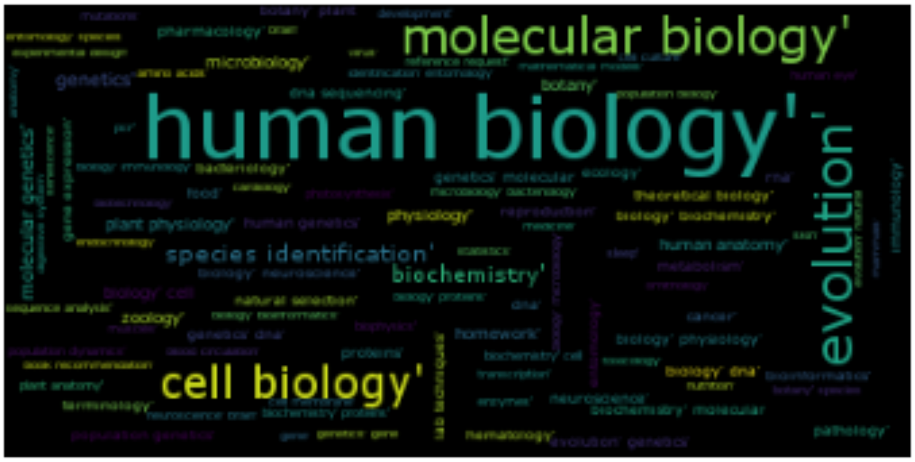
To get a sense of the questions, a histogram of the top 20 terms from the titles of the biology CSV (according to their $tf\text{-}idf$ values) provides some intuition as to what the classifier will be working with as input values, after a data cleaning step.

Biology Titles’ Top 20 $tf\text{-}idf$ Scores



Furthermore, the output values, or tags, are visualized below using python library Wordcloud[6], which ranks the tags to be larger if they are more popular, or frequent.

Biology Tags Wordcloud



Algorithms and Techniques

To solve the problem a Decision Tree classifier will be trained and used to see if it can give reasonable assignment of tags on the testing set.

As $tf\text{-}idf$ was used in the previous section to extract the dominating and significant features of the input data and visualize it via histograms, it will be utilized by the decision tree classifier as well. Transforming the input features into useful numerical values makes them easier for a classifier to consume.

Furthermore, cross validation grid search, `GridSearchCV`, is used to wrap the classifier and allows a wide range of tunable parameters to be optimized by trying different combinations of values.
For the `TfidfVectorizer`, used to construct the $tf\text{-}idf$ data, the minimum & maximum data frequency (`min_df` and `max_df`) parameters can be used to create a range of acceptable term frequencies and cut off trivial terms (those that appear too infrequently and cannot be used to generalize patterns or vice versa--terms that appear too frequently, across all rows, and thus add no value).
Term frequencies from both a question's title and its text will have to be combined, and more importance given to the titles, as they are a more forward and concise indicator of the question.

[6] https://github.com/amueller/word_cloud

The resultant term frequencies are analyzed by a `DecisionTreeClassifier` which is rooted in making choices that maximize the likelihood of predicting if a term frequency is correlated to a tag. The concept of decision trees similar to the game of '20 Questions', where each question tries to gain more information about the target answer. In this case, "good" questions can be defined mathematically through entropy to maximize the information gain at each question (or node/split in the tree).

To solidify what qualifies as "good" questions can be understood practically, without entropy: if playing the 20 questions game, one can start by asking a broad question to find out general classification (i.e. is it an object or living?) and narrowing in more as you get closer to the answer (if the answer to the previous question is living, we can ask--is it a human or other?).

Once these questions have been asked and answers/tags associated, the tree is built and one must simply follow it with a set of testing text (answering each question along the path as you go) to see if a tag is pertinent.

The criterion according to which the quality of a split is calculated varies between either 'gini' impurity or the traditional 'entropy' definition of information gain and both are examined as parameters using `GridSearchCV`.

DecisionTrees are vulnerable to over-fitting and thus the `max_depth` parameter is used to set the maximum depth of the classifier as a way of pruning it. If no parameter is given, the tree will continue to grow until all the leaf nodes are pure.

Lastly, to support the multi-label/multi-class nature of this problem, a `OneVsRestClassifier` is used to compare across DecisionTrees constructed per tag and create the assignments when predicting.

In order to gauge the general metrics of the classifier, the training sets are individually cross validated per topic to see how the classifier behaves in its “natural habitat” or within a topic.

Benchmark

The benchmark model for the fitted classifier will be running it on unseen physics questions and the evaluating the prediction results per question/row in the CSV submission as true positives (t_p) false positives (f_p), and false negatives (f_n). These categorizations allow for empirical analysis of the model and can be compared across implementations of classification algorithms using an averaged F_1 score.

The top ten F_1 scores of the competition leaderboard range between 0.2 - 0.29 (excluding an outlier in first place with a score of 0.67).
To be able to predict tags with an F_1 score above 0.2 would be satisfactory, above 0.25 would be great, and above 0.3 would be outstanding.

METHODOLOGY

Data Preprocessing

First, the data set is preprocessed by stripping the question text of its markdown properties and removal of HTML tags and code snippets.

Then, by utilizing classic NLP approaches such as removing stopwords, converting case to a convention (lower), removing punctuation/non-alphabet characters, and reducing permutations of the same word by using its linguistic stem, the feature data is transformed to allow for more meaningful results before being analyzed by `TfidfVectorizer`.

Tags are analyzed as strings, separated by spaces (preserving hyphenated words), and are split into an array. They are further encoded into bit-arrays where the length of the array is the number of unique combination of tags, and a 1 in a specified place/index represents the tags presence in a question.

Implementation

To get set up, two Python libraries are used predominantly throughout the investigation: `numpy`[7] for its linear algebra functionality and `pandas`[8] for CSV file I/O, in-memory DataFrame matrix representation, and data processing utilities.

CSV's are imported into pandas DataFrames:

```
    import pandas as pd
    biology = pd.read_csv("data/biology.csv")
```

and are combined and stored in a dictionary:

```
    df_hash = {
          "biology": biology,
          "travel": travel,
          "diy": diy,
          "cooking": cooking,
          "crypto": crypto,
          "robotics": robotics
    }
```

The implementation of data preprocessing is aggregated into the `clean_data` method seen below, which takes `raw_data` from the Dataframes as a parameter.

The method relies on two more linguistic Python libraries used commonly to accomplish the goals laid out in *Data Preprocessing* section: `bs4` (`BeautifulSoup4`)[9] used for parsing out code snippets and html elements and `nltk` (`Natural Language Toolkit`)[10] for lemmatizing/stemming words and only retaining those with meaning.

[7] numpy.org

[8] pandas.pydata.org/

[9] crummy.com/software/BeautifulSoup/bs4

[10] nltk.org

`clean_data(raw_data)` Implementation

```
from bs4 import BeautifulSoup
import re
from nltk.stem import WordNetLemmatizer

def clean_data(raw_data):
    if raw_data:
        # remove html tags & code snippets
        soup = BeautifulSoup(raw_data, "html.parser")

        [s.extract() for s in soup(['pre', 'code'])]

        question_text = soup.get_text()

        # remove everything but letters
        letters_only = re.sub("[^a-zA-Z]", " ", question_text)

        # normalize case
        words = letters_only.lower().split()

        # remove stopwords
        meaningful_words = [w for w in words if not w in stops]

        # remove permutations of the same word by reducing it to its stem
        wordnet_lemmatizer = WordNetLemmatizer()
        meaningful_word_stems = map(lambda x: wordnet_lemmatizer.lemmatize(x) ,
                                    meaningful_words)
        return( " ".join( meaningful_word_stems ))
    else:
        return ""
```

When running the experiment with the full data set, each topic in the `df_hash` is appended into one large dataframe, `df_all`, containing all 87,000 questions:

```
    frames = []
    for topic, df in df_hash.iteritems():
        frames.append(df)

    df_all = pd.concat(frames)
```

The remainder of investigation focuses on machine learning steps and utilizes the `scikit-learn`[11] Python library extensively.
For ease of use and chaining data transformations, a `pipeline` is created to funnel preprocessed data through a `TfidfVectorizer` transform for both the questions' titles and text (using the `Selector` class as a custom transformer to subset the DataFrames), and their resultant *tf-idf* values combined using a `FeatureUnion`.

These *tf-idf* values are used to construct decision trees via the `DecisionTreeClassifier` wrapped in a `OneVsRestClassifier` for each tag.

Pipeline transformation steps Implementation

```
steps = \
[('union', FeatureUnion(
    transformer_list=[
        # Pipeline for tfidf vectorization of the question's title
        ('title', Pipeline([
            ('selector', Selector(key='title')),
            ('tfidf', TfidfVectorizer(lowercase=True,
                                      stop_words="english") )
        ])),
        # Pipeline for tfidf vectorization of the question's content
        ('content', Pipeline([
            ('selector', Selector(key='content')),
            ('tfidf', TfidfVectorizer(lowercase=True,
                                      stop_words="english") )
        ]))
    ])),
("DT", OneVsRestClassifier(DecisionTreeClassifier( random_state = 42)))]
```

A `MultiLabelBinarizer` is used to accomplish encoding of tags into binary arrays.

```
     mlb = MultiLabelBinarizer()
     Y =  pd.DataFrame(mlb.fit_transform(df_all.tags) )
```

This constructed pipeline is then fit through `GridSearchCV` on the *tf-idf* values/features and their associated binary labels. A variety of parameters can be applied for each transform and the best combination is reported.

```
    pipeline = Pipeline(steps)
    clf = GridSearchCV(pipeline, parameters,
                       cv=3, scoring='f1_weighted')
    clf.fit( df_all, Y )
```

[11] scikit-learn.org

The `parameters` passed into `GridSearchCV` had the following values to try when tuning the `TfidfVectorizer`:

```
     tfidf__min_df : [ 0.001, 0.005, 0.01 , 0.05 ]
     tfidf__max_df : [ 0.9, 0.95 , 0.975 ]
```

While for the `DecisionTreeClassifier` parameters varied between:

```
     DT__criterion : ["gini", "entropy"]
     DT__max_depth : [5, 10, 25, 30]
```

to experiment with how shallow/pruned the DecisionTrees should be.

In total, the `parameters` hash that tunes almost every step/transform can be seen below:

```
parameters = {
    "union__title__tfidf__min_df" : [ 0.001, 0.005, 0.01 , 0.05],
    "union__title__tfidf__max_df" : [0.9, 0.95 , 0.975],
    "union__content__tfidf__min_df" : [ 0.001, 0.005, 0.01 , 0.05],
    "union__content__tfidf__max_df" : [0.9, 0.95 , 0.975],
    "DT__estimator__max_depth" : [5, 10, 25, 30],
    "DT__estimator__criterion" : ["gini", "entropy"],
    "union__transformer_weights" :  [{"title": 0.6, "content": 0.4},
                                     {"title": 0.75, "content": 0.25}]
}
```

‘`union_transformer_weights`’ is used to provide more emphasis to the $tf\text{-}idf$ values of a question’s title by scaling it more than its content when combining them with the `FeatureUnion`.

Refinement

Based on the results of `GridSearchCV`, the optimal parameters for the pipeline classifier are:

```
    DT__estimator__criterion: 'gini'
    DT__estimator__max_depth: 25
    union__content__tfidf__max_df: 0.9
    union__content__tfidf__min_df: 0.001
    union__title__tfidf__max_df: 0.9
    union__title__tfidf__min_df: 0.001
    union__transformer_weights: {'content': 0.4, 'title': 0.6}
```

The trees themselves are fairly shallow, at a `max_depth` of 25, so over-fitting the DecisionTrees was taken into account by pruning.

`Max` and `min` data-frequencies (`df`) for the vectorizer range between 0.001 - 0.9 for both the title and content of a question.

These parameters are set and used to make predictions on the test data.

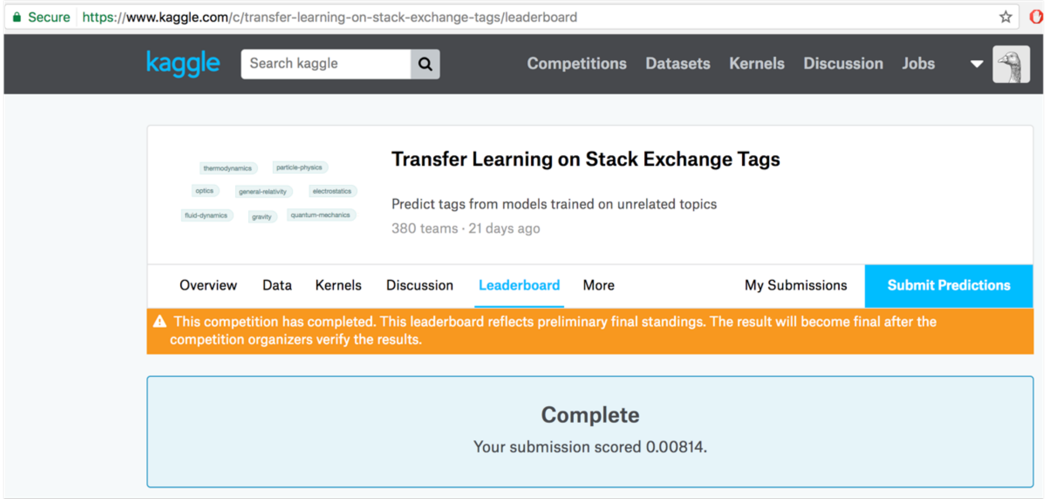
Results

Model Evaluation and Validation

The metrics discussed at the beginning of the investigation (the F_1 score) can be applied in two scenarios to judge the health and validate the designed classifier pipeline.

The first is the reported F_1 score from the fitted GridSearchCV instance which checks predictions against input data as it cross validates. When running this on one of the 6 CSV files at a time, the reported score is 0.351 or just over 35% of the guesses were correct, which is a non-trivial result.

The second, and big question of this investigation, is whether this knowledge can be transferred to physics, and unfortunately, the pipeline strongly underperforms on new features.
Since the nature of the challenge was a contest, the predictions created by the pipeline were submitted online on kaggle.com for a blind evaluation and received a measly score 0.814 %.



Transfer Learning on Stack Exchange Tags

Predict tags from models trained on unrelated topics

380 teams · 21 days ago

⚠ This competition has completed. This leaderboard reflects preliminary final standings. The result will become final after the competition organizers verify the results.

Complete

Your submission scored 0.00814.

The most glaring problem with the resultant physics predictions is that almost 50% of the 81,926 questions (40,569 to be exact) are simply blank and the classifier did not predict anything.

Justification

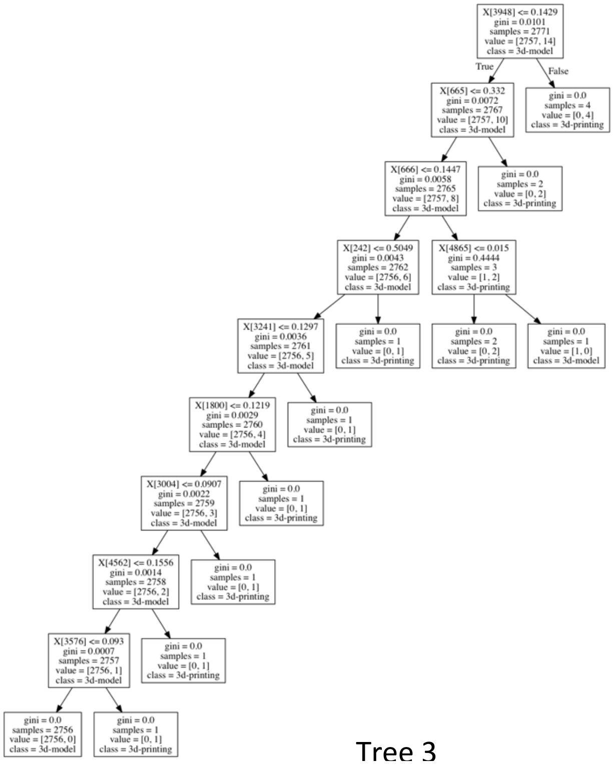
While the target problem is only marginally addressed, with a success rate under 1%, the classification system performed well on features within its respective topic set with scores in the mid 30%'s, which was roughly the mean of the scores on Kaggle's leaderboard.

CONCLUSION

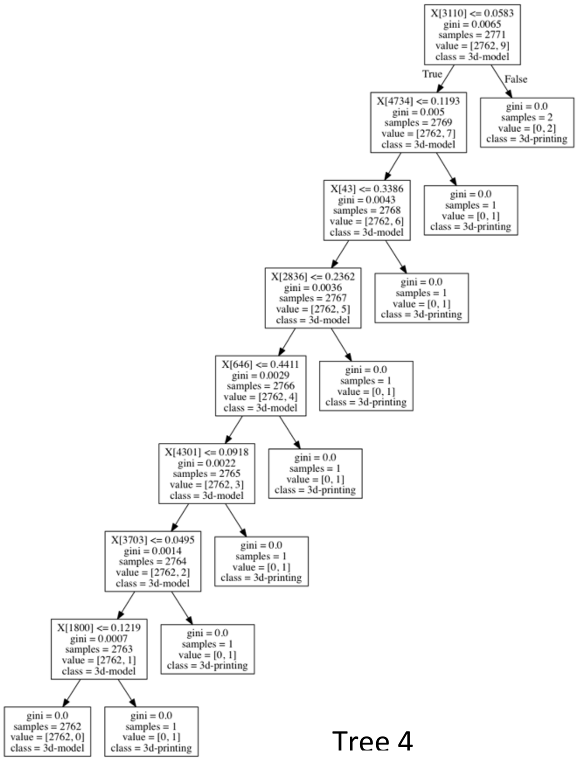
Free Form Visualization

Seen below is a sample of 4 various decision trees constructed for specific labels after fitting on the training data.

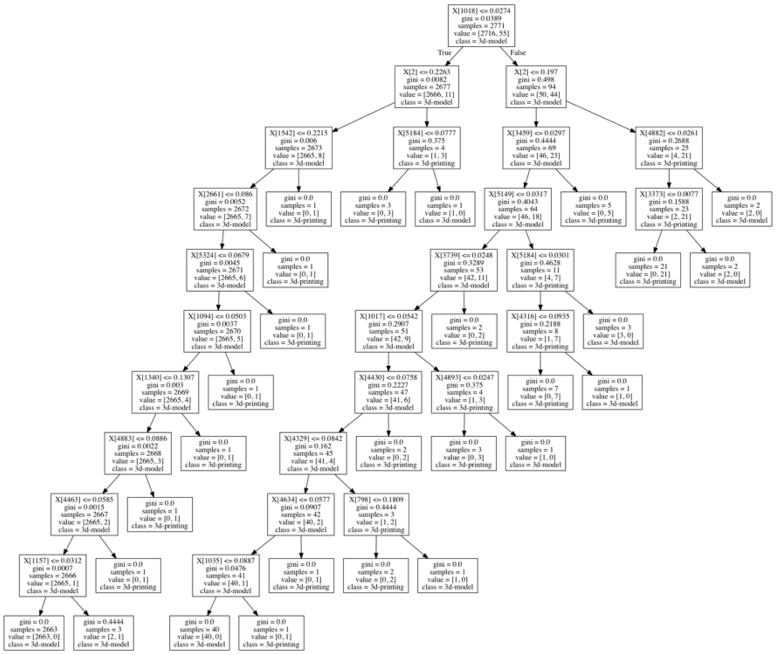
Tree 1



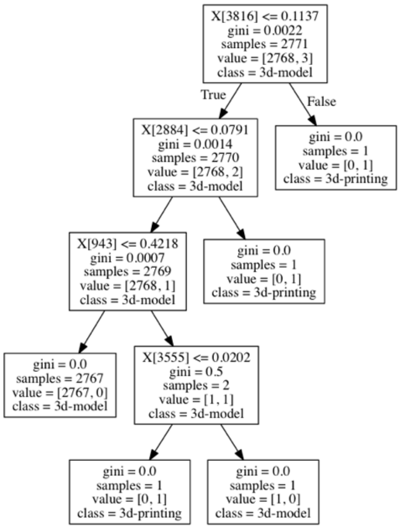
Tree 2



Tree 3



Tree 4



The numbers indexed by the `X[]` are the `TfidfVectorizer` encodings and their resultant values. Ideally, the information to decode these to their original strings can be provided to the `tree.export_graphviz` function as a parameter using the `.get_features_names` method. However, since there was a custom transformer introduced (`Selector` class) to subset the input data by title and text and combine it using a `FeatureUnion`, the pipeline was unable to provide the list of `feature_names` according to the encodings. Attempting to patch the class through decorators and with other libraries (singledispatch and eli5) provided little help.

The gini values are a statistical measure of dispersion and are related through entropy--along these inequalities the tree formulates its decisions. The given examples analyze the distinctions between the tags/class `3d-model` and `3d-printing` according to the tf-idf values.

Reflection

While it is certainly anticlimactic to receive such low results by the classifier when out in the wild, it is important to keep in mind that this challenge had an element of facetiousness inherent and it might be presumptuous to think that there is a unifying thread to a sample of knowledge. Pragmatically, the issue all along is that as a text classifier, both the features and the tags share a different vocabulary and is very difficult to correlate.

Improvement

With ample time, this investigation could be expanded by experimenting with the effects of scaling the classifier and seeing if there is a tipping point at which it loses accuracy. Overfitting is a usual suspect when using decision trees, however, this problem was mitigated by trying different combinations of `max_depth` with input data sets being just individual topics and finding the best value. Scale comes into play here as well and the question becomes if fitting the entire dataset at once plays well with the optimal depth chosen at fitting per topic.

Other classifiers are always fair contestants when trying to tackle such scale issues with larger datasets: k-nearest-neighbor (kNN) and Support Vector Machine (SVM) would be good candidates to investigate.