DatasetFeaturesClassification

March 27, 2024

1 Custom Classification Task: Dataset, Features, Experiments

In this assignment, you will:

1. Propose a custom classification task and create a corpus of documents that are annotated for this task.
2. Propose discriminative features to be included in the feature vector representation for the examples in this task and evaluate their utility by training and testing Logistic Regression models.

1.1 Write Your Name Here:

2 Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX and download a PDF version *.pdf* showing the code and the output of all cells, and save it in the same folder that contains the notebook file *.ipynb*.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing we will see when grading!
7. Submit **both** your PDF and notebook on Canvas.
8. Verify your Canvas submission contains the correct files by downloading them after posting them on Canvas.

2.1 Corpus acquisition and formatting (30p)

1. Create a corpus of documents that you can use to create at least 300 examples for a task that is of interest you, a task that can be modeled as classification. In class we discussed the following NLP tasks:
    - Document classification.
    - Named entity recognition.
    - Relation extraction.

However you can choose any text processing task as long as it can be solved using classification.

- If the labels are not readily available, manually annotate your data, such that you have at least 300 classification examples, with at least 50 examples for training in each class.
- For manual annotation, you can use one of the schemes or annotation tools discussed in class.
- The more data in your collection, the better your classification models will tend to perform on it.

2. Partition your data into three datasets: *train*, *dev*, and *test*, with the training set containing 80% of the documents, development 10%, and test 10%.

3. Your choice of task, documents, and labels is completely up to you. For example, if you choose to work on document classification, some possible sources of data are:

   - **Project Gutenberg**: Metadata is available at this Github repo along with URLs for the texts. Labels here can be author, subject, genre, etc.
   - **News articles**: Crawl news articles from different domains (e.g,. CNN, FoxNews); the label for each article is the domain.
   - **Movie summaries**: Labels here can be any categorical metadata aspect (genre, release date); note real-valued metadata (like box office, runtime) can be discretized by selecting some reasonable thresholds.
   - **Tweets**: Download your own tweets. Labels here can be any categorical metadata included in the tweet, or labels you add by hand (e.g., sarcasm).
   - **Book reviews**: Data and metadata is available from the GoodReads dataset and the Amazon reviews dataset.
   - **Restaurant reviews**: Data and metada is available from the Yelp dataset.
   - For more ideas about NLP tasks:
     - SuperGLUE is a benchmark dataset for 8 difficult NLP tasks.
     - BIG-bench is collaborative benchmark intended to probe large language models and extrapolate their future capabilities, containing more than 200 tasks.

4. Additional requirements:

   - No sentiment classification.
   - It is required that you **create your own dataset**. While the task may not be new (e.g. topic classification), the dataset itself must be new.
     - For graduate students this can also serve as the basis for a project.
     - Using an existing dataset as if it were your own will be in violation of academic integrity.

2.1.1 Task and Dataset description

Describe your data. What is the source of the documents, and what do the labels mean? How representative is the data, e.g. in terms of demographics, languages, social categories, … What are its limitations / biases?

// YOUR CONTRIBUTION HERE

2.1.2 Dataset reading and statistics

Read the documents in each of the 3 datasets (*training*, *development*, and *test*) and for each dataset display the following statistics:

1. The total number of examples in the dataset.

2. The distribution of labels, which is task dependent. For example:
    - For document classification, this would be the number of documents for each label.
    - For NE recognition, this would be the number of names found for each NE type.
    - For RE, this would be the total number of NE pairs in a sentence that are in the relationship (positive) or not (negative).

For document classification, you can reuse the dataset reading code from the sentiment analysis assignments. For a different classification task, you would have to write different functions to read the data and the annotations. For example, if you use the Brat annotation tool, the annotations are stored in the `.ann` text files, which are straightforward to read.

[4]:
```
datapath = "../data"

# YOUR CODE HERE
```

2.2 From textual examples to feature vectors (30p)

Read the documents in each dataset (train, dev, test) and generate the corresponding examples as feature vectors. Some skeleton code is provided below, but feel free to customize as you see fit for your task.

1. Tokenize each document using the spaCy tokenizer or the BPE tiktoken tokeninzer.
2. Create at least two non-trivial feature functions, and include them in the *features* list.
    - A passing grade (only 10p) will be given to generic features that apply across arbitrary text classification problems (e.g., a feature for bigrams);
    - A better grade (all 30p) will be given for features that reveal your own understanding of your data. What features do you think will help for your particular problem? Would features based on higher level NLP processing tasks (syntactic parsing, coreference resolution) be useful?
    - You are free to read in any other external resources you like (dictionaries, document metadata, etc.), but make sure you include them in the `../data` folder.
3. Process each dataset into a set of examples, by mapping each document in the dataset to its corresponding set of examples. Process each example into a feature vector, using the feature functions from *features*. You can reuse feature vector representation code from previous assignments. Features need to be relevant for the task.
    - map example to a dictionary of feature names.
    - map feature names to unique feature IDs.
    - each example is a feature vector, where each feature ID is mapped to a feature value (e.g. word occurences).

[18]:
```
import spacy
from spacy.lang.en import English
from scipy import sparse
from sklearn.linear_model import LogisticRegression

# Create spaCy tokenizer.
```

```
spacy_nlp = English()

def spacy_tokenizer(text):
    tokens = spacy_nlp.tokenizer(text)

    return [token.text for token in tokens]

# YOUR CODE HERE
features = []









def create_examples(path, dataset, features):
    instances = []
    labels = []
    # YOUR CODE HERE




    return instances, labels
```

2.3 Train and evaluate (15p + 15p)

Write a `train_and_test` function that takes as input the training and test examples, trains a Logistic Regression model on the training examples and evaluates it on the test examples. You can use the default value for the `C` hyper-parameter, or tune it on the development examples. Report accuracy, or precision and recall, depending on the task.

[15]:
```
def train_and_test(trainX, trainY, testX, testY):
    # YOUR CODE HERE
```

```
    return
```

2.3.1 Ablation experiments

Evaluate the impact of your features by training and testing with vs. without each feature.

[20]:
```
# Specify features to use. Do this multiple times, with and without any of the␣
↪new features.
features = []

# Create training, development, and test examples
trainX, trainY = create_examples(datapath, 'train', features)
devX, devY = create_examples(datapath, 'dev', features)
testX, testY = create_examples(datapath, 'test', features)

# Evaluate LR model.
train_and_test(trainX, trainY, testX, testY)
```

2.4 [5111] Plot ROC or PR Curve (30p)

Mandatory for graduate students, optional for undergraduate students.

Take your best classifier and plot a Receiver Operating Characteristic (ROC) curve if you use accuracy for evaluation, by varying a threshold on the probabilistic output. You can use the sklearn implementation, or implement your own. If you use precision and recall, plot a precision vs. recall (PR) curve instead.

[21]:
```
# YOUR CODE HERE
```

2.5 Analysis (30p)

Very important: Include an analysis of the data and the results that you obtained in the experiments above. It is important that results are formatted well, e.g. using tables, lists, etc.

2.6 Bonus points (∞p)

Anything extra goes here, e.g. evaluate other ML models, evaluate an LLM such as GPT using the Chat completion API, …

[ ]:
```

```