GENERAL ASSEMBLY

NATURAL LANGUAGE PROCESSING AND TEXT CLASSIFICATION

Sri Kanajan

LEARNING OBJECTIVES

- Demonstrate how to classify text or documents using scikit-learn
- Understand how to transform text into a numerical representation
- Understand what is Count Vectorizer and TFIDF Vectorizer
- Discussion around NLP packages

COURSE

PRE-WORK

PRE-WORK REVIEW

- Experience with scikit-learn classifiers, specifically random forests and decision trees

OPENING

NATURAL LANGUAGE PROCESSING AND TEXT CLASSIFICATION

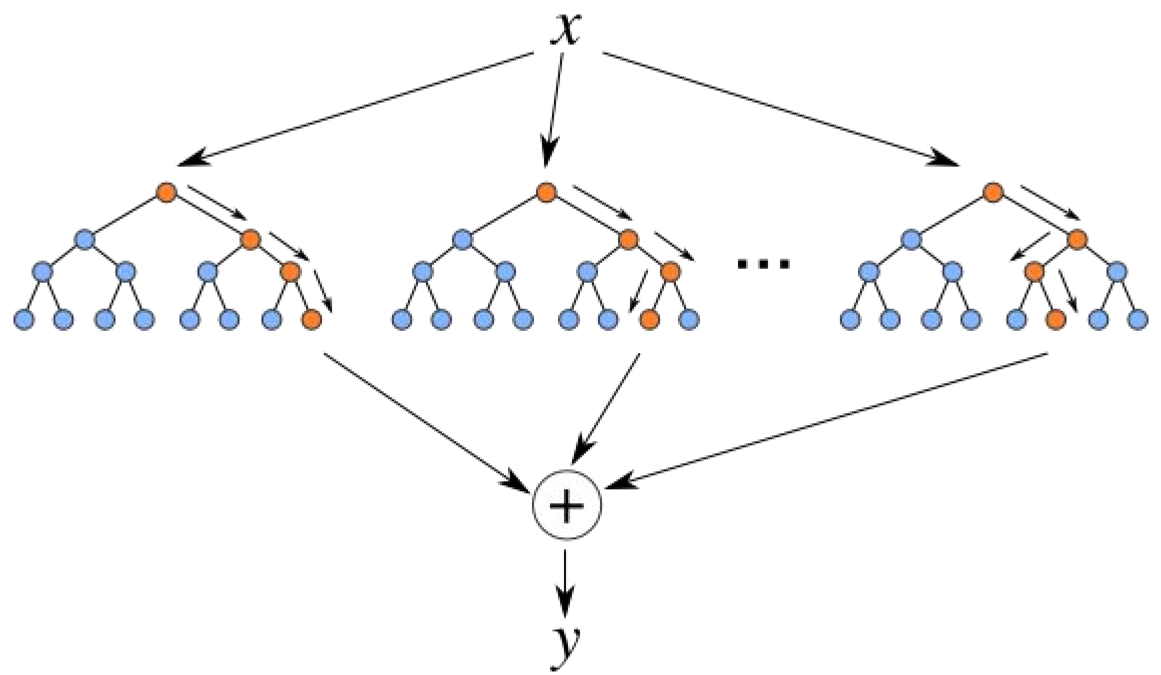
REVIEW: DECISION TREES AND RANDOM FORESTS

- What are decision trees?
- What are random forests?

REVIEW: DECISION TREES AND RANDOM FORESTS

- Decision trees are models that ask a series of questions. The next question depends upon the answer to the previous question.

- Random forest models are ensembles of decision trees that are randomized in the way they are created.



REVIEW: DECISION TREES AND RANDOM FORESTS

- Decision trees are weak learners that are easy to overfit.
- Random forests are strong models that are made up of a collection of decision trees.
  - They are non-linear (as opposed to logistic regression).
  - They are mostly black-boxes (no coefficients, although we do have a measure of feature importance).
  - They can be used for classification or regression.

OPENING

NATURAL LANGUAGE PROCESSING AND TEXT CLASSIFICATION

WHAT IS NATURAL LANGUAGE PROCESSING (NLP)?

- Natural language processing is the task of extracting meaning and information from text documents.

- These tasks may range from simple classification tasks, such as deciding what category a piece of text falls into, to more complex tasks like translating or summarizing text.

- For most tasks, a fair amount of pre-processing is required to make the text digestible for our algorithms. We typically need to *add structure* to our *unstructured data*.

- Text classification is one of the most common problems in NLP. I.e. you want to perform some sort of classification but using text as the features.

TEXT CLASSIFICATION

- An example, we may want to identify whether an article is a sports or business story. Or whether an article has positive or negative sentiment.
- Typically, this is done by using the text as features and the label as the target output. This is referred to as *bag-of-words* classification.
- To include text as features, we usually create a *binary* feature for each word, i.e. does this piece of text contain that word?

TEXT CLASSIFICATION

- To create binary text features, we first create a vocabulary to account for all possible words in our universe.
- As we do this, we need to consider several things.
    - Does order of words matter?
    - Does punctuation matter?
    - Does upper or lower case matter?

TEXT CLASSIFICATION

- This table illustrates features created from the following passage.

"It's a great advantage not to drink among hard drinking people."

Feature	Value
it's	1
great	1
good	0
advantage	1
not	1
think	0
drink	1
from	0
hard	1
drinking	1

Feature	Value
people	1
withhold	0
random	0
smoke	0
among	1
whenever	0
thoughtful	0
inexhaustible	0
men	0
Nick	0

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

Given the experience from the Evergreen prediction problem, discuss your answers to the following questions and explain your reasoning.

1. Does word order matter?
2. Does word case (e.g. upper or lower) matter?
3. Does punctuation matter?
4. What is "bag-of-words" classification?

DELIVERABLE

Answers to the above questions

DEMO

TEXT PROCESSING IN SCIKIT-LEARN

TEXT PROCESSING IN SCIKIT-LEARN

- Scikit-learn has many pre-processing utilities that simplify tasks required to convert text into features for a model.

- These can be found in the `sklearn.preprocessing.text` package.

- We will use the StumbleUpon dataset again to perform text classification. This time, we will use the text content itself to predict whether a page is ‘evergreen’ or not.

- Open the starter code notebook to follow along.

COUNTVECTORIZER

- `CountVectorizer` converts a collection of text into a matrix of features. Each row will be a sample (an article or piece of text) and each column will be a text feature (usually a count or binary feature per word).

- `CountVectorizer` takes a column of text and creates a new dataset. It generates a feature for every word in all of the pieces of text.

- **REMEMBER**: Using all of the words can be useful, but we may need to use *regularization* to avoid overfitting. Otherwise, rare words may cause the model to overfit and not generalize.

COUNTVECTORIZER

- Instantiate a new `CountVectorizer`.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max_features = 1000,
                             ngram_range=(1, 2),
                             stop_words='english',
                             binary=True)
```

COUNTVECTORIZER PARAMETERS

- There are several parameters to utilize.

- `ngram_range` - a range of word phrases to use
  - `(1,1)` means use all single words
  - `(1,2)` means use all contiguous pairs of word
  - `(1,3)` means use all triples

- `stop_words=’english’`
  - Stop words are non-content words (e.g. ‘to’, ‘the’, ‘it’, etc). They aren’t helpful for prediction, so they get removed.

COUNTVECTORIZER PARAMETERS

- `max_features=1000`
  - Maximum number of words to consider (uses the first `N` most frequent)

- `binary=True`
  - To use a dummy column as the entry (1 or 0, as opposed to the count). This is useful if you think a word appearing 10 times is no more important than whether the word appears at all.

COUNTVECTORIZER

- Vectorizers are like other models in scikit-learn.
    - We create a vectorizer object with the parameters of our feature space.
    - We `fit` a vectorizer to learn the vocabulary.
    - We `transform` a set of text into that feature space.

COUNTVECTORIZER

- Note: there is a distinction between *`fit`* and *`transform`*.
  - We `fit` from our training set. This is part of the model building process, so we don't look at our test set.
  - We `transform` our test set using our model fit on the training set.

COUNTVECTORIZER EXAMPLE

```
titles = data['title'].fillna('')

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max_features = 1000,
                             ngram_range=(1, 2),
                             stop_words='english',
                             binary=True)

# Use `fit` to learn the vocabulary of the titles
vectorizer.fit(titles)

# Use `tranform` to generate the sample X word matrix - one column per
feature (word or n-grams)
X = vectorizer.transform(titles)
```

RANDOM FOREST PREDICTION MODEL

- We can now build a random forest model to predict “evergreenness”.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators = 20)

# Use `fit` to learn the vocabulary of the titles vectorizer.fit(titles)

# Use `tranform` to generate the sample X word matrix - one column per feature (word or n-grams)
X = vectorizer.transform(titles)
y = data['label']

from sklearn.cross_validation import cross_val_score

scores = cross_val_score(model, X, y, scoring='roc_auc')
print('CV AUC {}, Average AUC {}'.format(scores, scores.mean()))
```

TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY

- An alternative *bag-of-words* approach to `CountVectorizer` is a Term Frequency - Inverse Document Frequency (TF-IDF) representation.

- TF-IDF uses the product of two intermediate values, the *Term Frequency* and *Inverse Document Frequency*.

TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY

- *Term Frequency* is equivalent to `CountVectorizer` features, just the number of times a word appears in the document (i.e. count).

- *Document Frequency* is the percentage of documents that a particular word appears in.

- For example, “the” would be 100% while “Syria” is much lower.

- *Inverse Document Frequency* is just 1/Document Frequency.

TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY

- Combining, TF-IDF = Term Frequency * Inverse Document Frequency or TF-IDF $w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$
- The intuition is that the words that have high weight are those that either appear ***frequently*** in this document or appear ***rarely*** in other documents (and are therefore unique to this document).

- This is a good alternative to using a static set of "stop" words.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
```

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. What does TF-IDF stand for?
2. What does this function do and why is it useful?
3. Use `TfidfVectorizer` to create a feature representation of the StumbleUpon titles.

DELIVERABLE

Answers to the above questions and feature representation

INDEPENDENT PRACTICE

TEXT CLASSIFICATION IN SCIKIT-LEARN

ACTIVITY: TEXT CLASSIFICATION IN SCIKIT-LEARN

EXERCISE

DIRECTIONS (30 minutes)

1. Use the text features of `title` with one or more feature sets from the previous random forest model. Train this model to see if it improves AUC.
2. Use the `body` text instead of the `title`. Does this give an improvement?
3. Use `TfIdfVectorizer` instead of `CountVectorizer`. Does this give an improvement?

Check: Were you able to prepare a model that uses both quantitative features and text features? Does this model improve the AUC?

DELIVERABLE

Three new models

NLP PACKAGES

- NLTK
    - nltk is the most popular, but it hasn't kept up with advances and isn't very well maintained.
    - Has many pre-packaged and pre-trained english functions. E.g. stemming/lemmatization

- Spacy
    - More modern but not for commercial use

- TextBlob
    - Easy to use and has sentiment analysis pre-packaged

- Word2Vec
    - *Word2Vec* is another unsupervised model for latent variable NLP.
    - Has word similarity functions

CONCLUSION

TOPIC REVIEW

LET'S REVIEW

- Natural language processing (NLP) is the task of pulling meaning and information from text.
- In scikit-learn, we use vectorizers to create text features for classification, such as `CountVectorizer` and `TfIdfVectorizer`.
- You can improve your text features by cleaning your text data. E.g. taking the lower case words, spell corrections, punctuations, order of words…

LESSON

EXIT TICKET

DON’T FORGET TO FILL OUT YOUR EXIT TICKET