# Feature Creation and Selection

INFO-4604, Applied Machine Learning
University of Colorado Boulder

**October 30, 2020**

Prof. Abe Handler

# Remember HW #4

# Features

Often the input variables (*features*) in raw data are not ideal for learning

Recently, we have discussed how there is usually an additional step of defining and creating the features you will use in your predictor.

# Feature Engineering

Feature **engineering** refers to the process of designing the types of features that will be used in a classifier or predictor

- In text, should you use the sequence of words? Counts of words? Counts of phrases?
- In images, should you use the raw pixels? Counts of colors? Other characteristics?

There are some standard types of features that are commonly used for various types of data, but sometimes it is appropriate to design your own

- Requires thought – be creative!

# Feature Extraction

Feature **extraction** refers to the actual step of converting raw data into vector of feature values*

Consider *efficiency* of extracting features:

- When making a prediction on a new instance, you will need to extract the features from the instance
- If you have complex features, it may be slow to make predictions

* The textbook also uses “feature extraction” to refer to certain types of transformations of features

# Feature Selection

Feature **selection** refers to choosing a subset of specific features out of all the features you have engineered and extracted

- Can reduce the complexity of the classifier (and therefore reduce risk of overfitting)
- Can help with runtime/memory complexity with a large number of features

Feature selection is a form of *dimensionality reduction* (more on that later)

# Feature Engineering

Certain types of data (text, images, video, audio, networks) will require specialized features specific to the data type

We'll look at a few examples

- Ch. 8 of the book gives a good overview of features commonly used for text classification

# Features: Text

The most common feature encoding for text is called a **bag of words** representation

- Describes which words are in an instance but not the order or position of the words
- Useful because it can generalize better than specific word positions
  - If *"river" is the 4th word in the document* is treated as a distinct feature from *"river is the 12th word in the document*, it will be hard to get enough training examples to cover all the possible word positions, and it will be hard to learn

# Features: Text

In a bag of words representation, the set of features is the set of unique words that appear in the dataset (called the **vocabulary**)

The values of the features for each instance (each document) are typically the number of times the word appears in the document

- e.g., a value of 16 for the feature “the” means “the” appeared in the document 16 times
- A value of 0 means the word is not in the document
  - Most values will be 0: *sparse* features
- Sometimes binary values are used (1 if present; 0 if not)

# Features: Text

**Tokenization** refers to splitting a text string into a sequence of words (word **tokens**)

Simple starting point: split string by whitespace
(but this won't work for languages that don't use spaces, like Chinese)

Usually there is additional preprocessing you'll want to do to text

# Features: Text

With bag of words features, it is common to make all words *lowercase* so that different capitalizations are treated as the same feature

Though sometimes capitalization is informative…

"they went to the White House" vs
"they lived in a white house"

# Features: Text

Also common to remove punctuation

- Don't want to treat "blue," differently from "blue"

Though sometimes punctuation is useful…

- Ending punctuation can indicate type of statement (? or !)
- Emoticons and emoji useful for sentiment
- Apostrophes informative ("its" different from "it's")
- Sometimes better to treat punctuation as its own token e.g., "blue ," instead of "blue,"

# Features: Text

**Stemming** is a technique to convert a word to its "root" or "base" form

- This forces different forms of a word to be treated as the same feature

| Word | Stem |
|---|---|
| fish | fish |
| fishes | fish |
| fished | fish |
| fishing | fish |

# Features: Text

**Stemming** is a technique to convert a word to its "root" or "base" form

Useful when you have a small number of examples of each word, but has limitations as well:

- Sometimes the tense/aspect of a word is important for the prediction task, and stemming loses that info
- Common stemming algorithms can make mistakes
  - university, universe, universal

# Features: Text

Raw word counts can overemphasize uninteresting words

- Common words like “the” and “and” will have a high count in most documents

Common technique: reweight the counts using **TF-IDF** weighting

- TF = term frequency
- IDF = inverse document frequency

# Features: Text

$$\text{tf-idf}(\text{t,d}) = tf(t,d) \times \text{idf}(\text{t,d})$$

$$\text{idf}(\text{t,d}) = log \frac{n_d}{1+\text{df}(\text{d,t})}$$

where $n_d$ is the total number of documents, and df(d, t) is the number of documents that contain the term t.

Basically: adjust the word count in a document by how common the word is in the entire dataset

# Features: Text

Generalizations of bag of words features are **n-grams**

- Sequences of words of length *n*
- For example, 2-gram features:
  count of every 2-word phrase in a text

1-grams are the same as “bag of words”

# Features: Text

"the quick brown fox jumped over the lazy dog"

1-grams (unigrams):

```
the
quick
brown
fox
over
lazy
dog
jumped
```

# Features: Text

"the quick brown fox jumped over the lazy dog"

2-grams (bigrams):

```
the_quick
quick_brown
brown_fox
fox_jumped
jumped_over
over_the
the_lazy
lazy_dog
```

# Features: Text

"the quick brown fox jumped over the lazy dog"

3-grams (trigrams):

```
the_quick_brown     jumped_over_the
quick_brown_fox     over_the_lazy
brown_fox_jumped    the_lazy_dog
fox_jumped_over
```

# Features: Text

What size n-grams?

- Too short: might miss important detail
- Example: sentiment classification
  - "great" might indicate positive sentiment…
    …unless it was part of the phrase "not great"
  - 2-grams would capture this; 1-grams would not

# Features: Text

What size n-grams?

- Too long: might not see enough examples
- Example: sentiment classification
  - “the sausage was great”
  - “the cheese was great”
  - “the crust was great”

  Maybe you only see each of these phrases once
    - With 4-grams, hard to learn from these
    - With 2-grams, can learn that “was great” indicates positive sentiment

# Features: Text

What size n-grams?

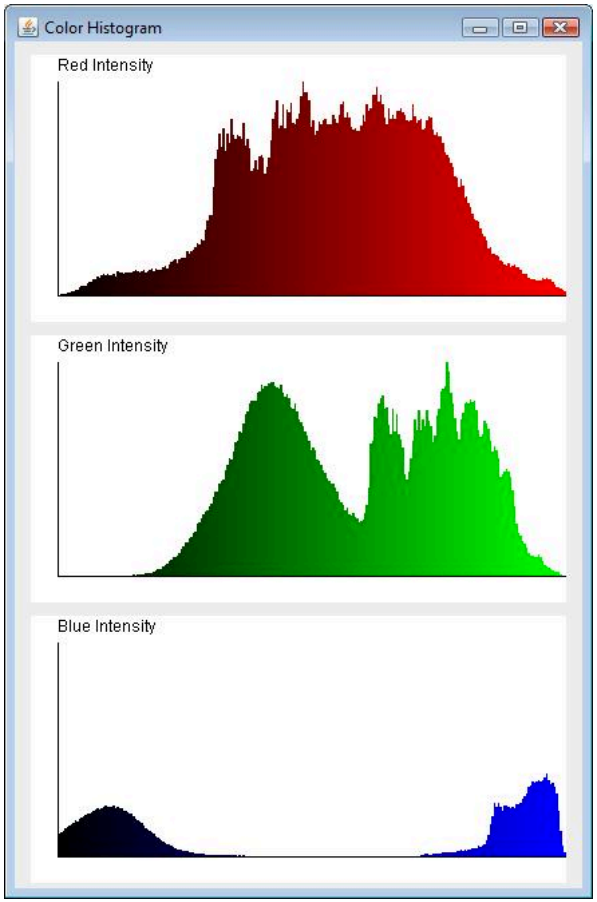
Good to include different sizes

- e.g., both 1-grams and 2-grams in your feature set
- 1 and 2 most common values of *n*
- 4 and higher usually less helpful unless you have a lot of data

# Features: Images

*Color histograms* for images are analogous to "bag of words" counts in text

- Number of times a color value appears, regardless of position in image
- Unlike words, color values are continuous (or can be treated that way), so histogram might group into "bins"




From:
http://marvinproject.sourceforge.net/en/plugins/colorHistogram.html

# Features: Images

In text, n-grams give more information than individual words, but are still *local*

- i.e., not the specific position in a document, but relative ordering within the n-gram

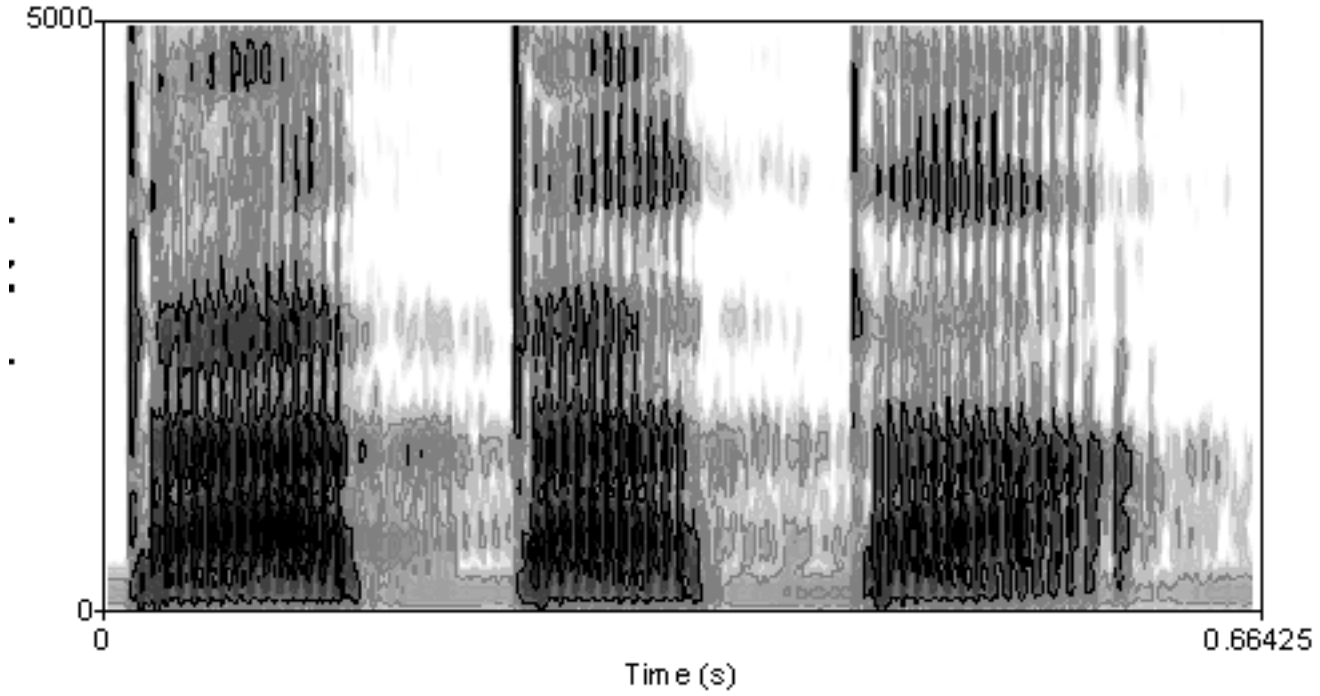
Local context can also be characterized in images

- e.g., if neighboring pixels have very different color values, this might indicate an *edge*

Lots of features have been designed for images, but are beyond the scope of this class.

# Features: Audio

Audio data is somewhat similar to image data in that there are different intensities (amplitudes) at different positions (signal frequency and time)



# Feature Engineering

Think about other characteristics of your data that might be informative…

- The length of a document?
- The aspect ratio of image?

These examples may or may not be useful for a particular task – but are features you could create that are separate from the content itself

- If you were to classify your data, what would be helpful to know?
- Be creative – don't need to rely on standard features

# Feature Engineering

Features can be created from other features

Examples:

- $x_3 = x_1 * x_2$
- $x_3 = x_1$ XOR $x_2$
- $x_1 = |x_1|$

You might automatically add these features for all existing features if you think they might help, or manually add these types of rules for specific features that you think are important.

# Feature Engineering

Features can be created from other classifiers

Examples:

- Multiple labels (predict one first, use it to predict the other)
- For text, *natural language processing* tools can predict characteristics of the language (e.g., grammatical relations between words)
- For images/video, *computer vision* tools can detect characteristics (e.g., faces or objects)

# Feature Selection

Once you've generated all features you want, you may want to select only a subset of them

Why?

- Too many features may result in overfitting
  - Though regularization can address this
- Too many features will make training and prediction slower
  - Efficiency is a big reason to do feature selection

# Feature Selection

Recall: L1 regularization tends to produce sparse solutions (weights of exactly 0)

- Can be used as a method of feature selection!
- Different from other selection methods in that the "selection" happens during training, not before

Often L1 regularization performs worse than L2, but the sparsity from L1 can help a lot with efficiency at prediction time

- Features with weight 0 can be ignored

# Feature Selection

**Sequential** algorithms pick features one-by-one

Sequential backward selection:

- Start with all features
- Train a classifier where you remove one feature
- Pick the feature whose removal least affected the classifier performance
- Remove that feature and repeat
    - Stop once removing features hurts the performance too much

Simple to implement, and directly optimizes for classification performance, but slow.

# Feature Selection

**Statistical tests** can be used to select features

General idea:
measure the statistical dependence or correlation between each feature and the labels

A common test is the **chi-squared** test.

# Feature Selection

**Statistical tests** can be used to select features

How to choose features?

- A test statistic is calculated for each feature
- To select features, choose:
  - the top N features (ranked by their test statistic); or
  - all features whose test statistic is below a threshold

# Feature Selection

**Statistical tests** can be used to select features

Unlike L1 regularization or sequential methods, feature selection through statistical tests does not explicitly try to help classifier accuracy

- But they can be much more efficient
- Popular for this reason

# Feature Selection

Certain heuristics can be used to remove features.

In text, **stop words** are common words (like “the”) that appear often but are not expected to be useful

- There are various lists of stop words out there
- You can also define stop words based on high frequency (e.g., words that appear in >90% of documents)

Removing stop words doesn’t dramatically reduce the number of features (relatively few words are stop words), but it will reduce the number of active features in each document.

# Feature Selection

Certain heuristics can be used to remove features.

In text, there is a potentially a very large number of features (words and n-grams), but most of the features will occur infrequently (long tail)

Removing infrequent features (e.g., an n-gram that only appears in 1 document) can substantially reduce the number of features without affecting performance.