COMP 551 – Applied Machine Learning
Lecture 12: Stacking and features

Instructor: Herke van Hoof (*herke.vanhoof@mcgill.ca)*

Class web page: *www.cs.mcgill.ca/~hvanho2/comp551*

Project 1 feedback

- Unexpected grading issues. Sorry! Grading should be done Thursday!
- Some common errors:
- Make live easy for the TA's!: answers not in report, excessively long reports, report not stapled together, missing name or ID…
- Make sure we can understand your plot: legend, lines look different in black&white, axes are labeled, …

Project 4 teams

- Project 4 and 5 will be bigger projects
- Longer time, but also more work
- Should be done in **teams of 3**

  (Working alone is discouraged, as you'll have to do the work of 3 people by yourself…)
- Teams must be **completely different** between project 4 and 5
- Teams must consist of only people in section 002!
- Project 4 starts end of next week. You can already start looking for teams. There'll be a myCourses forum to find partners.

Quizzes

- Practice questions available for:
  - SVM (2nd part)
  - Decision trees
  - Ensemble methods

Main types of machine learning problems

- Supervised learning
  - Classification
  - Regression
- Unsupervised learning
- Reinforcement learning

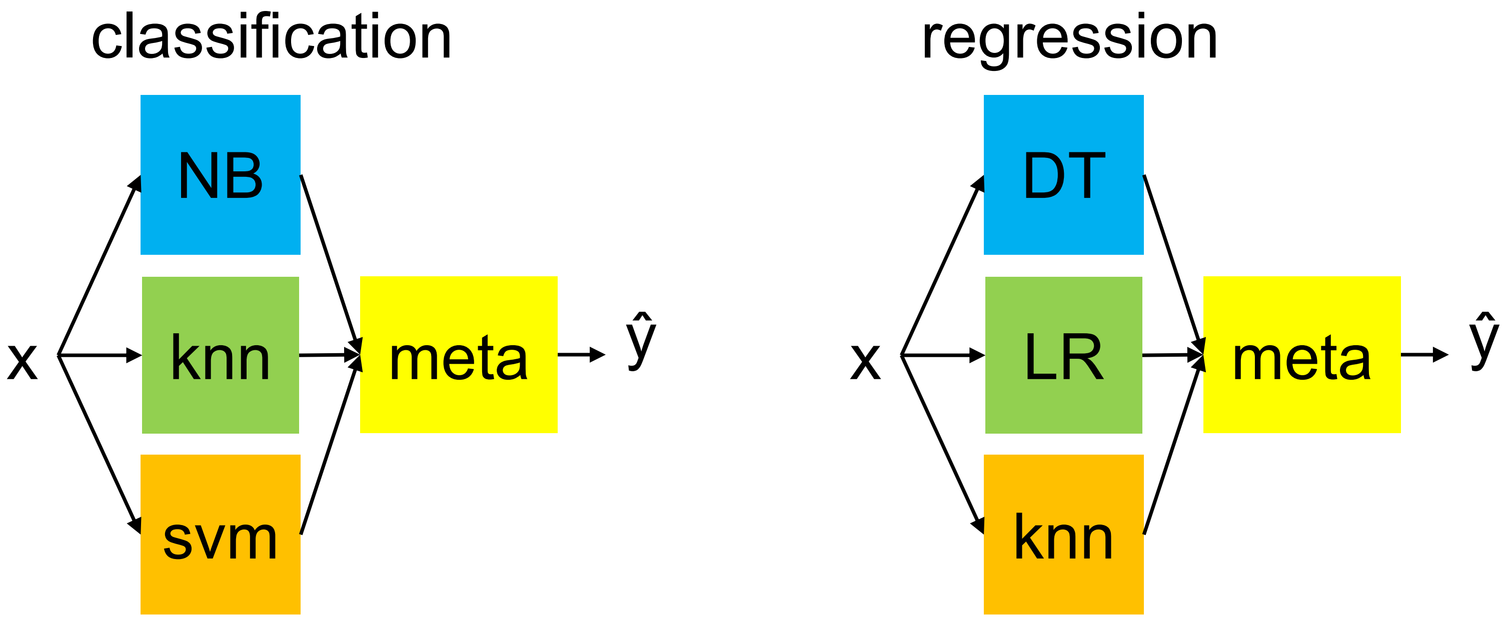
Ensemble methods

Recap ensemble methods so far

- Combine multiple classifiers (or regressors)
- Hopefully, each classifier makes different mistakes
- In that case, combining them might help
- 2 ways so far to obtain different classifiers from the same *family*
  - Independent training using randomized dataset and/or training
    - Bagging, random forests, extremely randomized trees
  - Add classifiers that focus on examples ensemble gets wrong
    - Boosting
  - Both typically use a large amount of classifiers
- Can we combine some classifiers of different types?

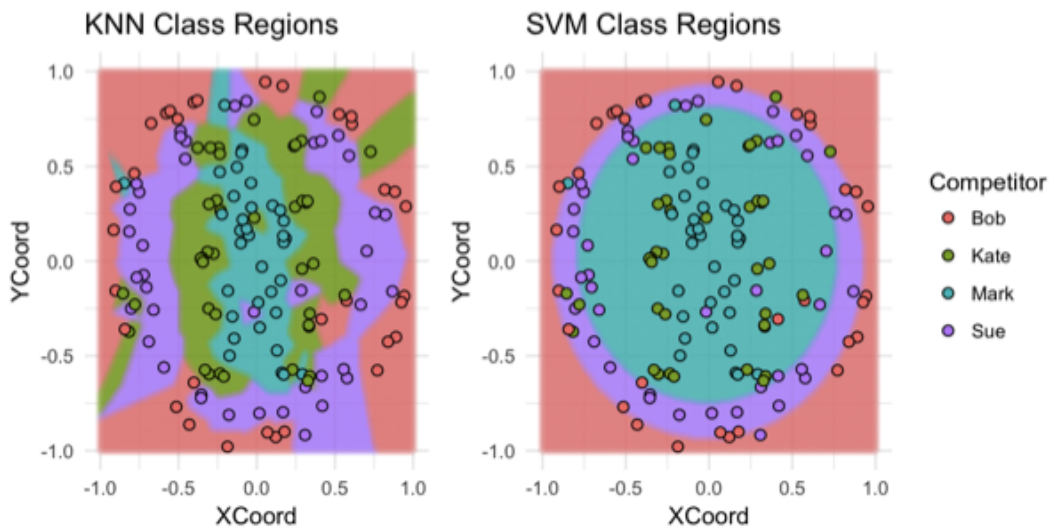
Stacking

- Basic idea: use the output of multiple classifiers as input to a meta-model
- We 'stack' the meta-model on top of the base models



Stacking example

Base models

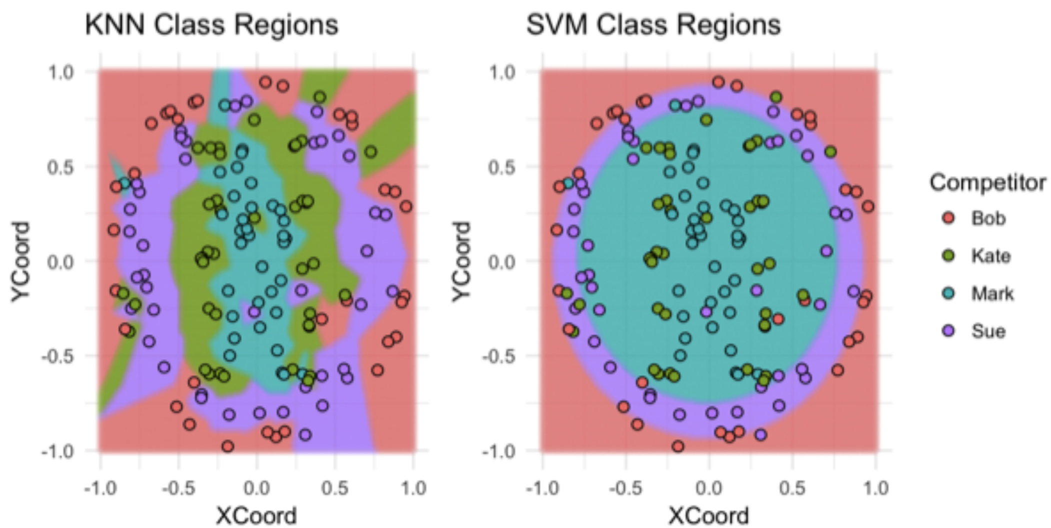


Stacked model

Source: https://gormanalysis.com/guide-to-model-stacking-i-e-meta-ensembling/

Stacking example

Base models



Stacked model

Source: https://gormanalysis.com/guide-to-model-stacking-i-e-meta-ensembling/

Stacking – a naïve approach

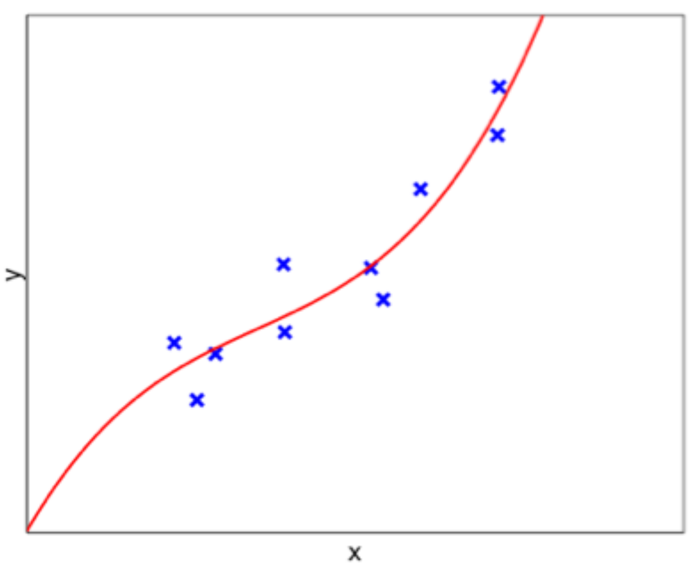
- Let's consider the regression case
- Base model predictions are $f_1(\mathbf{x}), \ldots, f_L(\mathbf{x})$
- Meta learner could be a simple linear combination

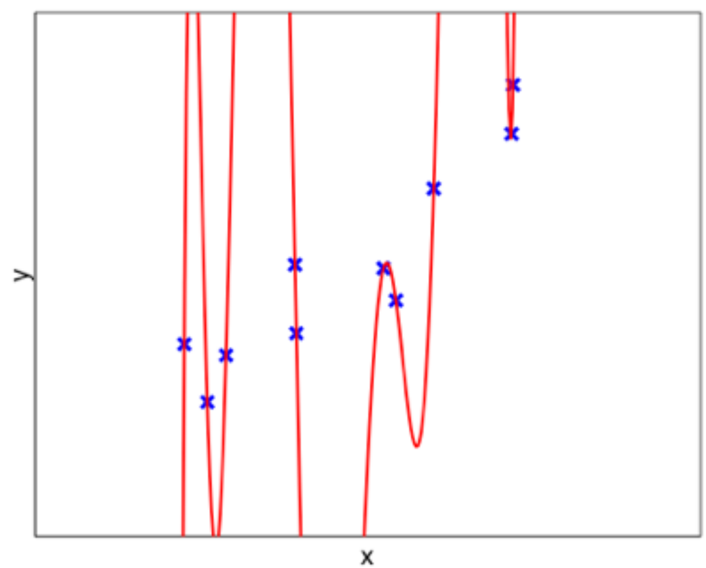
$$f_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^{L} w_i f_i(\mathbf{x})$$

- If we could choose w to minimize true error, the stacked model would always be at least as good as any base model!
  - Worst case we set all w_i to 0 except that of the best base model
- But what if we minimize the train error instead?

Stacking – a naïve approach

- Consider the following base models





- What weights would minimize train set error?
- Does that yield good generalization error for the meta model?

Stacking – a naïve approach

- Naïve implementation of stacking prefers over-fitted models
- Underlying problem: the outputs of the base models have been adapted to the labels.
- Thus, inputs of the meta model are not representative of the inputs it will get at test-time.
- To avoid preference for overfitted models, inputs to the meta-model should not have seen the labels for the data points themselves

Stacking – second attempt

- Naïve implementation of stacking prefers over-fitted models
- Underlying problem: the outputs of the base models have been adapted to the labels.
- Thus, inputs of the meta model are not representative of the inputs it will get at test-time.
- To avoid preference for overfitted models, inputs to the meta-model should not have seen the labels for the data points themselves

Stacking – second attempt

Fold	x_1	x_2	f_1	f_2	f_{meta}	y
1	0.8	0.1				1.5
2	0.9	0.9				0.8
4	0.1	0.9				-0.7
3	0.6	0.8				1.3
4	0.1	0.1				0.1
2	0.3	0.4				0.1
3	0.5	0.9				0.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0.9	1.0				1.0

Stacking – second attempt

- Train base models on folds 2-4 and predict for fold 1

Fold	x_1	x_2	f_1	f_2	f_{meta}	y
1	0.8	0.1				1.5
2	0.9	0.9				0.8
4	0.1	0.9				-0.7
3	0.6	0.8				1.3
4	0.1	0.1				0.1
2	0.3	0.4				0.1
3	0.5	0.9				0.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0.9	1.0				1.0

Train data

Stacking – second attempt

- Train base models on folds 2-4 and predict for fold 1

Train data

Fold	x_1	x_2	f_1	f_2	f_{meta}	y
1	0.8	0.1	2.4	0.5		1.5
2	0.9	0.9				0.8
4	0.1	0.9				-0.7
3	0.6	0.8				1.3
4	0.1	0.1				0.1
2	0.3	0.4				0.1
3	0.5	0.9				0.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0.9	1.0	0.0	0.2		1.0

Stacking – second attempt

- Train base models on folds 1,3,4 and predict for fold 2

Train data

Fold	x_1	x_2	f_1	f_2	f_{meta}	y
1	0.8	0.1	2.4	0.5		1.5
2	0.9	0.9	-2.2	0.7		0.8
4	0.1	0.9				-0.7
3	0.6	0.8				1.3
4	0.1	0.1				0.1
2	0.3	0.4	0.2	-0.3		0.1
3	0.5	0.9				0.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0.9	1.0	0.0	0.2		1.0

Stacking – second attempt

- Train base models on folds 1,2,4 and predict for fold 3

Train data

Fold	x_1	x_2	f_1	f_2	f_{meta}	y
1	0.8	0.1	2.4	0.5		1.5
2	0.9	0.9	-2.2	0.7		0.8
4	0.1	0.9				-0.7
3	0.6	0.8	1.3	1.2		1.3
4	0.1	0.1				0.1
2	0.3	0.4	0.2	-0.3		0.1
3	0.5	0.9	0.5	0.7		0.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0.9	1.0	0.0	0.2		1.0

Stacking – second attempt

- Train base models on folds 1,2,3 and predict for fold 4

Train data

Fold	x_1	x_2	f_1	f_2	f_{meta}	y
1	0.8	0.1	2.4	0.5		1.5
2	0.9	0.9	-2.2	0.7		0.8
4	0.1	0.9	-1.1	-0.7		-0.7
3	0.6	0.8	1.3	1.2		1.3
4	0.1	0.1	-0.5	0.3		0.1
2	0.3	0.4	0.2	-0.3		0.1
3	0.5	0.9	0.5	0.7		0.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0.9	1.0	0.0	0.2		1.0

Stacking – second attempt

- Now, we can train the meta-model on the data in the base model outputs paired with the target label
- Any base-model output is now a good indication of test-time behavior
- If the meta-model has free parameters itself, can cross-validate using the same folds
- Usually, the meta-model is relatively simple (e.g. linear regression or logistic regression)

Testing the stacked model

- To test the stacked model, again we set aside a test set from the very beginning
- Have several versions of the base models from cross-validation!
- Two approaches:
  - Retrain the base models on the whole dataset
    - Possible disadvantage: slightly different input to meta-model
  - Use an average of the trained base models
    - Possible disadvantage: time cost
- Then feed the base model predictions into the trained meta-model

Comparison to model selection

- If we force meta-learner to use just one base model (with weight 1) and set all other weights to 0, this is equivalent to selecting the best model with cross-validation
- More expressive meta-models (e.g. linear / logistic regression) can leverage the relative strength of multiple models
- A very complex meta-model (e.g. decision tree) could again easily overfit
- Could use cross-validation on the meta-level to ensure good generalization properties

Effectiveness of stacking

- Stacking generally improves performance, but not by much
- Additional cost of training and evaluating multiple models
- Depending on conditions, it might or might not be worth it:
  - If interpretability or latency are important consideration, stacking might not help you much.
  - In competitions where a small gain is important and time cost is not so much of an issue, it is usually effective!
  - Quite useful in collaborative approaches where everyone can integrate their own model in overall system

Any questions about stacking?

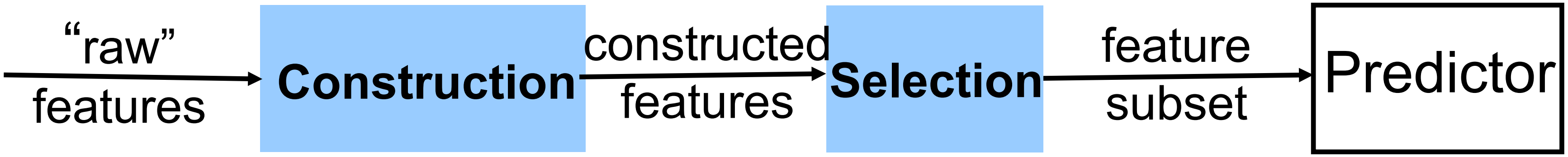
Steps to solving a supervised learning problem

1. Decide what the input-output pairs are.
2. Decide how to encode inputs and outputs.
   - This defines the input space *X* and output space *Y*.
3. Choose a class of hypotheses / representations *H.*
   - E.g. linear functions.
4. Choose an error function (cost function) to define best hypothesis.
   - E.g. Least-mean squares.
5. Choose an algorithm for searching through space of hypotheses.

Today and Monday: deciding on what the inputs are

So far: we have been focusing on this

Feature Extraction Steps



Ideas for feature construction?

A few strategies we discussed

- Use domain knowledge to construct "ad hoc" features.
- Normalization across different features, e.g. centering and scaling with $x_i = (x'_i - \mu_i) / \sigma_i$.
- Normalization across different data instances, e.g. counts/histogram of pixel colors.
- Non-linear expansions when first order interactions are not enough for good results, e.g. products x_1x_2, x_1x_3, etc.
- Other functions of features (e.g. sin, cos, log, exponential etc.)
- Selecting features by predictive value
- Regularization (lasso, ridge).

Delving deeper…

- Use domain knowledge to construct “ad hoc” features.
  - Look at some domain-specific features for language data
- Non-linear expansions
  - What are some good expansions to try?
  - How to select useful expansions?
- Selecting features by predictive value
  - Finding good subsets of features (next lecture…)
- New: combine multiple features into a single feature
  - Dimensionality reduction (next lecture…)

Feature Construction

Why do we do feature construction?

- Increase predictor performance.
- Reduce time / memory requirements.
- Improve interpretability.

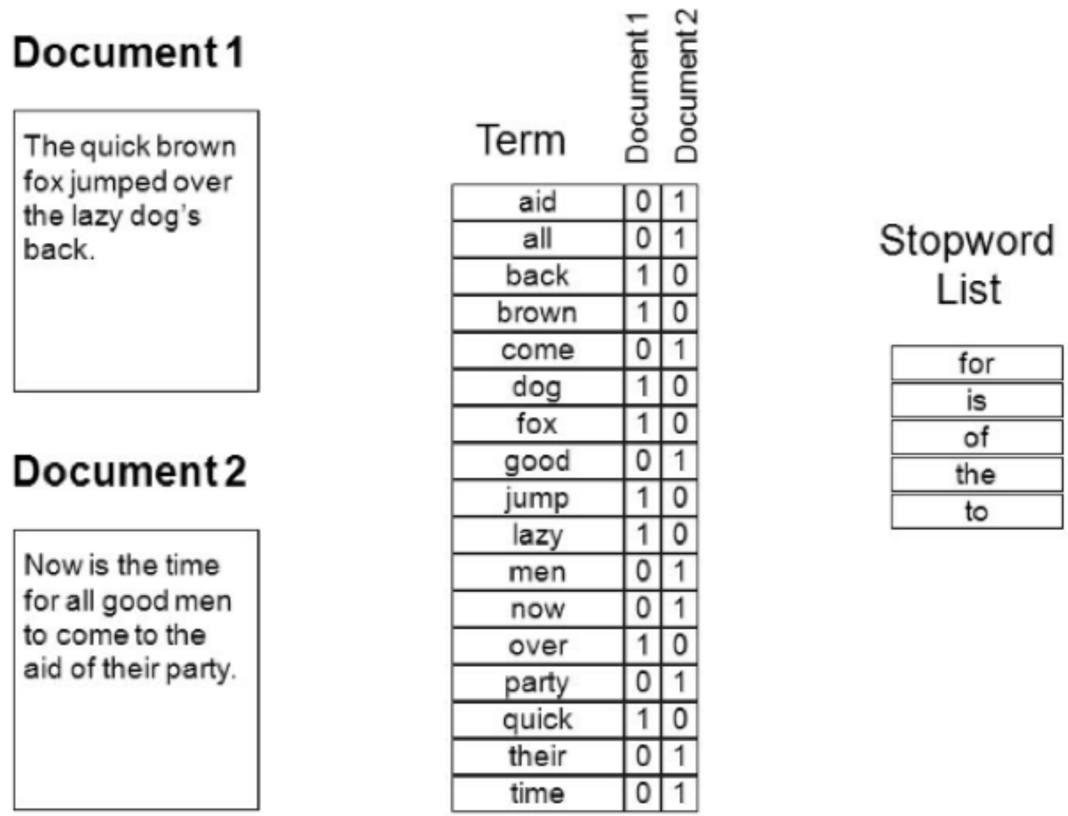
But: Don’t lose important information!

Features for modelling natural language

- Words
- TF-IDF
- N-grams
- Word embeddings
- Useful Python package for implementing these:
  - Natural Language toolkit: http://www.nltk.org/

Words

- Binary (present or absent)
- Absolute frequency
  - i.e., raw count
- Relative frequency
  - i.e., proportion
  - document length



More options for words

- Stopwords
  - Common words like “the”, “of”, “about” are unlikely to be informative about the contents of a document. Remove!
- Lemmatization
  - Inflectional morphology: changes to a word required by the grammar of a language
    - e.g., “perplexing” “perplexed” “perplexes”
    - (Much worse in languages other than English, Chinese, Vietnamese)
  - **Lemmatize** to recover the canonical form; e.g., “perplex”

Term weighting

- Words that occur more often influence decision boundary more
- Not all words are equally important.
- What do you know about an article if it contains the word
    - *the*?
    - *penguin*?

TF*IDF (Salton, 1988)

- **Term Frequency Times Inverse Document Frequency**
- A term is important/indicative of a document if it:
  1. Appears many times in the document
  2. Is a relative rare word overall
- TF is usually just the count of the word
- IDF is a little more complicated:
  - $IDF(t, Corpus) = \log \frac{\#(\text{Docs in } Corpus)}{\#(\text{Docs with term } t) + 1}$
    - Need a separate large training corpus for this
- Originally designed for document retrieval

N-grams

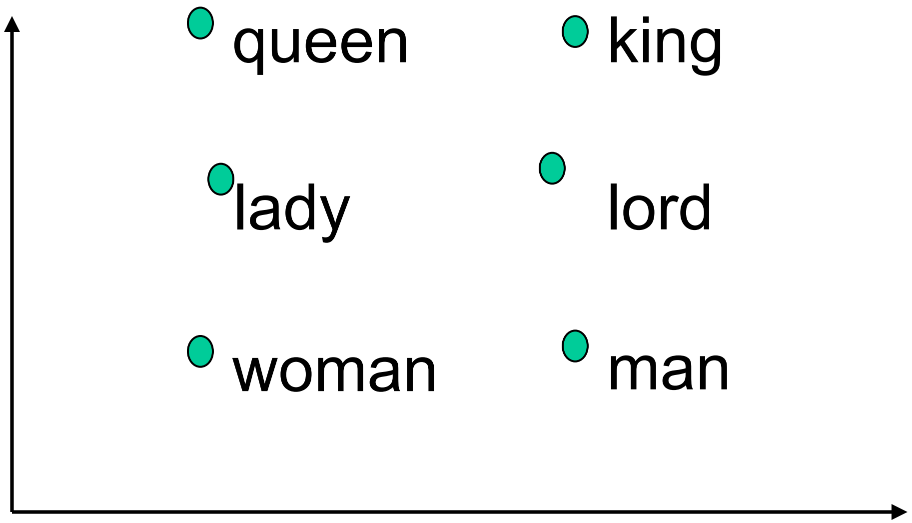
- Use sequences of words, instead of individual words
- e.g., *… quick brown fox jumped …*
  - Unigrams (i.e. words)
    - quick, brown, fox, jumped
  - Bigrams
    - quick_brown, brown_fox, fox_jumped
  - Trigrams
    - quick_brown_fox, brown_fox_jumped
- Usually stop at N <= 3, unless you have lots and lots of data

Word embedding models

- Problems with above:
  - Number of features scales with size of vocabulary!
  - Many words are semantically related and behave similarly (e.g., *freedom* vs *liberty*)
- Word embedding models can help us:
  - Embed each word into a fixed-dimension space
  - Learn correlations between words with similar meanings

Word embedding models

- Main idea: Represent each word by a vector
- The vector could encode different properties of the word, that should be locally consistent



word2vec (Mikolov et al., 2013)

- Intuition:
  - Words that appear in similar contexts should be semantically related, so they should have similar word vector representations
- Actually two models:
  - **Continuous bag of words (CBOW)** – use context words to predict a target word
  - **Skip-gram** – use target word to predict context words
- In both cases, the representation that is associated with the target word is the embedding that is learned.

word2vec Architectures

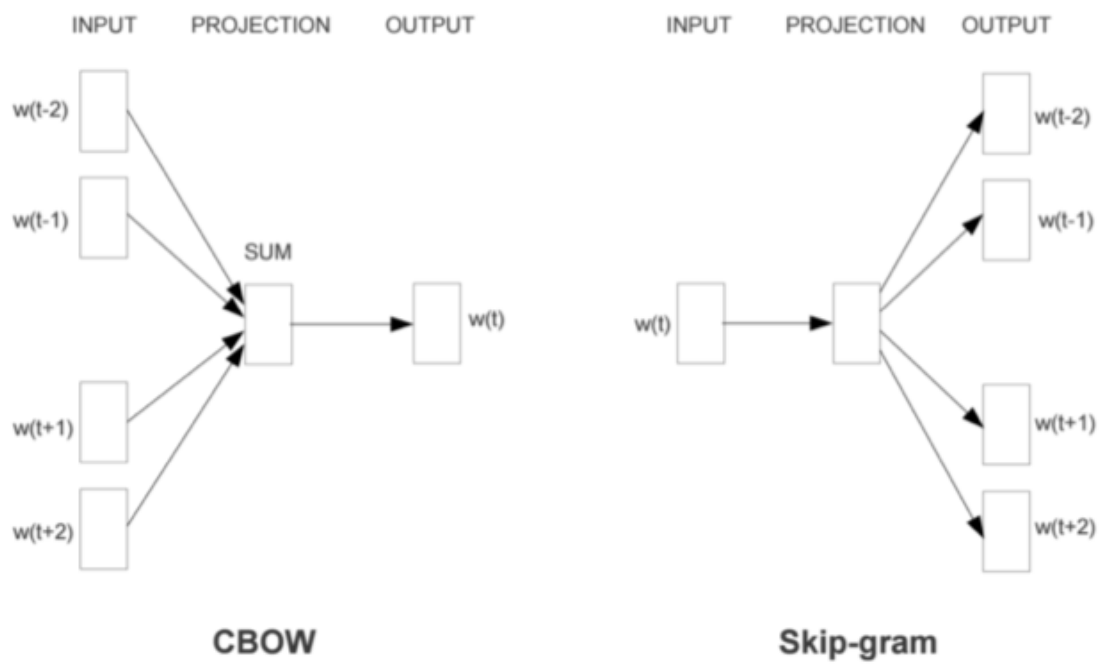


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Practical word2vec

- Pre-trained word embeddings are available for download online
  - Google News corpus
  - Freebase entities
- Can also train your own word2vec model, if you have more specialized data
- Training is done using gradient descent
- Another popular option:
  - GloVe (Pennington et al., 2014)

What you should know

- Basic idea of stacking model and how to use it
- Main strengths and limitations of stacking
- Know some features for natural language data