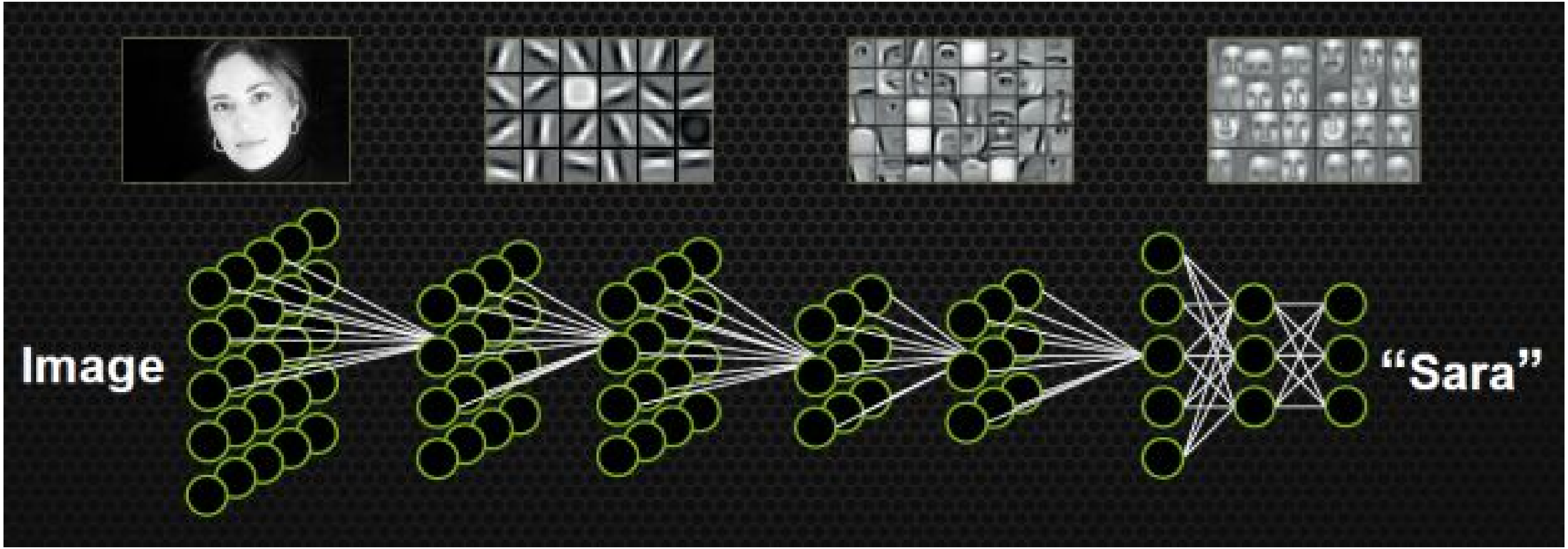
Word Embeddings

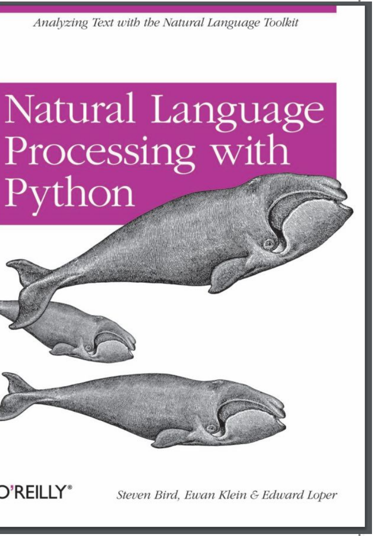
A Case Study...

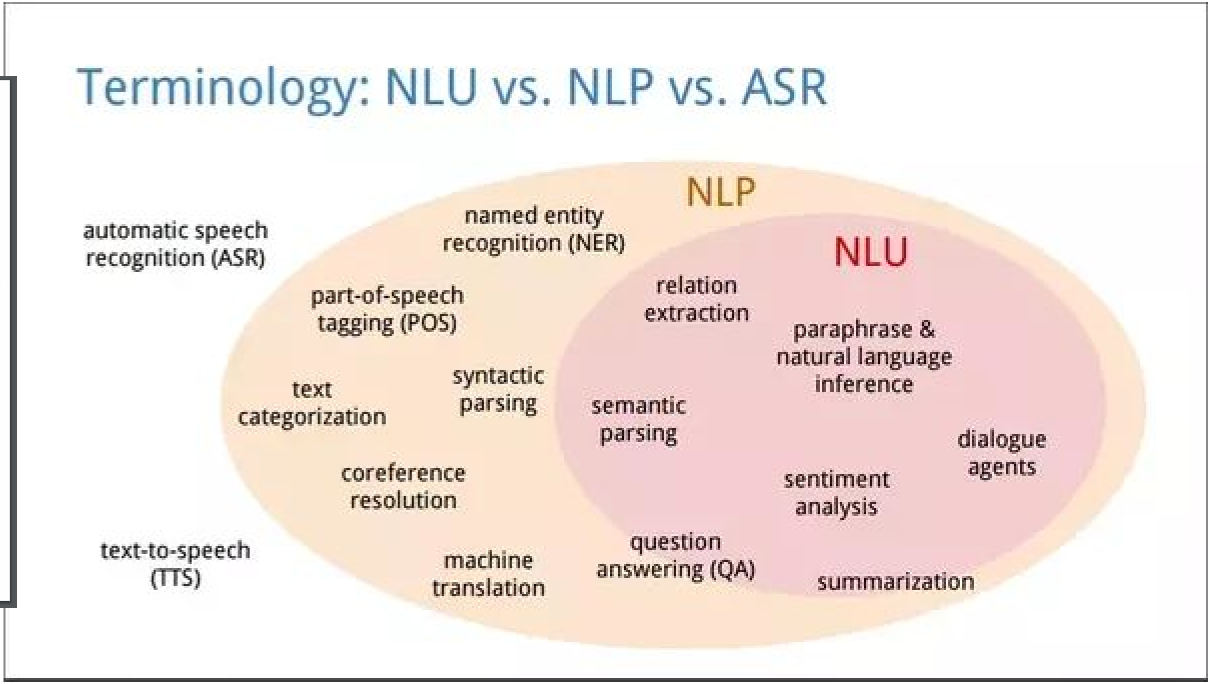
Obligatory Photo of Neural Network (*Everyone loves Neural Nets*)



Dont get your hopes up - we’re not covering this today

Natural Language Processing (aka NLP)

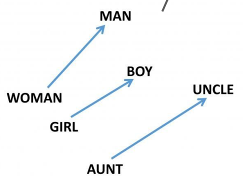




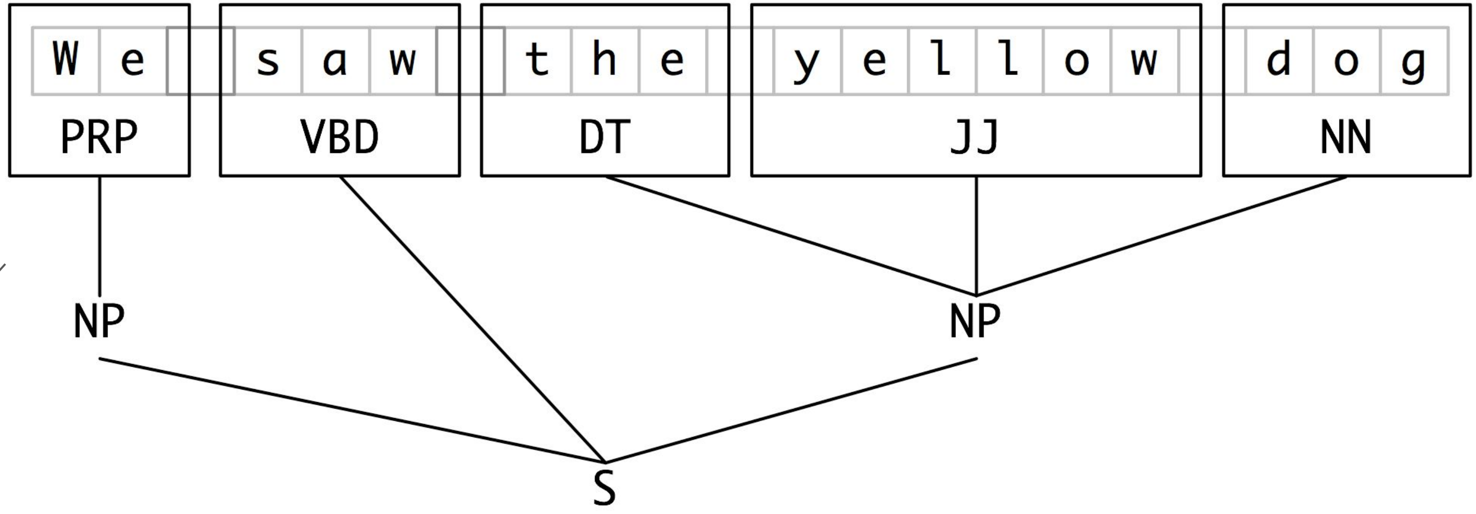
NER

Figure 1: An example of NER application on an example text

Word Embeddings



POS

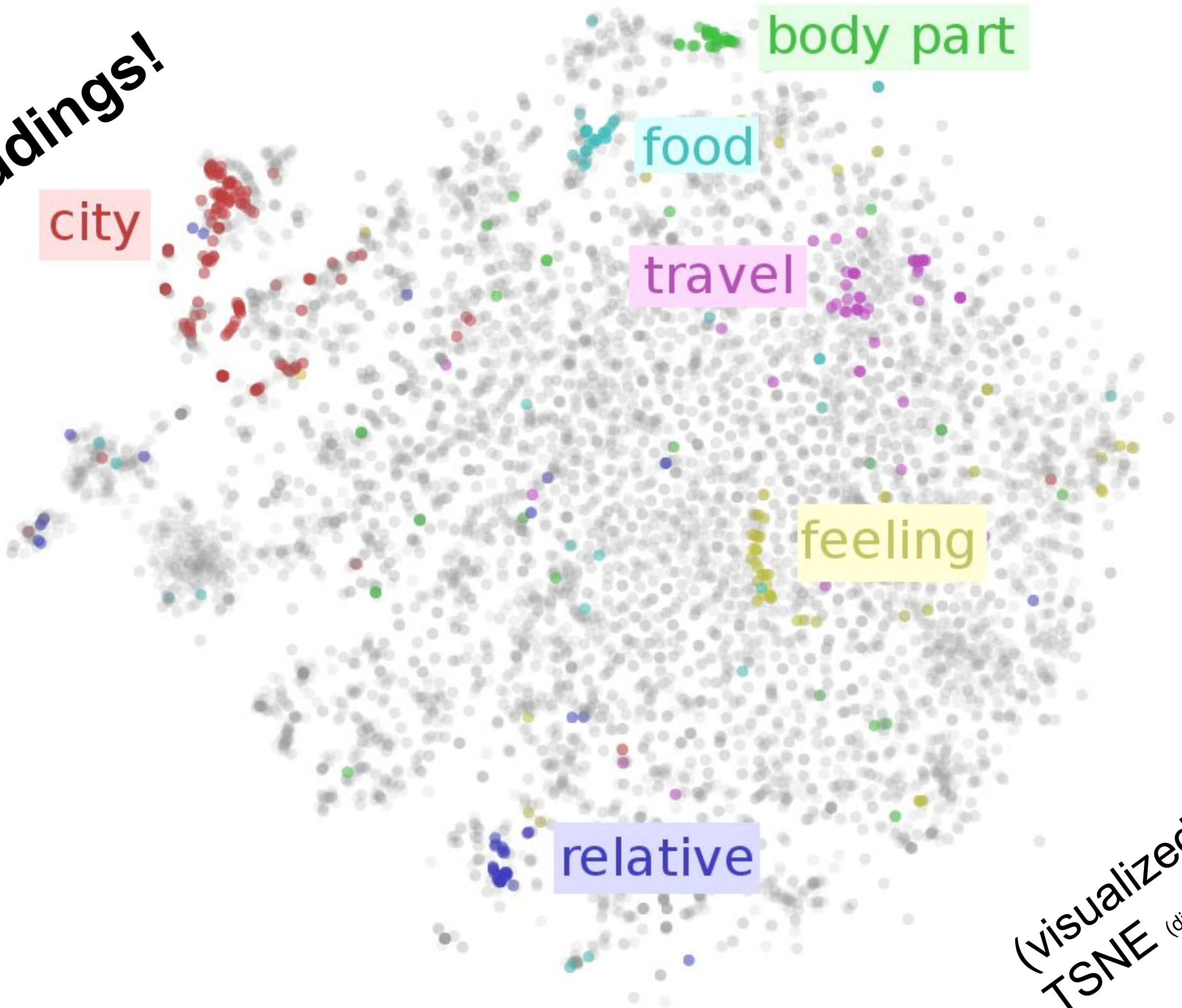


Semantic Subject Recognition

- Like NER/POS - extracts information from text which we can use as features
- Utilizes the semantic data embedded in word embeddings

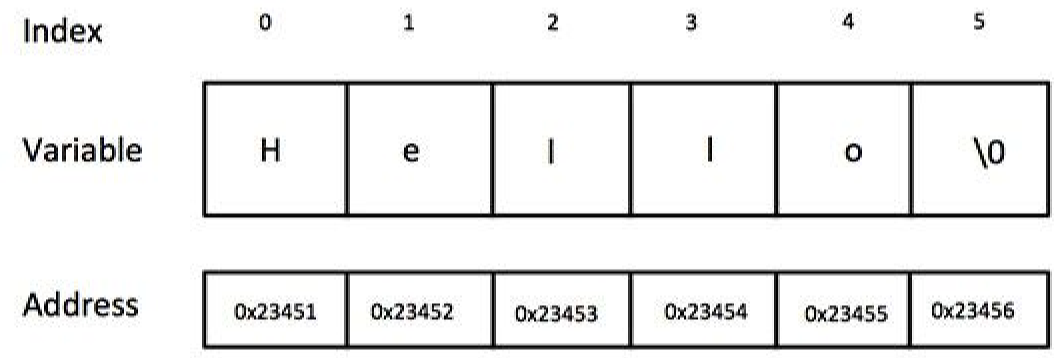
...

Word Embeddings!



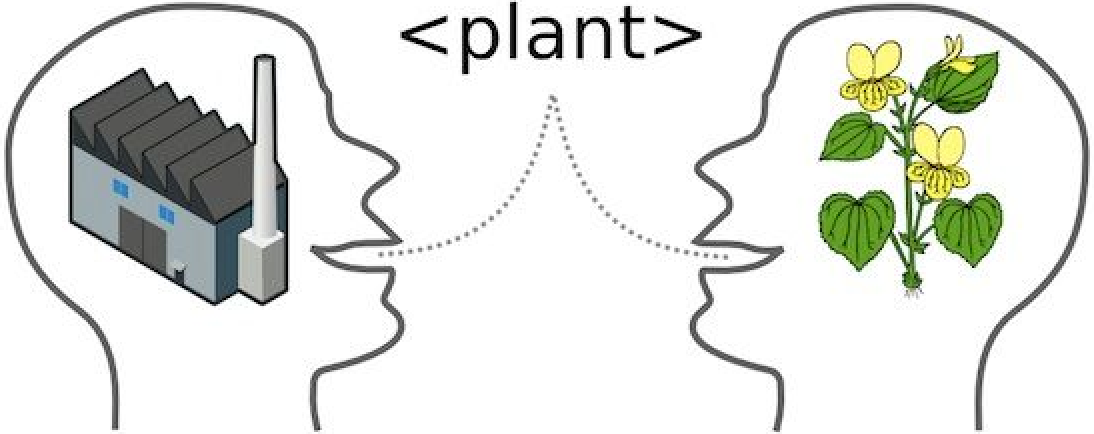
(visualized w/ TSNE (dimensionality reduction))

Why? All a computer sees is letters



One. Two. Three. Four. Five. Hundered. Thousand. (may as well be random.)

Except not even unique...

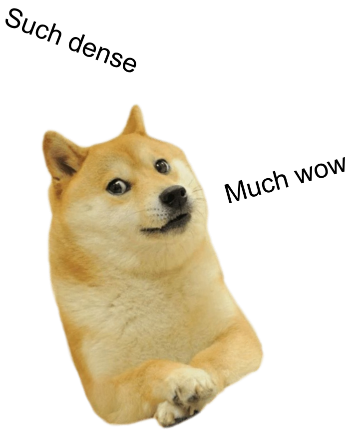


(We’ll ignore this though and be happy)

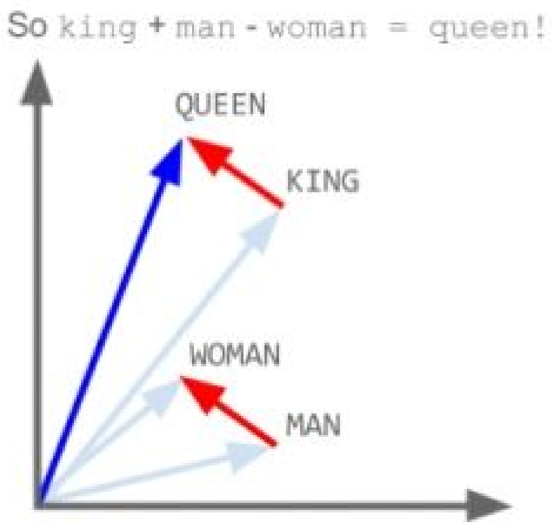
Bag of Words (Sparse)

	eveyrthing	interesting	learning	lerning	like	Machien	machine	not	predicts	problems	solving	sure	What
1	0	1	0	0	1	0	0	0	0	1	1	0	0
2	0	0	1	0	0	0	1	0	0	0	0	0	1
3	0	0	0	0	0	0	0	1	0	0	0	1	0
4	1	0	0	1	0	1	0	0	1	0	0	0	0

Word Vector




```
array([-0.05370992, -0.01796519, -0.13489808, -0.00400016, -0.01886696, -0.01855153, -0.0590021,
        0.04081715, -0.01833461,  0.01756546,  0.03327245, -0.05934121,  0.13161591,  0.09330324,
       -0.0576504,  -0.06708767, -0.14609909, -0.06536276,  0.04444694,  0.06847347,  0.0038306,
        0.08097503,  0.1450344,  -0.0606285,  -0.05798667, -0.02206576, -0.02363058, -0.01232632,
        0.04450377,  0.0536673,   0.14820194, -0.03370629,  0.00571465,  0.10534635,  0.06061808,
        0.05924838,  0.01724624,  0.00195224,  0.08353445,  0.07976257,  0.05860237,  0.02358891,
       -0.14326403,  0.02775767, -0.05105672,  0.07834172,  0.01482512, -0.10593458, -0.07428473,
       -0.00392154, -0.06843369, -0.0286187,   0.03206379,  0.01065825,  0.0212142,  -0.038199,
       -0.01821716, -0.16778027,  0.06967456,  0.02450488, -0.03385879,  0.0763156,  -0.00977732,
       -0.0511325,  -0.00714402,  0.07367945, -0.0687027,   0.00737988, -0.00394427,  0.08146569,
       -0.03385974,  0.03460994,  0.00039784, -0.0203238,   0.03031046, -0.04941517,  0.09776281,
        0.17635746, -0.00446904,  0.05661129, -0.05412859,  0.04316155, -0.07147998,  0.05980725,
       -0.06233541,  0.10460561,  0.00153925, -0.04334057,  0.0265348,   0.03904583,  0.06974371,
       -0.02253748, -0.00371694,  0.03108814, -0.08722486,  0.08058666,  0.08066339,  0.06889972,
        0.05318894,  0.0111025,   0.04847362,  0.04241608, -0.02344587, -0.11333624, -0.01625354,
        0.10140302,  0.03682268,  0.09101,    -0.01545408,  0.0857216,  -0.0635886,   0.01903083,
        0.06806806, -0.06100928,  0.08224337,  0.01855342,  0.01142929,  0.0219663,  -0.11795305,
       -0.05691156, -0.03229586,  0.07092301, -0.10715461, -0.07458216,  0.07924633, -0.08229263,
        0.20106314,  0.12279814,  0.03754162, -0.01622134,  0.06508806,  0.06969255, -0.02829286,
       -0.02122651, -0.11400309,  0.07765214,  0.03194822,  0.04968892,  0.04011241,  0.02989273,
       -0.04679892, -0.13507046,  0.02070364, -0.01047164, -0.03619466, -0.05266512, -0.12246187,
       -0.00721033, -0.10127704,  0.00698299, -0.04904006,  0.06502365, -0.01203647, -0.06826507,
        0.0650928,   0.01393946,  0.13613988,  0.06799417, -0.03203158,  0.02859124, -0.07497147,
        0.018202,   -0.07249352,  0.1334185,  -0.02979043,  0.02842926,  0.09712628, -0.08914774,
        0.15470658, -0.03547382,  0.15360495, -0.01643541, -0.09154803,  0.16215466,  0.14822088,
       -0.01966358, -0.04322122, -0.08516653,  0.02396685, -0.0373105,   0.07382059,  0.15486667,
        0.01114797,  0.01211035, -0.09367077,  0.02892656,  0.10523268, -0.06287628, -0.05812117,
       -0.00592967,  0.01626207,  0.07094574, -0.06422988, -0.01778995, -0.09563628, -0.10500913,
```

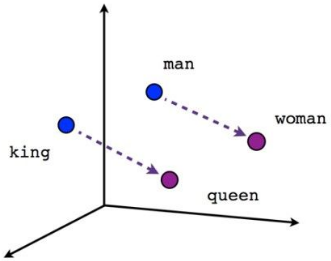


E.g.:

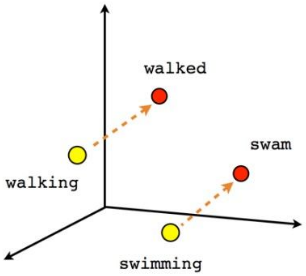
Man = <0, 0>
King = <0, 1>
Woman = <1, 0>
Queen = <1, 1>

King - Man + Woman = <1, 1>

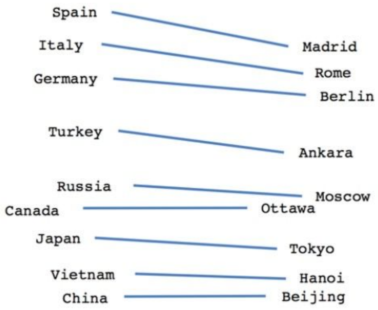
<--- See the femininity difference (vector)?



Male-Female



Verb tense



Country-Capital

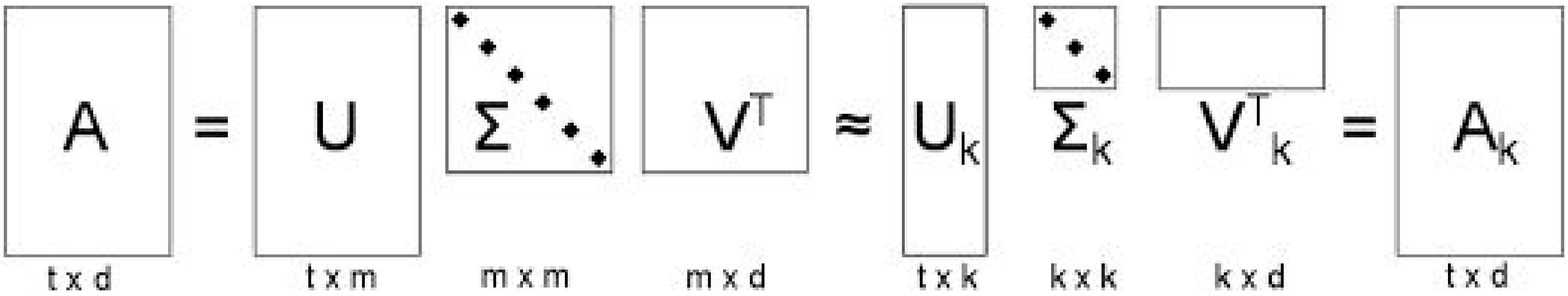
Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Firths Hypothesis - Basis of all embedding techniques

“You shall know a word buy the company it keeps…” - Firth, 1957

- LSA was first (SVD)



Types of Embedding Methods

Count Based Techniques (Use SVD on co-occurrence matrix)

- LSA (first word embedding technique documented)
- GloVe

Predictive Techniques

- Word2Vec

Word2Vec

- Architecture w/ two word embedding models introduced by Mikolov et. al in 2013.
- For predictive models, Firth's : "Words are conditionally dependent on the words that came before them"
- Predictive model, not the first predictive language model nor first NN pred. model.

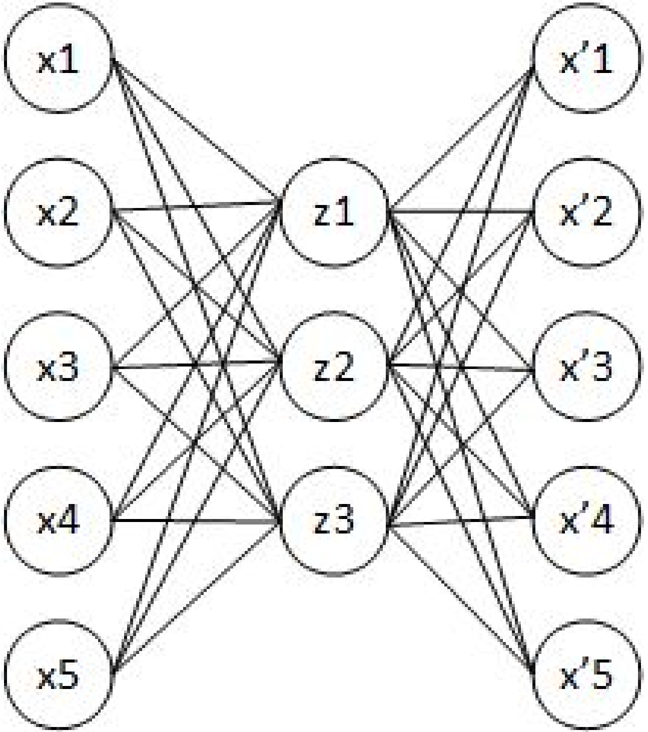
CBOW -vs- SKIP GRAM

- CBOW (averages)
    - Predict target word based on context window
- SkipGram (finer tuned)
    - Predict ***a*** context word based on target word

- “For the skipgram, the direction of the prediction is simply inverted, i.e. now we try to predict P(citizens | X), P(of | X), etc. This turns out to learn finer-grained vectors when one trains over more data. The main reason is that the CBOW smooths over a lot of the distributional statistics by averaging over all context words while the skipgram does not. With little data, this "regularizing" effect of the CBOW turns out to be helpful, but since data is the ultimate regularizer the skipgram is able to extract more information when more data is available.” - Smart internet man
- Mikolov:
    - Skip-gram: works well with small amount of the training data, represents well even rare words or phrases.
    - CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words

- SkipGram is more precise. But CBOW is faster.

Autoencoders have the right idea...




IS YOUR NAME
GOOGLE
BECAUSE YOU'RE EVERYTHING
I'VE BEEN SEARCHING FOR

Google™
News

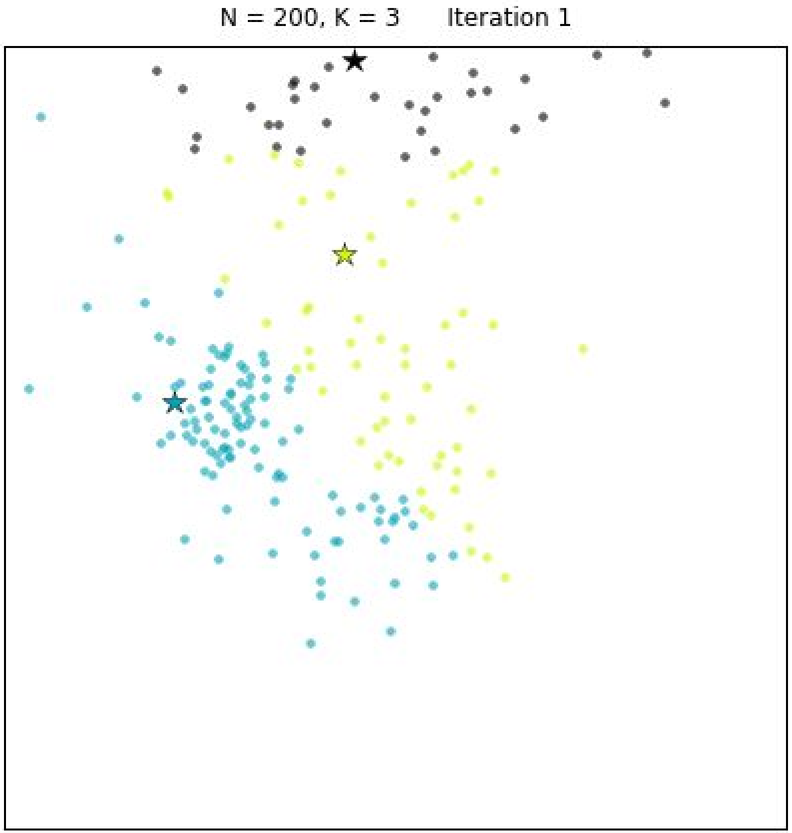
KNN - Clustering

If anyone doesn't know

Clustering -vs- Classification

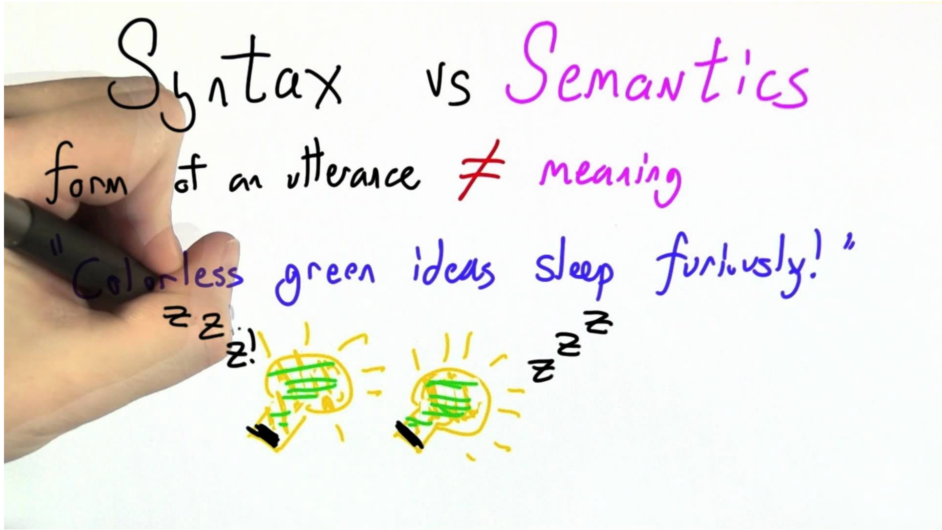
Raise your hand / ask now

Its simple but informative



Semantic Subject Recognition

- Like NER/POS - extracts information from text which we can use as features
- Utilizes the semantic data embedded in word embeddings



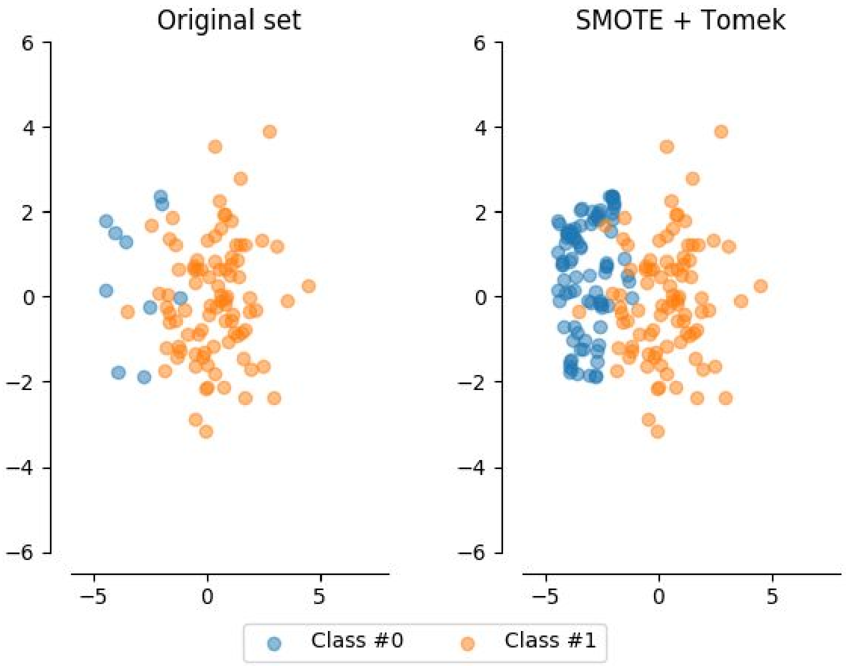
- Trains, Planes, and Automobiles -> Transportation

Sampling Techniques

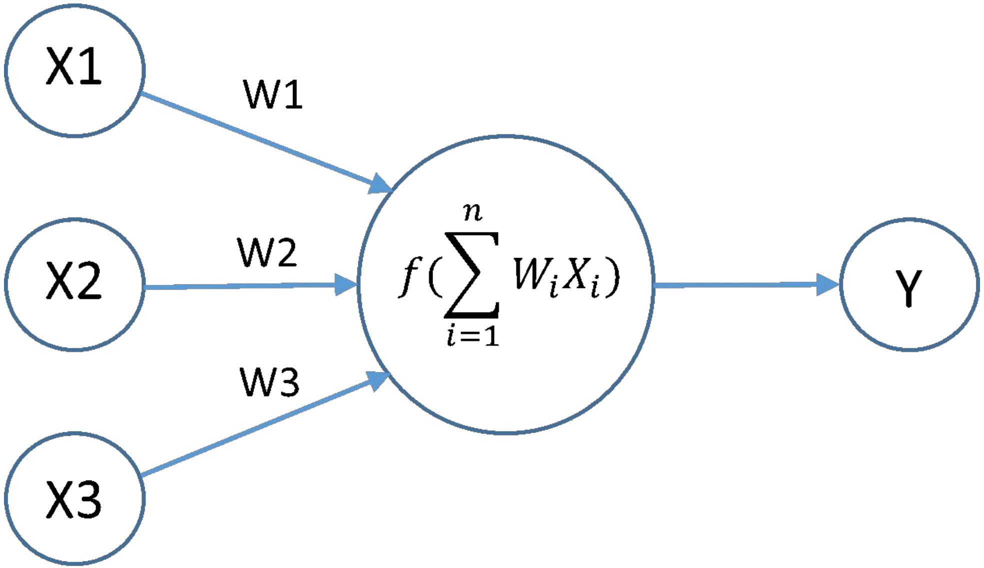
Over

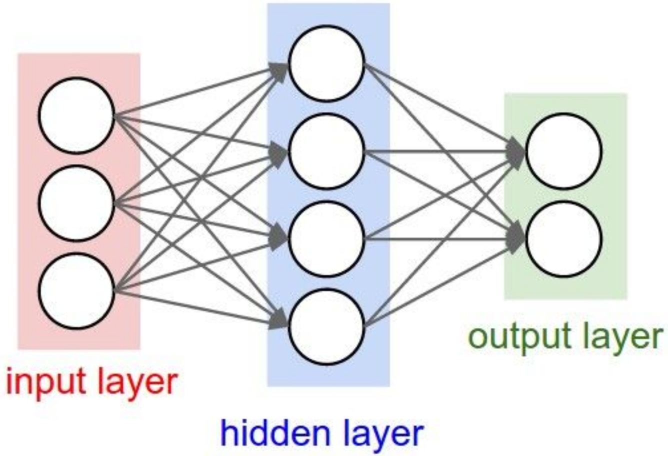
Under

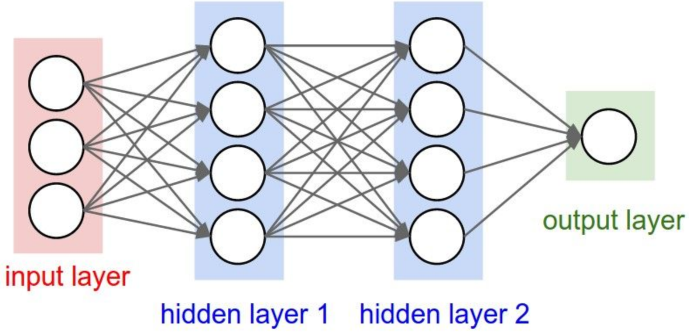
SMOTE!



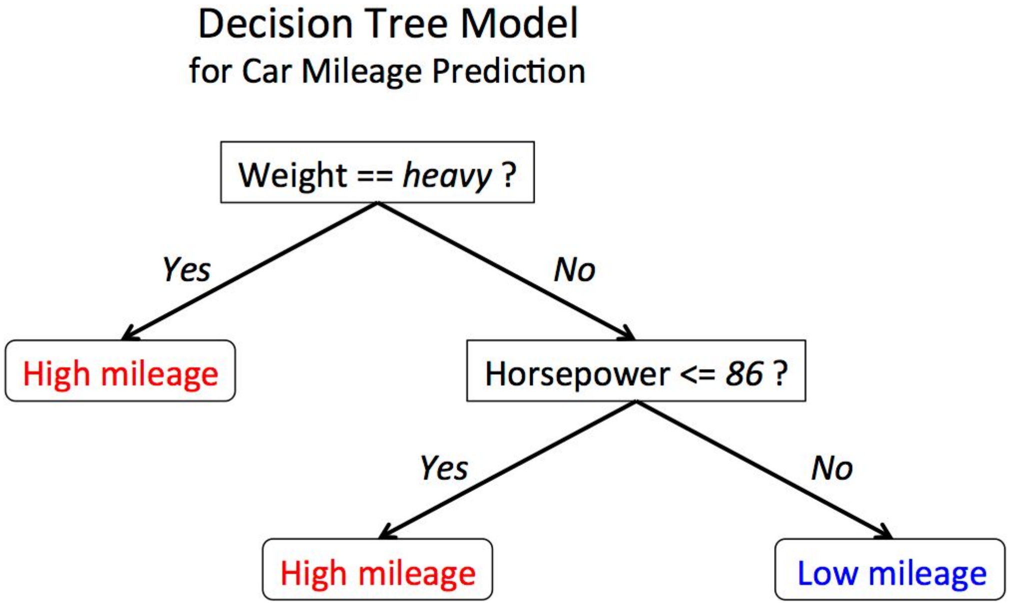
Neural Networks



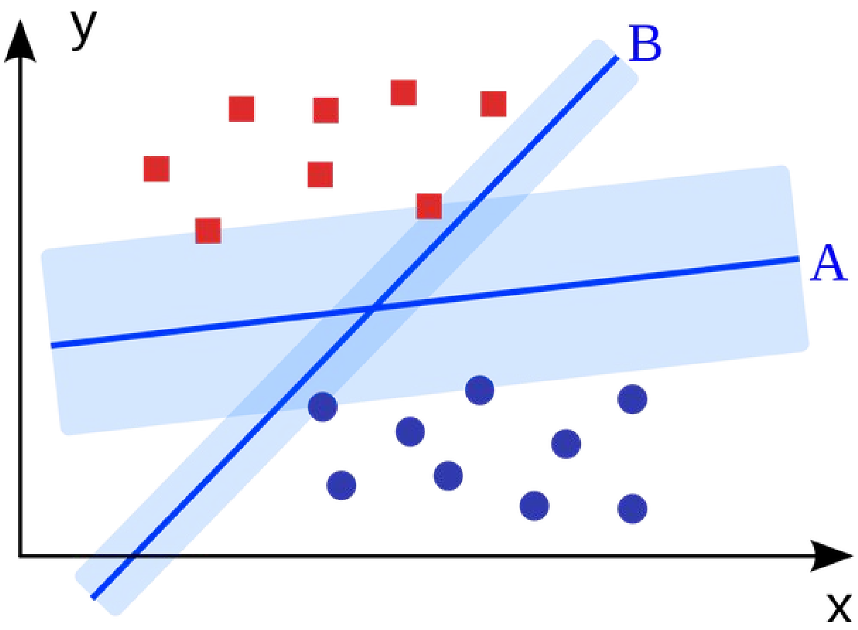




Random Forest



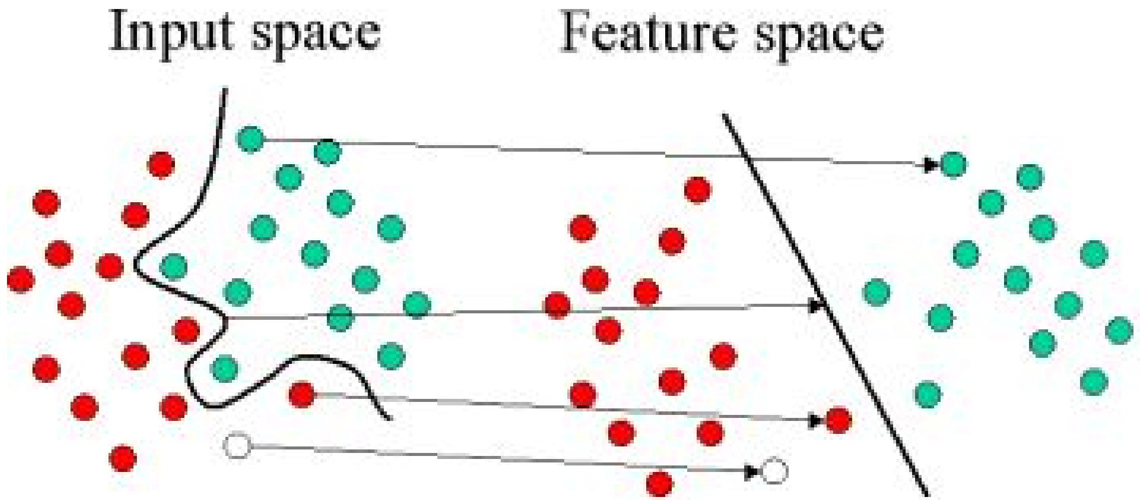
Support Vector Machines

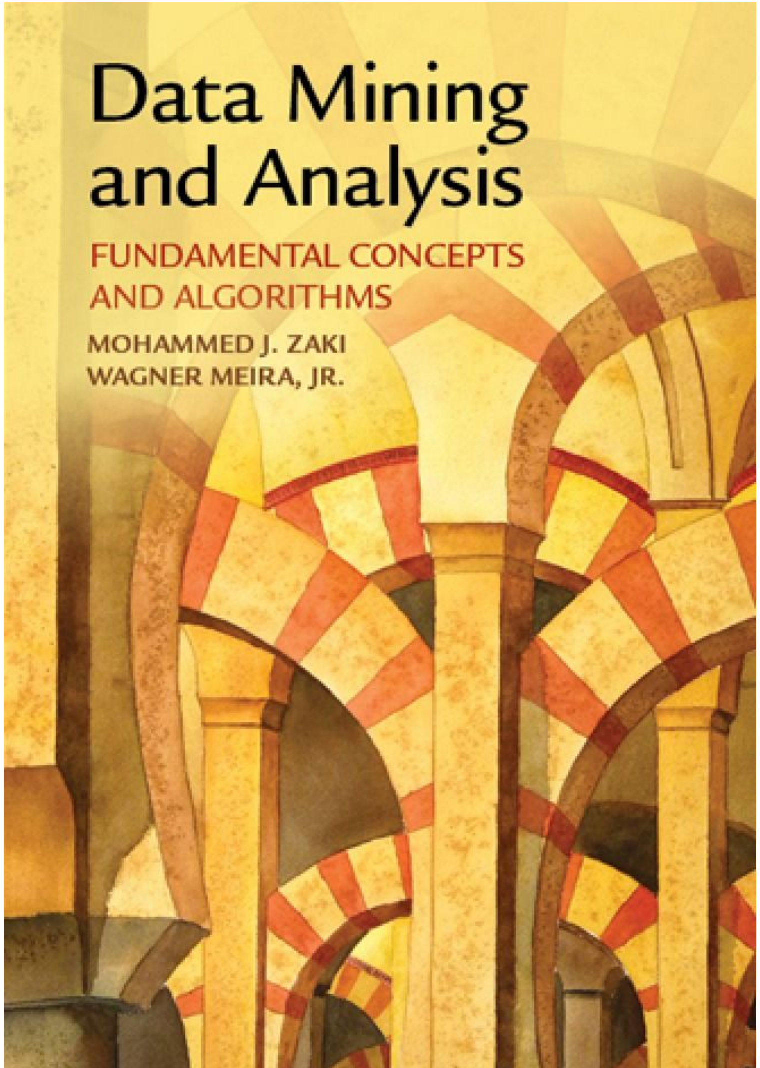


$$\min_{\vec{w},b,\xi} \left\{ \frac{|\vec{w}|^2}{2} + C \sum_{i=1}^{n} (\xi_i)^k \right\}$$
$$s.t. \quad y_i(\vec{w}^T \vec{x}_i + b) \geq 1 - \xi_i, \forall \vec{x}_i \in \mathbf{D}$$
$$and \quad \xi_i \geq 0, \forall \vec{x}_i \in \mathbf{D}$$

$$\max_{\alpha_i} L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j$$
$$s.t. \quad 0 \leq \alpha_i \leq C, \forall i \in \mathbf{D}$$
$$and \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

SVM Kernals - Proven by Maths





+

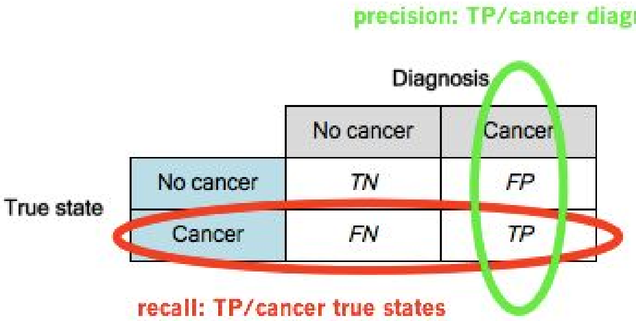
scikit learn

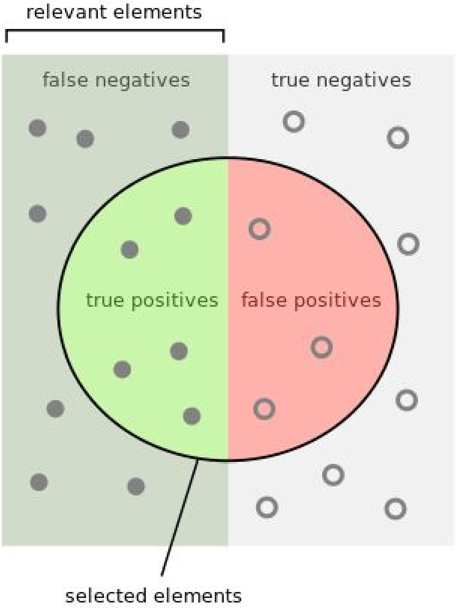
Precision -vs- Recall

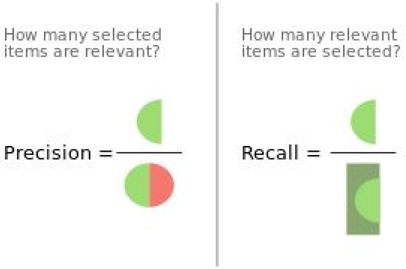
$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2TP}{2TP + FP + FN}$$







So, Semantic Recognition?

1) Label Seed Words
    a) Tree, lilac, oak, etc
2) Iterativly improve labels
    a) Expectation
        i) Classify and evaluate
    b) Maximization
        i) Check if words are misclassified or actually mislabeled
3) ???
4) Profit

i	Pre-Max F1	F1	Prec.	Recall	TP	FP	FN
0	64.73%	77.82%	68.50%	92.81%	1059	487	82
1	68.12%	80.00%	73.29%	88.06%	1136	414	154
2	70.01%	84.06%	80.01%	88.46%	1150	286	150
3	71.55%	79.16%	77.13%	81.29%	995	295	229

TABLE I. Performance statistics for the classifier with maximal F1 score for each iteration after the maximization step as well as the original F1 score pre-maximization.

i	Food	Gardening	Ecosystems	Cumulative	Total FP
2	40.21%	37.06%	6.64%	84.3%	286
3	52.54%	18.64%	5.08%	80.0%	295

TABLE III. Percentage of FP misclassifications of Sister Class words for iterations 2 and 3 for the maximum F1 score classifier after the maximization step.