Approaching (almost) any NLP problem.

```
from scipy import sparse
features = sparse.hstack((num_features, categorical_features))
```

If you want a reproducible pipeline, its better to use scripts.

Generator. — Use if your data is huge.
- Keras.
- is this same as estimator.

- Recursive Elimination of features.
- Based on model — (Random Forest, XGB) → feature_importance_
- (SelectKBest or SelectPercentile) → check this.
  with Mutual Information or Chi-Square.

tsfresh

fuzzy matching: with Similarity.
(approximate String matching).
The closeness of a match is defined by the Edit distance.

Package: for fuzzy features.

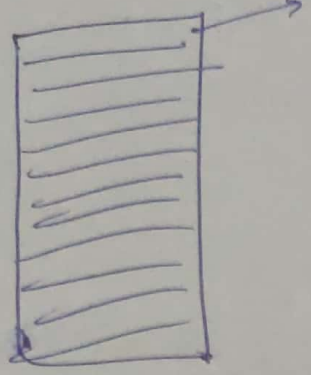
fuzzywuzzy. → you can extract a lot of fuzzy features using fuzzywuzzy.

Extra features, given 2 features. like { QRatio, WRatio, Token set Ratio, Token Sort Ratio, etc, etc.

SVD is also known as LSA (Latent Semantic Analysis)

word2Vec features

→ 2-layered Neural Networks.



→ they give multi-dimensional vector for all the words in any dictionary.

→ gives good insights.

→ Very popular in NLP tasks.

Every word gets a position in space.

dictionary / corpus.

Representing words
Representing sentences } Using word2vec.

↓

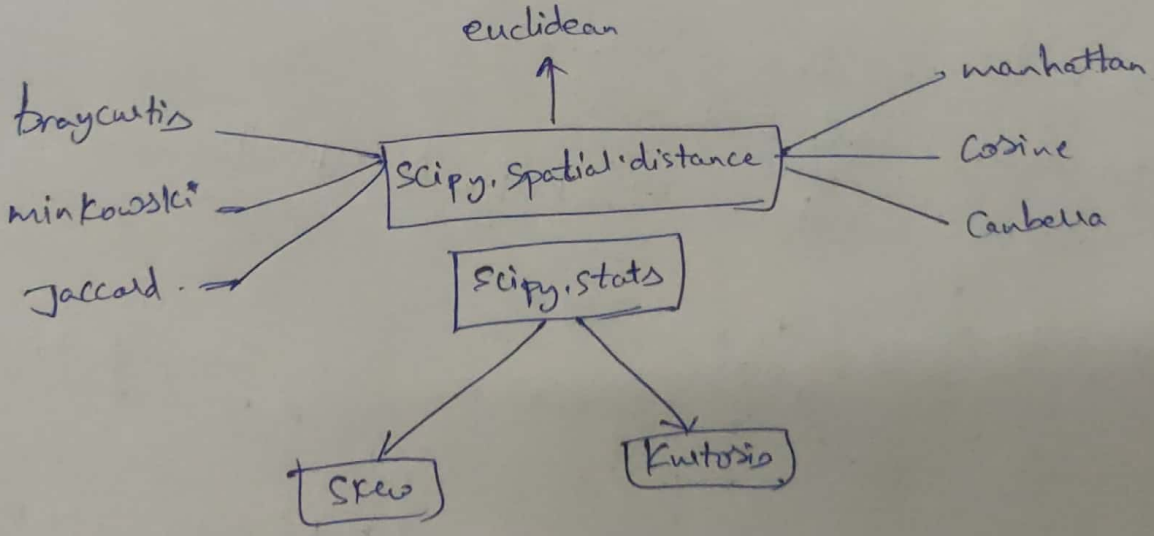
tokenize words and

Another method

Doc2vec.

word vectors of both Q_1, Q_2

word2vec features (Once you get ~~features~~, you can construct below features.)



Word2Vec features: WMD

↓

Word mover's distance.

WMD adapts Earth mover's Distance.

The distance b/w the 2 texts is given by the most needed to move from one sentence to another.

Obama speaks to the media in Illinois
The president greets the press in Chicago.

→ No common words. So Cosine distance is 1. (not useful)

But WMD is low

Train models using

- Basic features + fuzzy features + word2Vec features.
- TFIDF - SVD features

* What is fs 3-4 fs 3-3 ?

fs 3-1
fs 3-2
fs 3-3
fs 3-4
fs 3-5

LSTM → It learns the long term dependencies between words in the sentence.

1-D CNN
Temporal Convolution.

→ length of sample.

```
for i in range(sample_length):
    y[i] = 0
    for J in range(kernel_length):
        y[i] += x[i-J] * h(J)
```

y[i] ← output
x[i-J] ← input
h(J) → kernel

Embedding layers (we already seen).

↳ converts indexes to vectors.

([4],[20]) ⟶ [[0.25, 0.1], [0.6, -0.2]]

Time distributed dense layer

Keras Sub model.

Weights from GloVe, so no need to it to be trainable.

GloVe Embeddings. ⟶ Used them as initializes for weights.

Dimensionality reduction on Co-occurence counts matrix.

Common Crawl.

Handling data before training:

1) Tokenize data
2) Convert text data to sequences
3) Initialize GloVe embeddings.
4) Create the embedding matrix.

* fine-tuning often gives good results

Random weights, Pre-trained weights, Fine Tuned weights.

Hyperparameter tuning ⟶ Use scikit-opt

SKopt

↓

Read more on this.

1) GridSearch
2) Random Search
3) Bayesian optimization
4) Optimize by hand

GP → base estimator