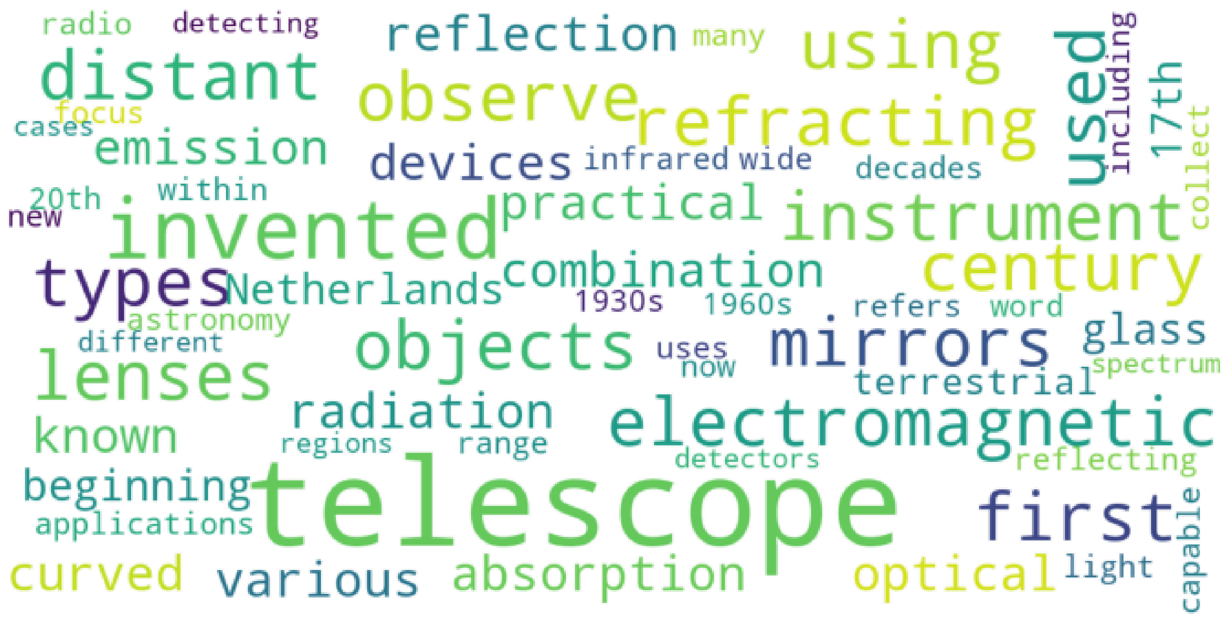
Word Cloud of Text taken:



**1.Word Embeddings tested:**

1.Word2Vec
2. GloVe
3. tf-idf vector
4. Poincare
5. BERT

**Notes:**

[for semantic relations:use Word2Vec
Poincare for hierarchical clustering
Word2Vec: Each word has a fixed representation regardless of the context
BERT produces word representations that are dynamically change according to words around them]

**2. SIMILARITY MEASURES TESTED**

1. Euclidean Distance
2. cosine
3. chebyshev
4. hamming
5. jaccard
6. helsinki
7. minkowski
8. seuclidean
9. sqeuclidean
10. Jensenshannon
11. word mover's distance

**Notes:**

Euclidean distance picked up on magnitude (the texts of similar lengths were grouped as similar) while Cosine distance focuses on similar proportions of words.
[add]

## 2.Keyphrase Extraction

### 1.RAKE

Supervised and Unsupervised(Here, unsupervised)
Time: 2.15
output: [http://localhost:8888/notebooks/Documents/College/Phrase%20Mining/Summary.ipynb](http://localhost:8888/notebooks/Documents/College/Phrase%20Mining/Summary.ipynb)
Scoring/similarity measure : Co-occurrence

### 2. BERT

Unsupervised
Embedding: Bert (transformer encoders)
Similarity Measure: Cosine similarity
Time: 0.234
output: [http://localhost:8888/notebooks/Documents/College/Phrase%20Mining/Summary.ipynb](http://localhost:8888/notebooks/Documents/College/Phrase%20Mining/Summary.ipynb)

### 3.Text Rank

Supervised and Unsupervised(Here, unsupervised)
Word embedding contains POS Tagging: SPACY - POS
Similarity Measure: Any(here, cosine similarity)
Time:  0.057
Output :
Word - Score

```
telescopes - 1.862714039748131
telescope - 1.8205563354276895
objects - 1.5098593171296297
mirrors - 1.3742266010802466
century - 1.350533339687459
lenses - 1.2885601663961035
types - 1.2147693811081592
regions - 1.107030520983646
spectrum - 1.0868444264069264
instruments - 1.0843745866402115
devices - 1.0591117283950617
applications - 1.0
```

### 4. TF-IDF

Supervised (weights)
Similarity Measure: tf-idf
Time:
output: [http://localhost:8888/notebooks/Documents/College/Phrase%20Mining/Summary.ipynb](http://localhost:8888/notebooks/Documents/College/Phrase%20Mining/Summary.ipynb)

### Observations:

Tf-idf gives relevant keywords
Rake give relevant key phrases

**Introduction**
A word embedding is a representation for text where words with the same meaning have a similar representation.  Each word is mapped to one vector and the vector values are either predefined or based on training using neural networks.
Distributed representations of words learned by neural networks:
As it was previously shown that they perform significantly better than LSA for preserving linear regularities among words [20, 31]; LDA moreover becomes computationally very expensive on large data sets.

1.Word2Vec
Word2Vec is a statistical method for learning word embeddings from a text corpus.
The algorithm accepts text corpus as an input and outputs a vector representation for each word. Training of the embedding is done using a two layer neural network. It is used to capture the syntactic and semantic regularities in the language.Two models used to learn embeddings are: CBOW and Skip-Gram. The CBOW model learns the embedding by predicting the current word based on its context. The continuous skip-gram model learns by predicting the surrounding words given a current word.The second architecture is similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence.
In word2vec, a distributed representation of a word is used. Take a vector with several hundred dimensions (say 1000). Each word is represented by a distribution of weights across those elements. So instead of a one-to-one mapping between an element in the vector and a word, the representation of a word is spread across all the elements in the vector, and each element in the vector contributes to the definition of many words.
An advantage of word2vec is that the context information is not lost. Another great advantage of Word2Vec approach is that the size of the embedding vector is very small.
For text preprocessing the following steps were performed :

- Lower casing the text
- Tokenization
- Removing punctuations
- Removing stopwords

The CBOW and skip gram models were created with this preprocessed text, using `Gensim’s Word2Vec`.Initially, minimum frequency was set to 1,window size to 5 and dimensionality of word vectors to 100. The two models were tested, by finding cosine similarity between the vectors of two words. On regulating the The size of the sliding window, it was observed that has a strong effect on the resulting vector similarities. Large windows tend to produce more topical similarities [Princess- Cinderella], while smaller windows tend to produce more syntactic similarities[].

2. GloVe
3. tf-idf vector
4. Poincare
5. BERT

Refrences:

1. https://medium.com/@zafaralibagh6/simple-tutorial-on-word-embedding-and-word2vec-43d477624b6d
2. https://arxiv.org/pdf/1301.3781.pdf
3. https://soda.la.psu.edu/new-faces/new-faces-papers-2019/new-faces-rodriguez
4. https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/
5. https://towardsdatascience.com/word-embeddings-exploration-explanation-and-exploitation-with-code-in-python-5dac99d5d795
6.

LSA:

Topic Modelling is an unsupervised technique to discover abstract topics across a text corpus. Latent Semantic Analysis (LSA) is a technique for topic modelling, based on the 'bag of words' model. It attempts to find the context around the words to capture the hidden concepts or topics.

For text preprocessing the following steps were performed :

- remove the punctuations, numbers, and special characters
- remove words shorter than length=3
- Lower casing the text
- Remove stopwords

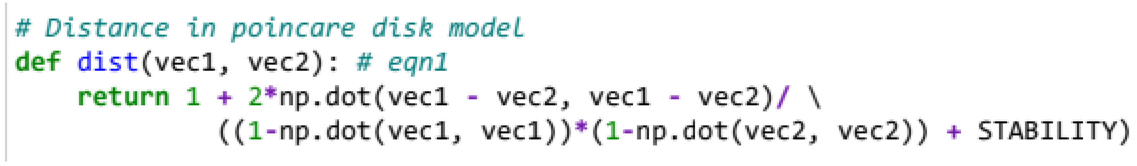
A document-term matrix was generated, having TF-IDF scores. sklearn's *TfidfVectorizer* was used for this purpose. The matrix was limited to 1500 features.
Then, the dimensions(features) of the above matrix were reduced to the number of desired topics, taken to be 30. This is done by Single Value Decomposition(SVD), and was executed using sklearn's *TruncatedSVD*. SVD gives vectors for every document and term of the text corpus, and length of each vector is the total number of topics taken, here 30. A good approximation of k can be determined by the elbow method.

GloVe

Word2vec relies only on local information of language. That is, relations of a given word, is only affected by the surrounding words, taken in the window. GloVe, on the other hand, captures both global and local statistics of a corpus, in order to come up with word vectors. It also is an unsupervised learning algorithm, and uses a neural network for training. Training is performed using co-occurrence counts of words.
GloVe is also implemented with the gensim library. The learning rate of the neural network was set to 0.03.

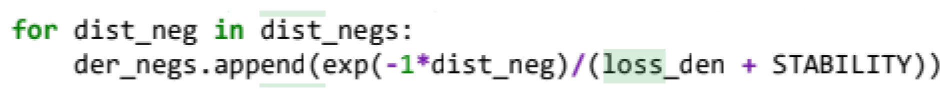
## Poincare

Words in a text may have an underlying hierarchy to them, which cannot be captured in the euclidean space.  This hierarchical information can be added to the numerical representation of a word using Poincare Embeddings. Hierarchies are often trees where the number of nodes increases exponentially by the level. Hyperbolic spaces are analogous to these continuous versions of trees. The hyperbolic space gives the ability to capture the similarity between words effectively (through their distance) and also preserves their hierarchy (through their norm).
The Poincaré ball follows this property, so we can define our vectors on this space. The model needs to give a good score for clustering vectors in the same hierarchy as well as vectors in different hierarchies spreading out on the same branch. Riemann gradient descent is used to reduce the loss function.
A dataset consisting of hyponyms was taken. Each word was added in a network, along with its level in the tree. Embeddings are initialised randomly. One node is chosen from the vocabulary and another node is chosen at random from the network.  And the distance between the nodes in the Poincare disk model is calculated, and checked. Stability = 0.00001 to ovoid overflow while dividing.



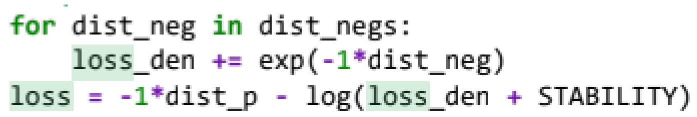
```python
# Distance in poincare disk model
def dist(vec1, vec2): # eqn1
    return 1 + 2*np.dot(vec1 - vec2, vec1 - vec2)/ \
            ((1-np.dot(vec1, vec1))*(1-np.dot(vec2, vec2)) + STABILITY)
```

In the code, if the distance is <-700, they are taken as negative relations while if distance>700, they are taken as positive relations. For all such negative distances, the loss is calculated using



```python
for dist_neg in dist_negs:
    der_negs.append(exp(-1*dist_neg)/(loss_den + STABILITY))
```

And for positive distances, it is calculated using



```
for dist_neg in dist_negs:
    loss_den += exp(-1*dist_neg)
loss = -1*dist_p - log(loss_den + STABILITY)
```

Derivatives of these losses are calculated and embeddings are updated.