COMP7630 – Web Intelligence and its Applications

Text Classification and Clustering with Scikit Learn

Valentino Santucci

(valentino.santucci@unistrapg.it)

Outline

- Text Vectorization
- Text Classification
- Text Clustering

Text Vectorization

- In order to apply common Machine Learning algorithms to texts, we need to vectorize a text

- Vectorizing a text = transform a text into a fixed-length vector of numerical features

- We already know how to vectorize a text by using Spacy & SBERT
- Classification and clustering techniques here described apply to all the vectorization methodologies

Build a data matrix *X* from a corpus using Spacy

```
In [74]: import spacy

In [75]: import numpy as np

In [76]: nlp = spacy.load('en_core_web_md')

In [77]: texts = [ 'Hong Kong is a beautiful city!',
    ...:           'Bruce Lee was from Hong Kong',
    ...:           'Hong Kong and Macau are two Chinese special administrative regions',
    ...:           'Macau has a very beatiful historical center!',
    ...:           'Hong Kong and Macau are two cities',
    ...:           'Perugia is a city as well' ]

In [78]: docs = [ nlp(text) for text in texts ]

In [79]: X = np.vstack([ doc.vector for doc in docs ])

In [80]: X.shape
Out[80]: (6, 300)

In [81]: X
Out[81]:
array([[-1.0696642 ,  0.6536115 , -2.9217427 , ...,  0.11435284,
        -2.347317  ,  3.4459343 ],
       [-3.1438649 , -3.4454544 , -0.4097681 , ..., -0.81481665,
        -0.9796772 ,  3.4029067 ],
       [-3.5210297 , -2.8748152 , -0.65785944, ..., -3.63582   ,
        -2.5489378 ,  1.9917101 ],
       [-0.38297462,  1.39113   , -2.1096926 , ..., -0.5993001 ,
        -1.8713462 ,  1.3179444 ],
       [-4.2557993 , -3.0536575 , -0.84754276, ..., -3.5127575 ,
        -2.472097  ,  1.6814859 ],
       [-2.864108  ,  2.0063782 , -3.8127003 , ..., -1.1261166 ,
        -1.373845  ,  2.8202565 ]], dtype=float32)
```

Data Matrix

A numpy matrix where each row represents a text and each column is a feature (obtained by using some methodology – word2vec vectors in this example).

This kind of data matrix is used by Scikit Learn for classification, clustering and preprocessing.

Note for Spacy word-embeddings

The vector of a Spacy container is the average of the vectors of its tokens, also stop-word tokens!!!

It may be useful to implement the average of non-stop-word tokens by our own.

… or using a SBERT Sentence Transformer

```
In [2]: from sentence_transformers import SentenceTransformer

In [3]: model = SentenceTransformer('all-MiniLM-L6-v2')

In [4]: sentences = [ 'Hong Kong is a beautiful city!',
   ...:               'Bruce Lee was from Hong Kong',
   ...:               'Hong Kong and Macau are two Chinese special administrative regions',
   ...:               'Macau has a very beaufiful historical center!',
   ...:               'Hong Kong and Macau are two cities',
   ...:               'Perugia is a city as well' ]

In [5]: X = model.encode(sentences)

In [6]: X.shape
Out[6]: (6, 384)

In [7]: X
Out[7]:
array([[ 0.11758427,  0.05357094,  0.09600548, ..., -0.03071702,
        -0.01504323,  0.05885128],
       [-0.00223547,  0.10790177, -0.0278198 , ..., -0.05692163,
        -0.00180312,  0.06139478],
       [ 0.11534194,  0.01797782,  0.06273731, ...,  0.01355424,
        -0.05856783,  0.05016575],
       [ 0.12823547,  0.04275572, -0.01268561, ..., -0.05665375,
        -0.08824164,  0.03712672],
       [ 0.1251669 ,  0.02269989,  0.02983497, ..., -0.02046551,
        -0.02853459,  0.06569158],
       [ 0.08116614, -0.00793275, -0.0166357 , ...,  0.02632581,
         0.01981751, -0.03471361]], dtype=float32)
```

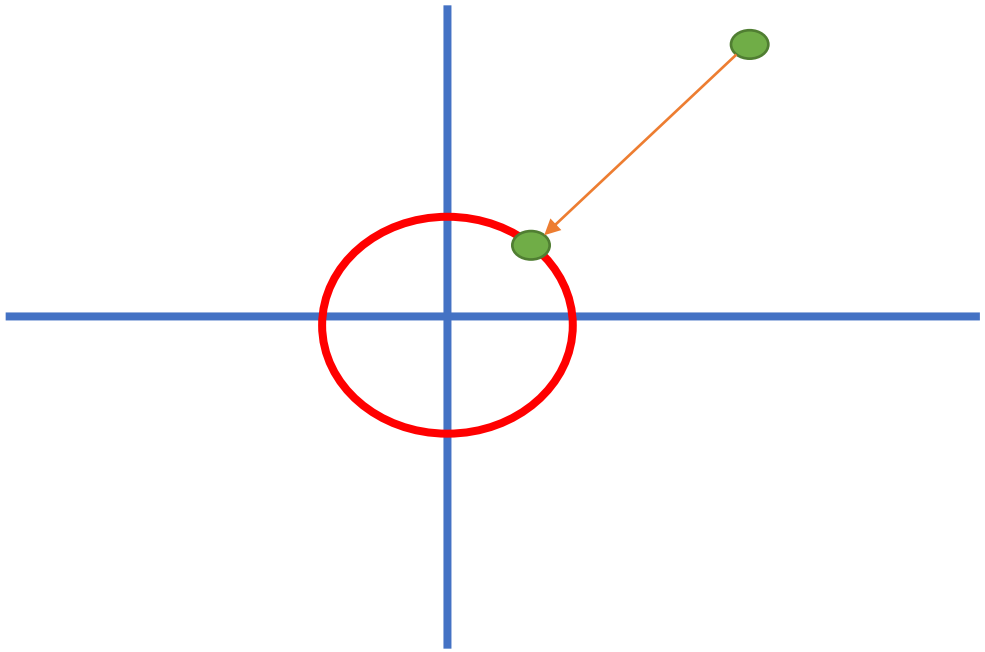
Notes for SBERT

The embeddings are no more average of words' embeddings, but SBERT models truncate long texts, so it may be helpful to first segment the sentences, then encode every sentence separately and average them (also using weighted average if some sentences are more important).

Since SBERT embeddings are vectors of entire sentences, stopwords removal is usually not necessary.

Normalization

- Normalizing a vector = dividing the vector by its (Euclidean/L2) norm
- A normalized vector has norm 1
- Geometrically, the normalization acts as a projection of the vector towards the closer point in the surface of a hyper-sphere with radius 1



Normalization is useful with vectorized texts

- Often, vectorized texts are further elaborated by means of algorithms which make use of the Euclidean distance (let think for example to K-means for clustering or K-nearest-neighbor for classification)
- However, we know that a good distance function for vectorized texts is the cosine distance (or similarity)
  - It has been observed that it works fine also when comparing texts of very different lengths
- Normalization helps because Euclidean distance between normalized vectors is (almost) the same as cosine distance between unnormalized vectors
  - The "almost" means that the distances are not really the same but they have the same ranking
- Final remark: if computational time is not an issue, the best option is always: try to perform the task both with and without normalization, then take the best result

Outline

- Text Vectorization
- Text Classification
- Text Clustering

Let install Scikit Learn library

- If you have created a `webintelligence` environment (highly suggested), then activate it with:

```
conda activate webintelligence
```

- Then:

```
pip install scikit-learn
```

Scikit Learn Classifiers

- Logistic Regression `LogisticRegression`
- Support Vector Machine `SVC`
- Random Forest `RandomForestClassifier`
- Multi-layer Perceptron `MLPClassifier`
- K-Nearest-Neighbor `KNeighborsClassifier`
- … and many others. See the following link for a complete guide to classification with Scikit Learn:

  https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

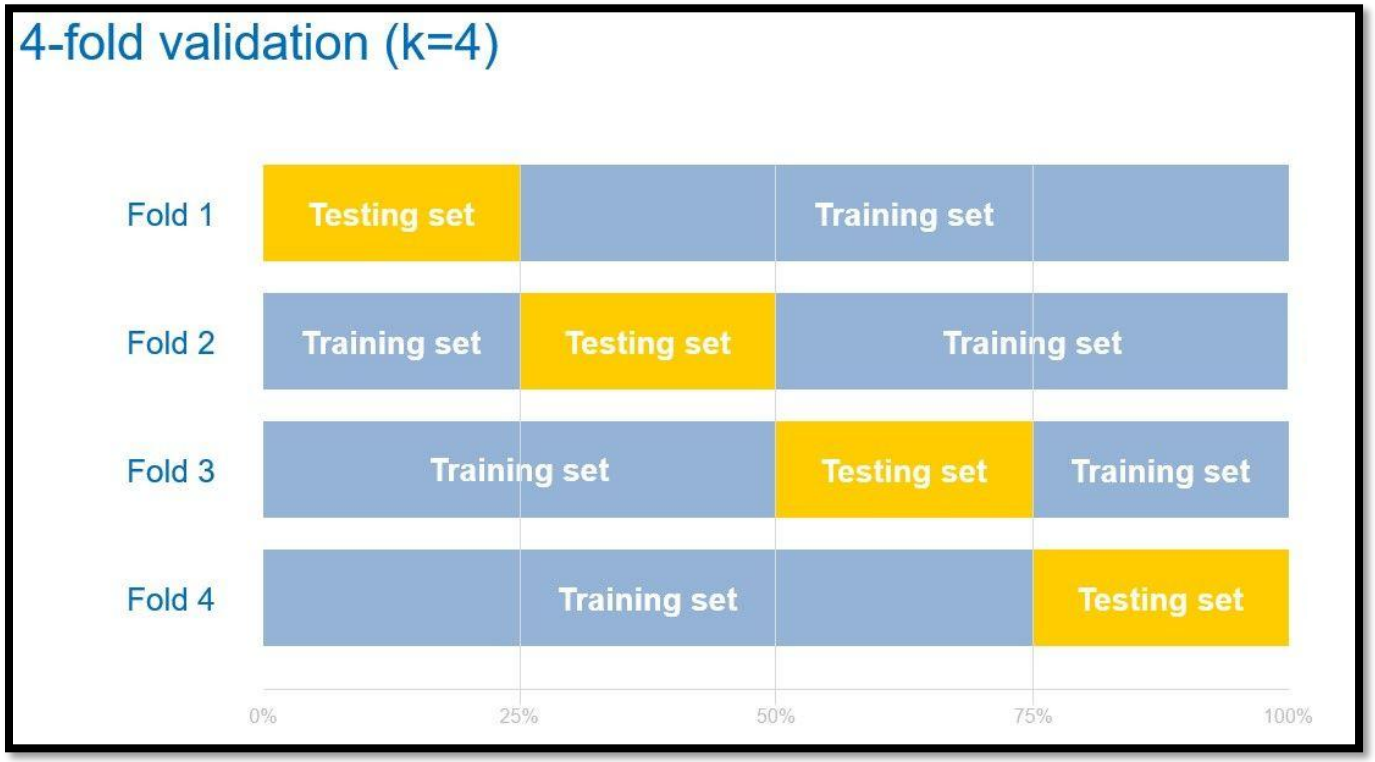
Common interface for all SKLearn Classifiers

- A constructor where to set classifier hyperparameters
- .fit(X,y) method which requires:
  - the vectorized data-matrix *X* where any row is a record/sample
  - a vector of integer values *y* such that *y*[*i*] is the label of record *X*[*i*]
- Executing .fit(X,y) means "training", so after its execution (which may require time), a model has been learned and it is now possible to make prediction on unseen records
- .predict(X_new) method returns the predicted labels of all the rows/samples in the data-matrix *X_new*
- IMPORTANT: always assess performance of a model generated by a classifier on data samples not used for training the model!!!

Training and Test Sets + Cross Validation

- Scikit Learn has useful functions and classes for:
  - splitting a dataset into training set + test set
    `train_test_split`
  - setting up cross-validation and quickly execute it
    `RepeatedStratifiedKFold`
    `cross_val_score`
  - calculate a variety of metrics
    `accuracy_score`, …

What is cross validation



Any single fold is an experiment (train + test), so there is a score (e.g. accuracy) for any fold.

"Stratified" means that the original distribution of labels is maintained (approximately) in all the training and test sets.

Hands on Classification with SKLearn

- Classifying 200 texts selected from the 20newsgroups dataset which is publicly available at http://qwone.com/~jason/20Newsgroups/

- They are messages obtained from thematic newsgroups and two themes are selected as labels: `auto` and `space`

- See the files in
  `classification_clustering_examples.zip`

Again … a very important concept for a correct experimentation in classification

- Never use test set information for training, so also in preprocessing!!!
- This principle applies also to all the preprocessing operations that work "vertically" on the dataset, i.e. that modify entire columns of the data matrix
- Examples?
  - Standardization
  - Dimensionality reduction with PCA
- In these cases you need to fit (`.fit`) the preprocessing operation only on the training set, then apply it (`.transform`) to the test/validation set
- What about normalization?
  - Normalization works on the rows and does not involve columns, so it does not pose any problem when validation or cross-validation is considered.

How to translate this into Scikit Learn? (1/2)

- As an example, let's consider Standardization as preprocessing step and Logistic Regression as classifier
- Validation case (`train_test_split`)

Split the dataset

```
In [24]: #'X' is a data matrix already loaded, while 'y' is its corresponding labels vector.

In [25]: X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, shuffle=True, stratify=y)

In [26]: scaler = StandardScaler()

In [27]: scaler.fit(X_train)
Out[27]: StandardScaler()

In [28]: X_train = scaler.transform(X_train)
```

Fit the scaler only on the training set, then transform the training set

```
In [29]: clf = LogisticRegression()

In [30]: clf.fit(X_train, y_train)
Out[30]: LogisticRegression()
```

Run the training (on the training set only)

```
In [31]: X_test = scaler.transform(X_test)

In [32]: y_pred = clf.predict(X_test)

In [33]: acc = accuracy_score(y_test, y_pred)

In [34]: acc
Out[34]: 0.825
```

Before predicting on the test set, transform the test set using the same scaler fitted before

How to translate this into Scikit Learn? (2/2)

- Cross Validation case (`cross_val_score`)

```
In [56]: from sklearn.pipeline import Pipeline

In [57]: #'X' is a data matrix already loaded, while 'y' is its corresponding labels vector.

In [58]: clf_plus_preprocessing = Pipeline( steps = [ ( 'standardization', StandardScaler()     ),
    ...:                                               ( 'classification',  LogisticRegression() )  ] )

In [59]: cv = RepeatedStratifiedKFold(n_repeats=10, n_splits=5)

In [60]: accuracies = cross_val_score(clf_plus_preprocessing, X, y, cv=cv, scoring='accuracy')

In [61]: accuracies.mean()
Out[61]: 0.8725
```

A Sklearn pipeline is a sequence of zero or more preprocessors + (optionally) one classifier as very last element

Run cross validation using the defined pipeline.
The implementation of the `Pipeline` object ensures that no data of the validation set is used for preprocessing!!!

One remark about PCA on Scikit Learn

class `sklearn.decomposition.`**PCA**(*n_components=None, *, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', n_oversamples=10, power_iteration_normalizer='auto', random_state=None*) [source]

Parameters:

n_components : *int, float or 'mle', default=None*

Number of components to keep. if n_components is not set all components are kept:

```
n_components == min(n_samples, n_features)
```

If `n_components == 'mle'` and `svd_solver == 'full'`, Minka's MLE is used to guess the dimension. Use of `n_components == 'mle'` will interpret `svd_solver == 'auto'` as `svd_solver == 'full'`.

If `0 < n_components < 1` and `svd_solver == 'full'`, select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components.

If `svd_solver == 'arpack'`, the number of components must be strictly less than the minimum of n_features and n_samples.

Hence, the None case results in:

```
n_components == min(n_samples, n_features) - 1
```

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Outline

- Text Vectorization
- Text Classification
- Text Clustering

Clustering

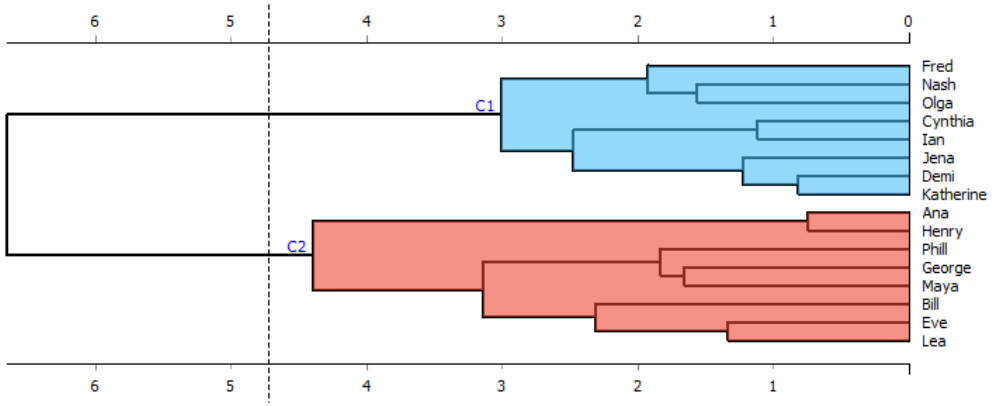
- Clustering is an Unsupervised Machine Learning task, where the goal is to group data points based on similarity, aiming to identify inherent patterns or structures within the data.
  - Input = a dataset of unlabeled instances
  - Output = a cluster ID for each instance

- Also Dimensionality Reduction (PCA or SVD) are Unsupervised Machine Learning
  - Input = a dataset of unlabeled instances
  - Output = new numerical representations for the instances

Agglomerative Hierarchical Clustering

- Clustering is usually based on some definition of distance
- The most simple way to cluster instances is:
  1. Calculate the distances between all the pairs of instances
  2. Initially set singleton clusters (each instance defines a cluster, with only itself)
  3. Merge the two closer clusters
  4. Go back to step 3 until a termination condition is satisfied

  (e.g.: pre-established number of clusters reached, some quality measure is reached, etc.)
- Pros: it is agnostic to the distance measure, it allows to plot a dendrogram
- Cons: it requires a large amount of distance computations (at least $O(n^2)$)



K-Means

- K-Means is an unsupervised machine learning algorithm used for clustering data into *k* groups (or clusters)

- K-Means requires in input:
  - a data-matrix *X* whose rows are vectorized objects that have to be partitioned
  - the number of clusters *k*
  - some hyperparameters (such as the initial guess of *k* cluster's centroids)

- K-Means returns in output:
  - a vector *y* such that *y*[*i*] contains the id of the cluster decided for object *X*[*i*]
  - a goodness measure (called *inertia*) of the clusterization performed

How K-Means works

1. The algorithm starts with an arbitrary initial setting of *k* centroids
2. Each data point is assigned to the closest centroid
3. The centroids are then recomputed as the mean of all the data points assigned to it
4. This process repeats until the centroids no longer move or a maximum number of iterations is reached

- Pros: it is more efficient than Hierarchical Aggl. Clust. (each iteration, only require *k*n* distance calculations)
- Cons: it is implicitly based on the Euclidean distance (due to centroids calculations) … but normalization of the input practically mitigates this issue

Initial Setting for the Centroids?

- The final clustering depends on the initial setting of the centroids
- In step 1, the *k* centroids are usually randomly initialized, but other methodologies can be adopted. In fact, the initial setting of the centroids is an hyperparameter of the K-Means which can also be adjusted by using some strategy.
- K-Means also compute an inertia measure for a given centroids' setting
- The final inertia returned by K-Means can be used to measure the goodness of different initial centroids' settings.
- What is the inertia of a centroid setting? Sum of squared distances of samples to their closest cluster centroid (i.e. the cluster to which the sample is assigned by K-Means).
- However, note that inertia is meaningful only for comparing clusterings with the same *k*.
  - Indeed, as *k* increases, the inertia decreases, because there are more clusters, so each cluster has fewer points, so the points are closer to their cluster centers.

Number of Clusters in K-Means

- The number of clusters (*k*) must be specified in advance
- If you do not have any extra-information which guide you in setting *k*?

- Silhouette Score measures the similarity of an observation to its own cluster compared to other clusters
- Silhouette scores of each sample can be averaged

- Run K-Means with different values of *k*, compute the average silhouette scores in each case, then select the one with the largest silhouette score.

Silhouette Score (of a sample)

$$s_i = \frac{b_i - a_i}{max(b_i, a_i)}$$

where:

- a_i is the mean intra-cluster distance of sample *i*

(i.e. the mean distance between *i* and samples assigned to its same cluster)

$$a_i = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j)$$

- b_i is the mean nearest cluster distance

(i.e. the mean distance between i and samples assigned to nearest cluster)

$$b_i = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j)$$

- *s* lies in [-1,+1] and need to be maximized

Hands on Clustering with SKLearn

- Cluster 200 texts selected from the 20newsgroups dataset which is publicly available at http://qwone.com/~jason/20Newsgroups/

- We do not use label informations for clustering!

- See the files in

  `classification_clustering_examples.zip`

References

- Manuals, tutorials and examples available in the Scikit Learn website https://scikit-learn.org/stable/