Feature Extraction

What is Feature Extraction?

One of the most important parts of data preprocessing is feature extraction, which is a process of reducing data dimensionality by modifying variables describing data such way, that created set of features (feature vector) describe data model accurately and overall in a direct way. Thanks to feature extraction data is easier to compare thus easier to analyze.

> *Feature extraction is the transformation of original data to a data set with a reduced number of variables, which contains the most discriminatory information.* ~ Transient Electromagnetic-Thermal Nondestructive Testing, 2017

Other Data Preprocessing Techniques

Besides feature extraction data preprocessing includes:

- **feature engineering** - transforming raw data (which often is very redundant) into a dataset with more informative format.
- **feature transformation** - to improve algorithm performance with normalization, scaling and dealing with missing values.
- **feature selection** - removing unnecessary features settle a subset of the original features. Unlike feature extraction which transforms attributes (e.g. creates a linear combination of them etc.)

Why we need to use it?

Clarity of data is important to algorithm as it is to humans. Redundancy doesn't allow us to focus on the important detail, machines work the same. The more concentrated the information in data is the better. It's worth emphasizing that information is something different from data. It's an abstract concept of useful value that data brings to the phenomena that we are analyzing.

Not every data is informative, and the role of every data analyst is to draw useful information from the given data. Many believe that well-performed feature extraction is the key to build an effective model.

What are the most common techniques in NLP?

Natural Language Processing requires creating tabular data from the raw text. It's done many ways but the most common are:

- **BoW** (Bag of Words/Ngrams) - represent text that describes the frequency of each word in a document. New feature vector has a column for each word in a document, and each row has a number of occurrences of this word in a sample (sentence, paragraph, etc.)
- **CBoW** (Continuous Bag of Words) - extend BoW, where the model tries to predict learned word by surrounding words disregard the order.
- **CSG** (Continuous Skip Gram) - this model uses a current word to predict surrounding words, with higher wages for closer words. CBoW is faster, but skip-gram has better results for rare words.

- **Word2vec** - is a group of related models: CBoW (Continuous Bag of Words) and Continuous skip-gram, used to produce word embeddings
- **GloVe** (Global Vectors) - is an unsupervised learning model that used to obtain dense word vectors similar to Word2Vec. The basic method of the GloVe model is to first create a huge word-context co-occurence matrix consisting of (word, context) pairs such that each element in this matrix represents how often a word occurs with the context (which is a sequence of words).
- **TF-IDF** (Term frequency–inverse document frequency) - is a numerical statistic of the importance of each word in a document. TF divides the number of occurrences of each word in a document by the total number of words in the document. IDF is to downscale weights for words that occur in many documents in the corpus and are less informative than those that occur only in a smaller part of the corpus.
- **Autoencoders** - is an artificial neural network that encodes the input in an unsupervised way that output can be preserved. This algorithm basically learns to ignore the noise in input data. Its hidden form can be used as an encoding of the input with preserved all information that the second part of the network uses to recreate input data.

Python Implementation Essentials

- **BOW** using *CountVectorization* - sklearn

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = [
    'This is the first document.',
    'This document is the second document.',
    'And this is the third one.',
    'Is this the first document?',
    ]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
print(vectorizer.get_feature_names())
>   ['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third',
'this']
print(X.toarray())
>   [[0 1 1 1 0 0 1 0 1]
     [0 2 0 1 0 1 1 0 1]
     [1 0 0 1 1 0 1 1 1]
     [0 1 1 1 0 0 1 0 1]]
```

- **TF-IDF** using *TfidfTransformer* - sklearn

```
from sklearn.feature_extraction.text import TfidfTransformer
tf_transformer = TfidfTransformer(use_idf=False).fit(X_train_counts)
X_train_tf = tf_transformer.transform(X_train_counts)
X_train_tf.shape
> (2257, 35788)
```

- **CBOW** and **CSG** using **Word2Vec* - genism

```
from gensim.models import Word2Vec
# CBOW model
cbow = Word2Vec(data, min_count = 1, size = 100, window = 5)
# Skip Gram model
sg = Word2Vec(data, min_count = 1, size = 100, window = 5, sg = 1)
```

- **Glove** using *EmbeddingTransformer* - zeugma

```
from zeugma.embeddings import EmbeddingTransformer
glove = EmbeddingTransformer('glove')
X_train = glove.transform(corpus)
```

- **Autoencoder** using *layers* and *models* - keras

```
from keras.layers import Dense, Flatten, Reshape, Input, InputLayer
from keras.models import Sequential, Model

def build_autoencoder(img_shape, code_size):
    # The encoder
    encoder = Sequential()
    encoder.add(InputLayer(in_shape))
    encoder.add(Flatten())
    encoder.add(Dense(code_size))

    # The decoder
    decoder = Sequential()
    decoder.add(InputLayer((code_size,)))
    decoder.add(Dense(np.prod(in_shape)))
    decoder.add(Reshape(in_shape))

    return encoder, decoder
```