Preprocessing

Machine Learning, A.Y. 2022/23, Padova

Fabio Aiolli

November 16th, 2022

Supervised learning pipeline

- Analysis of the problem
- Collection, analysis and cleaning of data
- Preprocessing and missing values
- Study of correlations among variables
- Feature Selection/Weighting/Learning
- Choice of the predictor and Model Selection
- Test

Objects

- Vectors: e.g. blood pressure, heart rate at rest, height and weight of a person, useful to a insurance company to determine her/his life expectancy
- Strings: e.g. words in a textual document for the NER task, or the structure of DNA
- Sets and Bags: e.g. the set of terms in a document, or maybe even their frequency?
- Tensors: e.g. Images (2D) and Video (3D)
- Trees and graphs: e.g. the structure of a XML document, or a molecule in chemistry
- ...
- Compound structures: e.g. a web page can contain images, text, videos, tables, etc.

Objects

- Categorical or symbolic features
  - Nominals [no order]
    e.g. for an auto: country of origin, brand, color, type, etc.
  - Ordinals [do not preserve distances]
    e.g. military ranks of the army: soldier, corporal, sergeant, marshal, lieutenant, captain
- Quantitative and numeric features
  - Intervals [Enumerables]
    e.g. level of appreciation of a product from 0 to 10
  - Ratio [Reals]
    e.g. the weight of a person

Encoding categorical or simbolic variables

OneHot Encoding

Categorical variables can be represented in a vector with as many components as the number of possible values for the variable.

Ex. Possible values of the variables:

- Brand: `Fiat [c0], Toyota [c1], Ford[c2]`
- Color: `White [c3], Black [c4], Red [c5]`
- Type: `Subcompact [c6], Sports [c7]`

```
(Toyota, Red, Subcompact) -> [0,1,0,0,0,1,1,0]
```

OneHot encoding in sklearn

OneHot encoding can be obtained easily in sklearn using the methods fit and transform of the class OneHotEncoder in the following way:

```
>>> enc = preprocessing.OneHotEncoder()
>>> enc.fit([[0,0,0], [1,1,1], [2,2,1]])
OneHotEncoder(categorical_features='all', dtype=<...
'numpy.float64'>,handle_unknown='error', n_values='auto',
parse=True)
>>> enc.transform([[1,2,0]]).toarray()
array([[ 0., 1., 0., 0., 0., 1., 1., 0.]])
```

Encoding of continuous variables

In this case, it is more difficult to find a good mapping

The features are typically transformed to obtain values that are "comparable" with other features

- Standardization (centering and/or variance scaling)
- Scaling in a range
- Normalization

Let $\hat{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$ and $\sigma_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \hat{x}_j)^2}$

- Centering: $c(x_{ij}) = x_{ij} - \hat{x}_j$
- Standardization: $s(x_{ij}) = \frac{c(x_{ij})}{\sigma_j}$
- Scaling to a range: $h(x_{ij}) = \frac{x_{ij} - x_{min,j}}{x_{max,j} - x_{min,j}}$
- Normalization: $g(\mathbf{x}) = \frac{\mathbf{x}}{||\mathbf{x}||}$

Preprocessing of continuous variables in sklearn: StandardScaler

```
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> scaler
StandardScaler(copy=True, with_mean=True, with_std=True)
>>> scaler.mean_
array([ 1.  ..., 0.  ..., 0.33...])
>>> scaler.scale_
array([ 0.81..., 0.81..., 1.24...])
>>> scaler.transform(X_train)
array([[ 0.  ..., -1.22..., 1.33...],
 [ 1.22..., 0.  ..., -0.26...],
 [-1.22..., 1.22..., -1.06...]])
```

Preprocessing of continuous variables in sklearn: MinMaxScaler

```
>>> min_max_scaler = preprocessing.MinMaxScaler()
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)
>>> X_train_minmax
array([[ 0.5 , 0.  , 1.  ],
 [ 1.  , 0.5 , 0.33333333],
 [ 0.  , 1.  , 0.  ]])
array([ 0.81..., 0.81..., 1.24...])
>>> scaler.transform(X_train)
array([[ 0.  ..., -1.22..., 1.33...],
 [ 1.22..., 0.  ..., -0.26...],
 [-1.22..., 1.22..., -1.06...]])
```

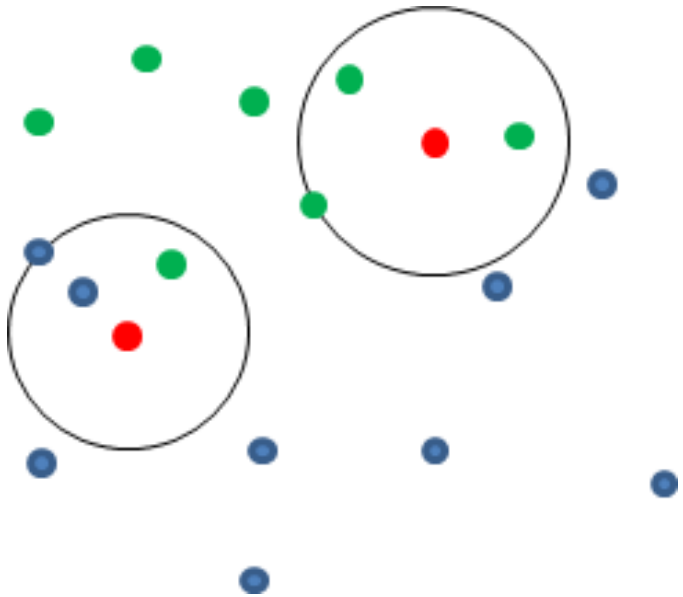
Preprocessing of continuous variables in sklearn: Normalizer

The dot product (and distances) between vectors are influenced by their norm, as well as the angle between the vectors: $\langle \mathbf{x}, \mathbf{z} \rangle = ||\mathbf{x}|| \cdot ||\mathbf{z}|| \cdot \cos(\theta)$. When we want to consider ONLY the angle formed between the vectors we can resort to normalization.

```
>>> normalizer = preprocessing.Normalizer().fit()
>>> normalizer
Normalizer(copy=True, norm='l2')
>>> normalizer.transform(X)
array([[ 0.40..., -0.40..., 0.81...],
 [ 1.  ..., 0.  ..., 0.  ...],
 [ 0.  ..., 0.70..., -0.70...]])
```

K-Nearest Neighbors

K-Nearest Neighbors is a simple classification algorithm where a test example is classified with the majority class of its k-neighbors in the training set

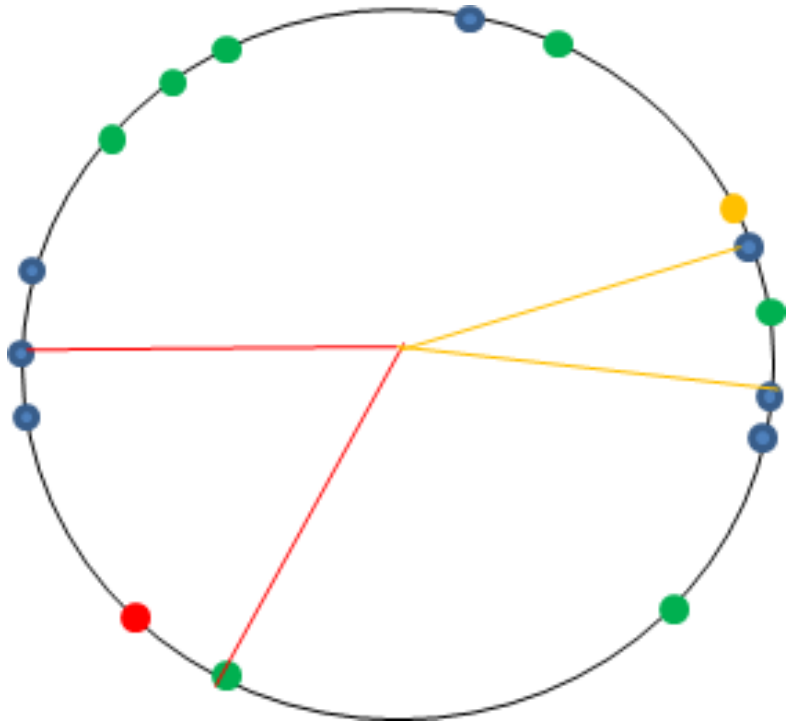


Example 3-NN in 2D using the Euclidean distance

K-Nearest Neighbors with normalization of instances

K-Nearest Neighbors when the examples are all in a ball of unit radius.
The distance becomes equivalent to the dot product.
In fact,

$$||\mathbf{x} - \mathbf{z}||^2 = ||\mathbf{x}||^2 + ||\mathbf{z}||^2 - 2\langle \mathbf{x}, \mathbf{z} \rangle = \text{const} - 2\langle \mathbf{x}, \mathbf{z} \rangle$$



Example 3-NN in 2D with unit norm of the instances

Preprocessing of continuous variables in `sklearn`: Other options

- `Binarizer` for the binarization of the features (based on a threshold)
- `KBinDiscretizer` for the discretization of the features into k bins
- `Imputer` for the imputation of missing values (based on mean, median, most frequent value in the row or in the column)
- `FunctionTransformer` for the customized definition of preprocessing

Feature selection and feature extraction

- Selection: Reduction of the dimensionality of the features obtained by removing the irrelevant or redundant features. Maintains the interpretability of the generated model.
- Extraction: Reduction of the dimensionality of the features obtained by combining the original features (e.g. PCA). Generally, the interpretability of the generated model is lost.

Feature Selection

Top reasons to use feature selection are:

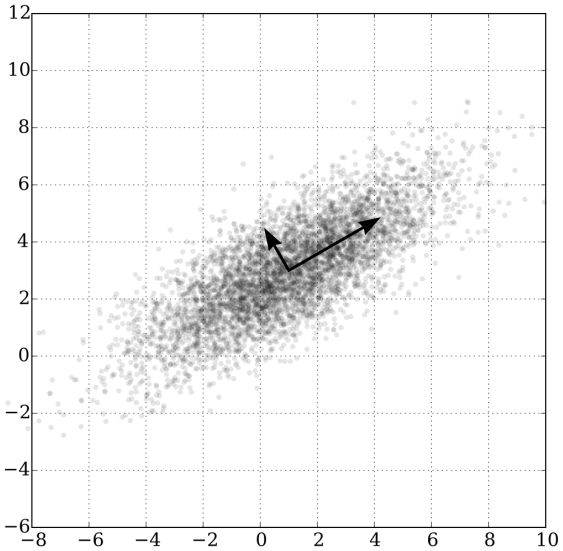
- It enables the machine learning algorithm to train faster.
- It reduces the complexity of a model and makes it easier to interpret.
- It improves the accuracy of a model if the right subset is chosen.
- It reduces overfitting.

Feature selection methods:

- Filter methods: They use an efficient scoring function (e.g. Mutual Information, Chi squared, Information Gain) that determines the usefulness of a given set of features (independent of the predictor);
- Wrapper methods: The predictor is evaluated on a hold-out sample using sub-sets of different features (e.g. RFE for SVMs);
- Embedded methods: The selection of features occurs in conjunction with the creation of the model, for example by modifying the objective function to be optimized (e.g. regularization, LASSO).

Feature extraction

Principal Component Analysis (PCA): converts a set of instances with possibly related features into corresponding values on another set of linearly unrelated features (principal components).



Neural Networks can also be seen as a particular way to perform feature extraction on their hidden layers.

Recap

- Encoding of categorical variables
- Encoding of continuous variables
- Preprocessing
- Feature Selection and Extraction

Activities:

- Think about how range-type variables might be represented. Is it reasonable to represent them as any numeric variables or is it better to have ad-hoc encoding?
- Try to preprocess the TITANIC dataset available on kaggle. Fixed a predictive model (for example SVM, k-NN, etc.) verify if there are significant variations in the prediction accuracy.
- Gain experience using the feature selection methods found in sklearn.