Introduction to Python for Social Science

Lecture 5 - Machine Learning I

Musashi Harukawa, DPIR

5th Week Hilary 2020

Overview

Last Week

- Data Visualisation

This Week

For the next two weeks, we will be discussing *machine learning* and an accompanying library, `scikit-learn`.

This week, we focus on *unsupervised learning*.

Background

What is Machine Learning?

Machine learning (ML) could be described as a field of study located between algorithms, statistics, econometrics, and numerical computation. In general, machine learning methods are concerned with algorithmic solutions to numerical and data-based problems, and their efficient implementation in modern computing.

Some ML Terminology

ML has its own terminology, separate to the fields of econometrics, statistics and computer science. Here are some key terms:

- *Features*: Input variables, the column names of the input data, or X.
- *Labels*: Output variable, the column you are trying to predict, or Y.
- *Supervised Learning*: Methods where the within-sample labels are known.
- *Unsupervised Learning*: Methods where the within-sample labels are not known.

Unsupervised Learning: Only X, no Y

In other words, *unsupervised learning* methods start with only X, and discover some latent pattern within X. They are therefore primarily *descriptive*, as they do not tell us the relationship between X and some known quantity y.

In this course, we look at two general cases of unsupervised learning:

- Clustering
- Dimensionality Reduction

Clustering

What is Clustering?

Clustering is the process of dividing observations in data into *groups*.

Moreover, it is understood that:

- Members of the same cluster are similar to each other.
- Members in different clusters are different from each other.

Many algorithms differ on how to define this *metric of similarity*.

Why Cluster?

There are two main situations in which social science researchers may want to use clustering algorithms:

- *Discovery*: The researcher has an *a priori* expectation that particular "natural" groups exist within the data, and wants to group their data accordingly.
  - Measuring intra-party unity/coherence using voting data.
  - Recognising conflict hotspots from geospatial data.
- *Exploration*: The researcher has no expectation of what groups may exist, but is seeking to better understand the information contained in their data.
  - Exploring voter surveys to discover "types".

In Notation

Given:

- j-dimensional feature space
- Data matrix X_{ij} of i observations of j features

Clustering algorithms:

- Assign each observation x_i to a cluster $k \in K$
- Where $K \leq j$

Parameters

- Parameters are values you pass to the clustering algorithm that determine its behaviour.
- Some clustering algorithms require you to specify K, the number of clusters you want returned, whereas others require you to specify a different metric such as minimum number of observations per cluster.
- As far as I am aware, *there is no completely non-parametric clustering algorithm*, as even those not requiring any user input (e.g. Gaussian Mixture Models) make parametric assumptions about the distribution of the clusters within the data.

Algorithms

There are many algorithms for clustering. We focus on the first one in detail:

- `k-means` Clustering
- Agglomerative (Hierarchical) Clustering
- Density-based Clustering (DBSCAN)
- Affinity Propogation

k-means

The k-means algorithm separates X into a specified number of K groups with equal variance, minimizing the within-cluster-sum-of-squares (called *inertia*).

It has the following advantages:

- Straightforward algorithm (to be explained) with intuitive clusterings.
- Highly scalable: works well with large N.

And the following disadvantages:

- The less "circular" the clusters are, the less well it performs; elongated or irregularly-shaped clusters are not usually found.
- High-dimensional or mixed-type data often require regularisation for efficient and effective functioning.

k-means: The Problem

Given N samples X, find K disjoint clusters C, such that the sum of squared distances of each point x_i within cluster k to the mean μ_k is minimised.

In other words:

$$\sum_{i=0}^{n} \min_{\mu_k \in C} ((x_i - \mu_k)^2)$$

Another way to think of this problem:

Given some data, find the location for K centroids whereby the distance from each observation to its closest centroid is minimised.

k-means: The Algorithm

The algorithm to solve this problem (known as Lloyd's Algorithm) has three steps. The first step is done once, and then steps 2 and 3 are repeated:

1. Choose K points $\mu_{k \in K}$, called "centroids".
2. Assign each point $x_i \in X_{ij}$ to a cluster by finding the nearest centroid.
3. Re-calculate the centroids by taking the mean of each cluster.

At step 3, the distance each centroid has moved is calculated, and once this distance goes below some threshold for all clusters, the algorithm is understood to have converged.

Some variations and issues:

- *Multimodality*: For a given dataset, there are many possible clusterings that can result in the centroids no longer moving significantly at each step.
  - The cluster assignments are therefore a result of the random initialisation, and they themselves random.
- `k-means++`: This variant of the algorithm, implemented by default in `scikit-learn`, chooses random starting centroids that are far from each other, resulting in more efficient computation (faster convergence) and more consistent clusterings.

Other Clustering Algorithms

- `k-means` is an instance of a Euclidean centroid-based algorithm. We can cluster based on different distance metrics, or take a different approach to assigning and separating values into clusters.
  - Other distance metrics include cosine distance, L1 (Cityblock) distance, or graph distance.
  - Other algorithms for separation/assignment include hierarchical splitting of the sample.

Dimensionality Reduction

What is Dimensionality Reduction?

Whereas clustering is used to find "groups" within the data, dimensionality reduction methods aim to provide a lower-dimensional representation of the same data.

This "reduced form" of the data can either contain a subset, or mix, of the original variables.

Dimensionality reduction is almost always *lossy*, that is, the reduced representation of the data contains less information than original dataset.

Use Cases

- *Feature Selection*: The researcher may want to account for/rule out a great number of variables and interactions in their model, and uses a method such as LASSO/EN to show which variables "drop out" of their model.
- *Latent Variable Discovery*: The researcher may have a large, especially unstructured dataset without strong priors over the individual variables, and wants to find underlying dimensions that drive the majority of variation in their dataset.
- *Computational Efficiency*: The dataset is too large to be feasibly analysed with existing resources, so a reduced representation of the relevant information is obtained for use in analysis.

In Notation

Given:

- j-dimensional feature space
- Data matrix X_{ij} of i observations of j features

Dimensionality Reduction Methods:

- Provide a transformation of $X_{ij} \rightarrow X'_{ik}$, where $k \leq j$.
  - *Feature Selection*: $k \subset j$
  - *Latent Variable Discovery*: $k \not\subset j$

A Short List of Algorithms

- Principal Components Analysis (PCA)
- t-SNE/UMAP
- Topic Modelling: Latent Dirichlet Allocations, Structural Topic Model, etc.

Principal Components Analysis

Given an i-length dataset of j variables, PCA returns a $i \times K$ transformed matrix of K principal components indexed by i.

Each principal component $k = \{1, 2, ..., K \leq j\}$ are orthogonal vectors that successively capture the most variance within X_{ij}.

Thus:

- The first principal component ($k = 1$) maps the observations $i = \{1, ..., N\}$ onto the axis that contains the greatest degree of variance in X_{ij}.
- The second principal component ($k = 2$) maps the observations onto an axis that is perpendicular to the first principal component and captures the highest degree of remaining variance, once the axis captured by the first PC has been accounted for.
- And so on for the third, fourth, etc.

Interpreting Principal Components

Interpreting principal components can be complicated when the researcher has strong priors over the link betweeen the variables in the original dataset and the outcome (if there is one).

As mentioned, the first principal component contains the location observations along the latent dimension along which observations in X_{ij} vary the most. In a UK survey dataset, we could imagine that this dimension might be an aggregate of geography and class (or maybe not!)

In order to interpret principal components, one can look at the feature-component weights in the $j \times k$ matrix, which shows the "prevalence" of each of the individual variables of X_{ij} in each principal component k.

Combining PCA and k-means

PCA and k-means can be used in conjunction; the data is reduced using PCA, and then clustered with k-means. There are a number of reasons/advantages to this:

- *Visualisability*: Plotting the results of k-means in high-dimensional data is difficult. Note that t-SNE and UMAP are alternative, and arguably better methods for this.
- *Clustering over Latent Variables*: Where we can interpret the principal components, clustering over them means that we are clustering over normalised axes that contribute variance to the data.

A Short Warning

Finally, a short warning on the appropriate use of unsupervised models for social sciences:

- While these models excel at discovering latent patterns within the data, our ability to interpret what those patterns indicate in "real terms" is much weaker.
- It is easy to fall into the trap of assigning a label to a pattern, and then letting the label do all of the heavy theoretical lifting.
- The best remedy is a clear understanding of how the algorithm works: what kinds of distances does it measure, what do these distances mean, etc.

Implementation

`scikit-learn`

`scikit-learn` is a Python library containing many of machine learning algorithms that you might be interested in using. Its strengths include:

- *Compatibility*: it's designed to be compatible with many of the standard data science libraries, e.g. `numpy` and `matplotlib`.
- *Consistent Interface*: most `scikit-learn` models use the same syntax, meaning if you learn to use one model, then you learn to use almost all!
- *Diagnostic and Validation Tools*: in addition to algorithms and models, `scikit-learn` provides a huge array of tools for evaluating your model.

The Steps to Machine Learning

Once you have your data, machine learning consists of just three steps!

1. Data Pre-Processing
2. Model Fit
3. Model Evaluation

Pre-Processing

`scikit-learn` models take `numpy` arrays as inputs. These arrays:

- Must be numeric
- Must not contain NA values
- Optional: Normalization

One-Hot Encoding

In order to convert all categorical variables into numeric ones, we do what is sometimes known as a one-hot encoding. This is also known as using dummy variables.

Given a column containing a categorical variable with n possible values, we can represent it as n columns, each corresponding to one of the n possible values. Each row, then, contains a single 1 for the cell corresponding to the value in the original series, and 0 for the rest.

One-Hot Encoding

$$\begin{bmatrix} \textbf{animal} \\ \textit{dog} \\ \textit{cat} \\ \textit{fox} \end{bmatrix} \rightarrow \begin{bmatrix} \textbf{cat} & \textbf{dog} & \textbf{fox} \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Accounting for NAs

Here are three categories for dealing with NA values:

1. Removal (`pd.DataFrame.dropna()`)
2. Treating NA in a categorical response as its own answer
3. Imputation

Normalization

Normalizing data consists of subtracting the mean and dividing by the variance. This ensures that all features are on the same "scale":

$$f(x) = \frac{(x - \mu_x)}{s_x}$$

Conversion from `pandas` to `numpy`

`pandas` dataframes have a convenient method for converting to a numpy ndarray:

```
pd.DataFrame.values
```

Model Fit

Most `scikit-learn` models follow the same pattern:

1. Import the model
2. Instantiate the model with parameters
3. Fit the model to data

Steps 2 and 3 can sometimes be combined

Model Output

Once you have a fitted model object, the method for accessing the output depends on the model in question.

- For `k-means`, it is the `.labels_` method.
- For `PCA`, you need to apply the `fit_transform(X)` method to get a reduced version of the input data.

Model Evaluation

For next week!

Readings

- Clustering in General:
  - `scikit-learn` Documentation
  - Survey of Techniques
- Dimensionality Reduction:
  - PCA Visually Explained
  - UMAP Visually Explained