Feature Learning

MInDS @ Mines

Feature learning is the process of determining the best representation for our data for a given task. Feature learning can be difficult as the dimensionality of the input data increases. There are two main types of feature learning methods; feature selection and feature extraction. We will cover the three main types of feature selection methods; filter, wrapper and embedded methods, as well the general idea around feature extraction with an example method of principal component analysis.

Feature Learning

It is often difficult to decide on the features we wish to use to represent data. Today, we often have a significant amount of data and must narrow down our choices of the data in order to efficiently compute our algorithms. Feature learning is the area within machine learning that focuses on using a variety of methods to determine the best representation of the data for a given task. Depending on the task, the "best" representation might mean different things.

From a feature learning perspective, we may want to create as many combinations of features to obtain as much potentially relevant information as we can. We may want to determine the most informative features in our data. We may want to create new features that reduce the information redundancy in our data. We may want to create new features that allow us to easily separate different types/classes of samples from each other. There are many approaches.

There are two main approaches to feature learning and we'll go over them in the following sections. Feature selection is the subset of feature learning that utilizes the available features and selects the subset of them that are best for the task at hand. Feature extraction is the subset of feature learning that generates or creates new features from existing features that are best for the task at hand.

Curse of Dimensionality

One key area that we often utilize feature learning in is when we have a large ratio between features and data points. We usually refer to the number of features in our data as d and the number of samples as n. As d increases, we need a significantly higher value for n in order to fully utilize our data. d is also referred to as the dimensionality of the data and a higher value of d results in *high dimensionality*. We may think that increasing dimensionality, or collecting more information about our underlying data, is a good thing but in fact, it can create a lot of problems.

The problems that arise from the increase in dimensionality of data are

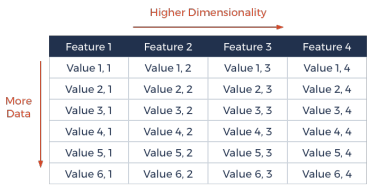

Figure 1: A visual explanation of dimensionality increase versus data increase.

called the *curse of dimensionality*. Let's examine an example where each feature in our data is a boolean value, and we want to make sure that we cover every possible combination in our data. If we have one feature, we only have two possible values. With 2 features, we have 4 possible values. 3 features is 8 values, 4 features is 16 values, and d features is 2^d values. As we add just one more boolean feature, we immediately double the number of data points needed to cover all possible values. This becomes significantly more problematic when we have features with many discrete values and worse when we have continuous values. Now if we have a sufficient number of data points that cover the possible variety in values for the features we are collecting, what may be an issue is the computational efficiency of our models. In both cases we would want to lower the dimensionality of our data.

Feature Selection

Feature selection is a feature learning approach to select a subset of the features from our data to use for a particular task.

The feature selection process can be thought of as following a flow of 4 steps:

1. Start with all features
2. Select a subset of the features
3. Train a model using the subset
4. Evaluate the performance of trained model

Feature selection methods fall under three categories; filter, wrapper, and embedded methods, that affect different stages in the feature selection process and have a variety of pros and cons.

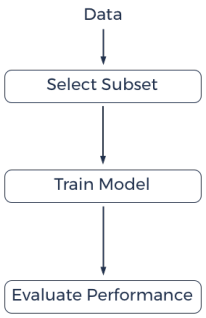

Figure 2: The feature selection process.

Filter Methods

Filter methods directly select a subset from the features using a metric that they filter by.

An example filter method would be using the mutual information score between features to determine the redundancy in the data. With the goal of reducing dimensionality by minimizing redundancy we can then remove features based on the highest amount of redundancy calculated by the mutual information. Mutual information between two vectors of discrete values, **x** and **y** is,

$$MI(\mathbf{x}, \mathbf{y}) = \sum_{x_i \in \mathbf{x}} \sum_{y_j \in \mathbf{y}} p(x_i, y_i) \log \left(\frac{p(x_i, y_j)}{p(x_i) \times p(y_j)} \right). \quad (1)$$

Mutual information between two vectors of continuous values, **x** and **y** is,

$$MI(\mathbf{x}, \mathbf{y}) = \int_{\mathbf{x}} \int_{\mathbf{y}} p(x_i, y_i) \log \left(\frac{p(x_i, y_j)}{p(x_i) \times p(y_j)} \right). \quad (2)$$

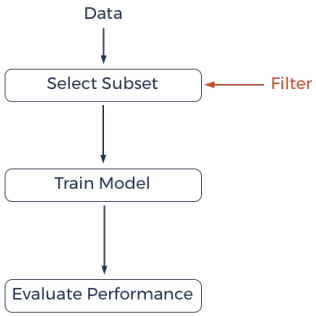

Figure 3: The feature selection process with filter methods.
Filter methods are commonly used in data preprocessing.

Filter methods are a good choice for feature selection due to their low computation cost and being robust to overfitting. They usually provide a good value for computation time and effort but we can achieve significantly better results with other methods. Filter methods are also greedy which means that they may not get the best possible results.

Wrapper Methods

Wrapper methods will loop through the subset selection and model training stages.

Based on the trained model's parameters the wrapper methods will select a new subset. An example wrapper method is recursive feature elimination (RFE). RFE utilizes models that provide a coefficient for each feature that represents its applicability or use in determining the target of the model.

Algorithm 1: Recursive Feature Elimination

Input: Features from data
Parameter: r features to select from the data
Parameter: k features to remove per iteration
Output: Subset of r features from the data

```
Train a model on all features and obtain feature coefficients
while selected features count > r do
    Remove up to k features with the lowest absolute feature
     coefficients
    Train a model on the new subset of features
end
```

Since wrapper methods utilize information learned in model training, they will usually perform better than filter methods. They are also usually greedy methods, will require more computation time than filter methods and are prone to overfitting.

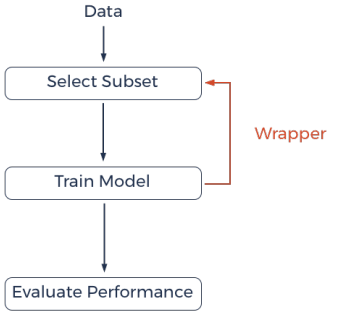

Figure 4: The feature selection process with wrapper methods.

Embedded Methods

Embedded methods will loop through the subset selection, model training and evaluation stages.

Based on the trained model parameters and performance, the embedded method will select a new subset. Examples of embedded feature selection methods are any of the regularization based regression methods we discussed previously.

A regularization based method is defined as one with an objective function as,

$$\min_{\mathbf{w}} f(\mathbf{x}, \mathbf{w}) + g(\mathbf{w}), \tag{3}$$

where f is a core objective function with learned parameters $\mathbf{w}$ and g is a regularization function applied to the learned parameters. The idea here is that we have a portion of the objective that focuses on solving the core error

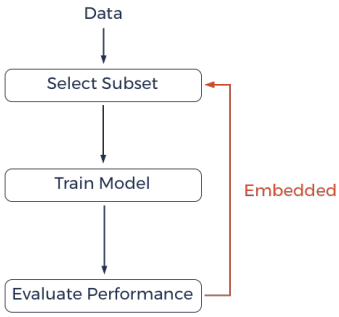

Figure 5: The feature selection process with embedded methods.

minimization and another portion that regularizes the learned parameters. Regularizing will induce sparsity on the learned coefficients of a function, effectively selecting which features to utilize and which features to ignore.

An example regularization based method that we covered was LASSO which uses the ℓ_1 norm resulting in an objective as,

$$\min_{\mathsf{w}} ||\mathsf{y}^T - \mathsf{w}^T\mathsf{X}||_2^2 + \alpha||\mathsf{w}||_1. \tag{4}$$

Here, the first portion of the objective minimizes the distance to the target whereas the second portion controls the values of the learned parameters, inducing some sparsity.

Embedded methods have the benefits of filter and wrapper methods with the addition of also incorporating the performance of the task at hand. Incorporating all these portions will usually result in much better results. However, this comes at a cost of more computation time.

Feature Extraction

Feature extraction is a different approach to feature learning where we generate new features from the original ones for the given task. We've created some basic feature transformations already such as with polynomial regression where we add more features to our data by raising the values in the data to the various powers. This allowed us to use just one feature but have multiple values per sample.

Polynomial or other simple feature transformations can be useful but we can also create much more effective data representations using other methods. Several feature extraction methods transform the original feature space into a new smaller feature space with specific properties that achieve a particular goal. The goals may include maximizing explained variance with less dimensions, maximizing information, or maximizing discriminant ability.

One example feature extraction method that is commonly used is principal component analysis (PCA). PCA transforms the original data into a new space with orthogonal dimensions that maximize the covariance between the new features called *principal components*. The objective for PCA is,

$$\begin{aligned} &\min_{\mathsf{U},\mathsf{V}} ||\mathsf{X} - \mathsf{U}\mathsf{V}^T||_F^2, \\ &s.t.\ \mathsf{U}^T\mathsf{U} = \mathsf{I}. \end{aligned} \tag{5}$$

Here, U is a $d \times r$ matrix representing the principal components or the direction of the transformed dimensions. V is an $n \times r$ matrix providing the new representation for each of the n samples in the r dimensional principal component space. The objective for PCA minimizes the reconstruction error which is the error incurred when moving from the principal component space back into the original space. Being able to reconstruct the matrix means that if we were to select a new point in the principal component space, we would be able to determine its place in the original space.

Victor Powell and Lewis Lehe created a great demo of PCA in action that explains it visually at `http://setosa.io/ev/principal-component-analysis/`.