**Data Preprocessing**

1. Dealing with missing data

It is crucial that we take care of those missing values before we proceed with further analyses.

1. Eliminating samples or features with missing values

df.dropna()

**Pro:** convenient

**Con:** end up removing too many samples, which will make a reliable analysis impossible;losing valuable information that our classifier needs to discriminate between classes.

1. Imputing missing values

**Mean imputation:** simply replace the missing value by the mean value of the entire feature column.

Code: Imputer

Github

1. dealing categorical data
2. mapping ordinal features

We need to convert the categorical string values into integers.

Code: map,use size\_mapping and inv\_size\_mapping

1. Encoding class labels

Many machine learning libraries require that class labels are encoded as integer values.

1. Performing one-hot encoding on nominal features

**Onehot encoding**: create a new dummy feature for each unique in the nominal feature column,Here, we would covert the color feature into three new features:blue,green and red.

**OneHotEncoder**

1. Partitioning a dataset in training and test sets
2. A convenient way to randomly partition this dataset into a separate *test* and *training* dataset is to use the train\_test\_split function from scikit-learn's cross\_validation submodule
3. Bring features onto the same scale

Decision trees and random forests are one of the very few machine learning algorithms where we don’t need to worry about feature scaling. However, the majority of machine learning and optimization algorithms behave much better if features are on the same scale.

There are two common approaches to bringing different features onto the same scale:

1. Normalization: refers to the rescaling of the features to a range of [0,1], which is a special case of min-max scaling.



**Sklearn: MinMaxScaler**

1. Standardization:

Center the feature columns at mean 0 with standard deviation 1 so that the feature columns take the form of a normal distribution, which makes it easier to learn the weights. Furthermore, standardization maintains useful information about outliers and makes the algorithm less sensitive to them in contrast to min-max scaling.



**Sklearn: StandardScalar**

1. Selecting meaningful features

Reason for overfitting: model is too complex for the given training data

Common solutions to reduce the generalization error:

1. Collect more training data (often not applicable)
2. Introduce a penalty for complexity via regularization
3. Choose a simpler model with fewer parameters
4. Reduce the dimensionality of the data

**Spare solutions with L1 regularization**

L2 regularization:

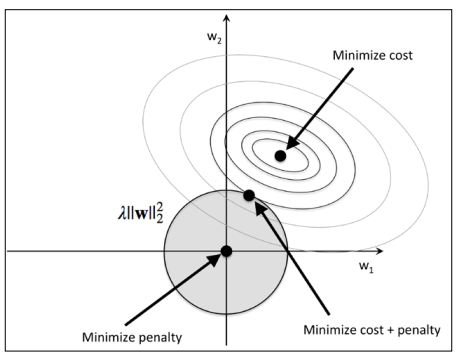


L1 regularization:



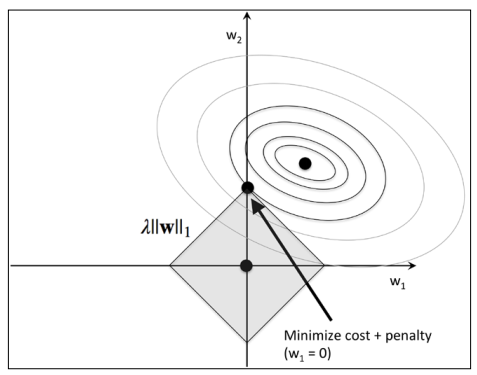
In constrast to L2 regularization , L1 regularization yields sparse feature vectors; most feature weights will be zero. Sparsity can be useful in practice if we have a high-dimensional dataset with many features that are irrelevant, especially cases where we have more irrelevant dimensions than samples. In this sense, L1 regularization can be understood as a technique for feature selection.

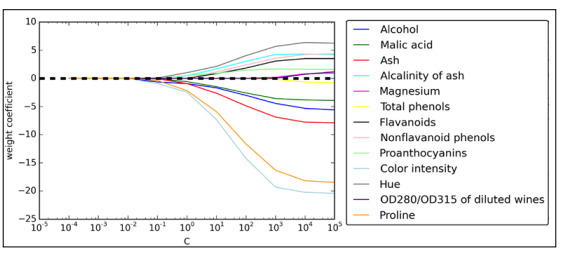
Following figure illustrate how parameter lamda of L2 penalty term shrink the weights towards zero and decrese the dependence of our model on the training data.



The quadratic L2 regularization term is represented by the shaded ball. Here, our weight coeffcients cannot exceed our regularization *budget*—the combination of the weight coeffcients cannot fall outside the shaded area. On the other hand, we still want to minimize the cost function. Under the penalty constraint, our best effort is to choose the point where the L2 ball intersects with the contours of the unpenalized cost function. The larger the value of the regularization parameter λ gets, the faster the penalized cost function grows, which leads to a narrower L2 ball. For example, if we increase the regularization parameter towards infnity, the weight coeffcients will become effectively zero, denoted by the center of the L2 ball. To summarize the main message of the example: our goal is to minimize the sum of the unpenalized cost function plus the penalty term, which can be understood as adding bias and preferring a simpler model to reduce the variance in the absence of sufficient training data to ft the model.

Following illustrate L1 regulariation and sparsity.





The resulting plot provides us with further insights about the behavior of L1 regularization. As we can see, all features weights will be zero if we penalize the model with a strong regularization parameter (*C* < 0.1); *C* is the inverse of the regularization parameter λ .

**Sequential feature selection algorithms**

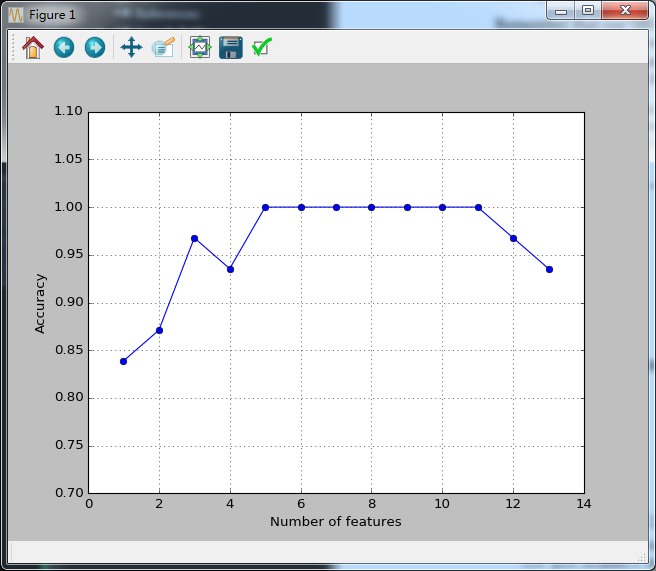
Dimensionality reduction:

1. Feature selection: select a subset of the original features
2. Feature extraction: derive information from the feature set to construct a new feature subspace
3. Sequential Backward Selection (SBS):

SBS sequentially removes features from the full feature subset until the new feature subspace contains the desired number of features.4 simple steps:

1. Initialize the algorithm with *k =d*  , where *d* is the dimensionality of the full  
   feature space **X***d* .
2. Determine the feature *x*- that maximizes the criterion  where .
3. Remove the feature *x*- from the feature set: 
4. Terminate if k equals the number of desired features, if not ,go to step 2.

Source code:github



Scikit-learn feature selection methods: recursive backward elimination based on feature weights, tree-based methods to select features by importance, and univariate statistical test.

**Assessing feature importance with random forests**

Using a random forest, we can measure feature importance as the averaged impurity decrease computed from all decision trees in the forest without making any assumptions whether out data is linearly separable or not.

Reference :

Python machine learning.books