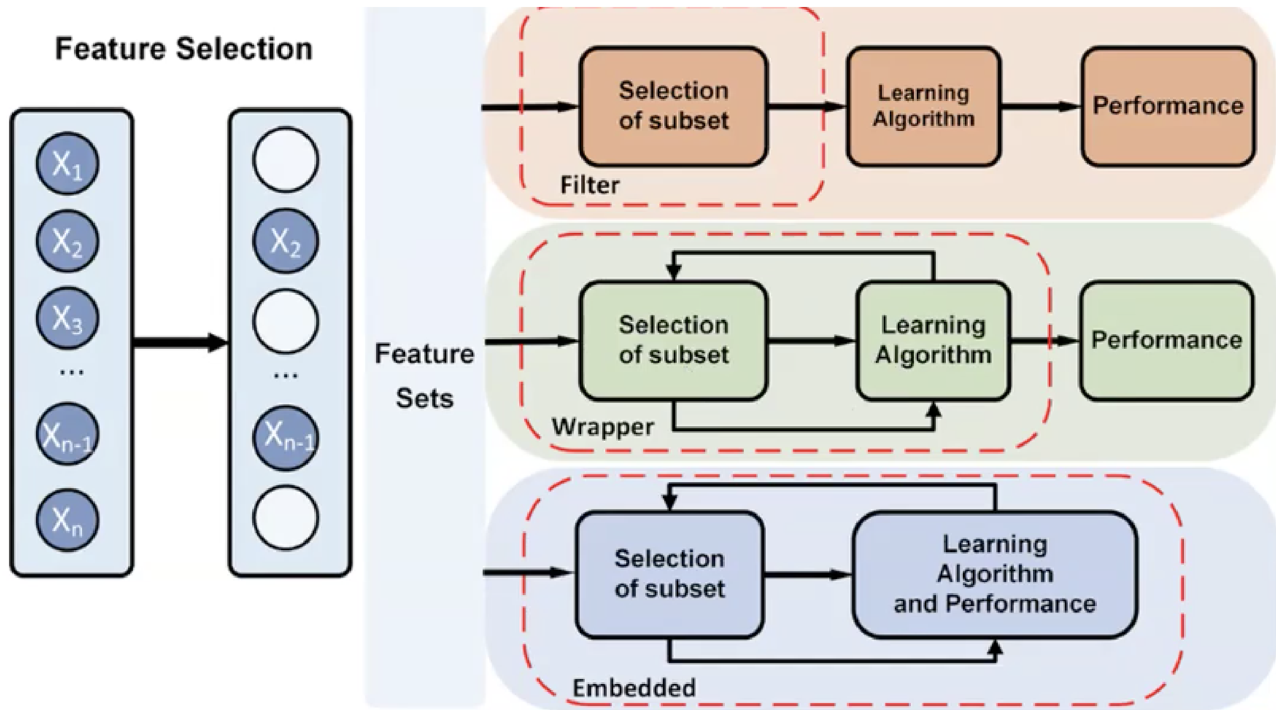
the process of reducing the number of input variables when developing a predictive model. This reduces **computation cost** and improves model performance. We may also select features using statistical based methods by evaluating the relationships between the input and target variable.

Methods of Feature Selection



Unsupervised

do **not** use the target variable. Usually just measures correlation between the independent variables.

Supervised

uses the target variable.

we may use methods such as

- **Recursive Feature Elimination (RFE)**

- **Filters** - selecting subsets of features based on their relationship with the target
    - uses statistical methods
    - feature importance methods
- **Intrinsic** - algorithms that perform automatic feature selection during training
    - decision trees
        - select features during the **splitting process**

Wrapper Based Feature Selection

creates **many models with different subsets of input features** and selects the features that result in the best performing model according to a performance metric

- computationally expensive

Filter Based Feature Selection

uses **correlation type statistic measures** between input and output variables as the basis

- these statistics are generally calculated one input variable at a time with the target variable (univariate statistical measures)
- **any interaction between input variables is *not* considered in the filtering process**

Statistics methods for filter-based feature selection

Numerical input & output -> use Regression Predictive Modeling

- Pearson's coefficient (linear relationships)
- Spearman's rank coefficient (non-linear relationships)

Numerical input, Categorical output -> use Classification

- **ANOVA** Correlation coefficient

Sklearn Library

- sklearn.feature_selection

Correlation Statistics

- Pearsons Correlation Coefficient: f_regression()
- ANOVA: f_classif()
- chi-squared: chi2()
- mutual

Selection Functions

- **SelectKBest** - pics the top k variables
- **SelectPrecentile** - selects the top percentile variables

```
from sklearn.feature_selection import SelectKbest
from skkearn.feature_selection import f_regression

# dfine the feature selection
fs = SelectKBest(score_func=f_regression, k=10)
```

Preprocessing Techniques for Feature Selection

sometimes, we may want to apply transformations or preprocessing strategies on data that are not intuitive. For example, we may want to...

- transform a categorical variable to an ordinal one, even if it is not and see if anything interesting happens
- make a numerical variable discrete

- transform variables to fit a normal distribution

```
from sklearn.datasets import make_friedman1
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
X, y = make_friedman1(n_samples=50, n_features=10, random_state=0) #only first 5 attributes are meaningful
estimator = SVR(kernel="linear")
selector = RFE(estimator, n_features_to_select=5, step=1)
selector = selector.fit(X, y)
print(selector.support_)
selector.ranking_
```

```
[ True  True  True  True  True False False False False False]
array([1, 1, 1, 1, 1, 6, 4, 3, 2, 5])
```

Linear Regression for Feature Selection

```
from sklearn.datasets import make_regression
from sklearn.feature_selection import SelectKBest,
f_regression

# generating the data set
X, y = make_regression(n_samples=100, n_features=100,
n_informative=10)
```

we generate 100 samples with 100 features where we will select the 10 most informative samples from the dataset.

```
# define feature selection
fs = SelectionKBest(score_func=f_regression, k=10)

# apply feature selection
x_selected = fs.fit_transform(X,y)
print(X_selected.shape)
```

ANOVA Feature Selection

```
from sklearn.datasets import make_classification
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

```
# generate the data set
X, y = make_classification(n_samples = 100, n_features=20,
n_informative=2)
```

we generate some samples with 20 features and will make 2 of them informative. We then define the feature selection function.

```
fs = SelectKbest(score_func_classif, k=2)

X_selected = fs.fit_transform(X, y)

# selected features from the data
names = list(range(20))
fs.get_feature_names_out(names)
```

this outputs an array of what the model determines are the most informative attributes from the data

The Curse of Dimensionality

many algorithms work well in low dimensions become intractable when the input dimension is high

Increase in Dimensionality means...

- increases the volume of the space
- decrease in the density of data
- insufficient data to get good estimates for any method
  when ever you add more features to the data set theres always a risk of added noise.

For example, we may have a moderate dimension of size 100, each with a binary input so we then have $2^{100} = 10^{30}$ possible configurations of the data. A Huge training set of a trillion examples would only cover a fraction of that!

Overfitting

100% accurate on the training data but only 50% accurate on the test data due to overfitting

Generalization error

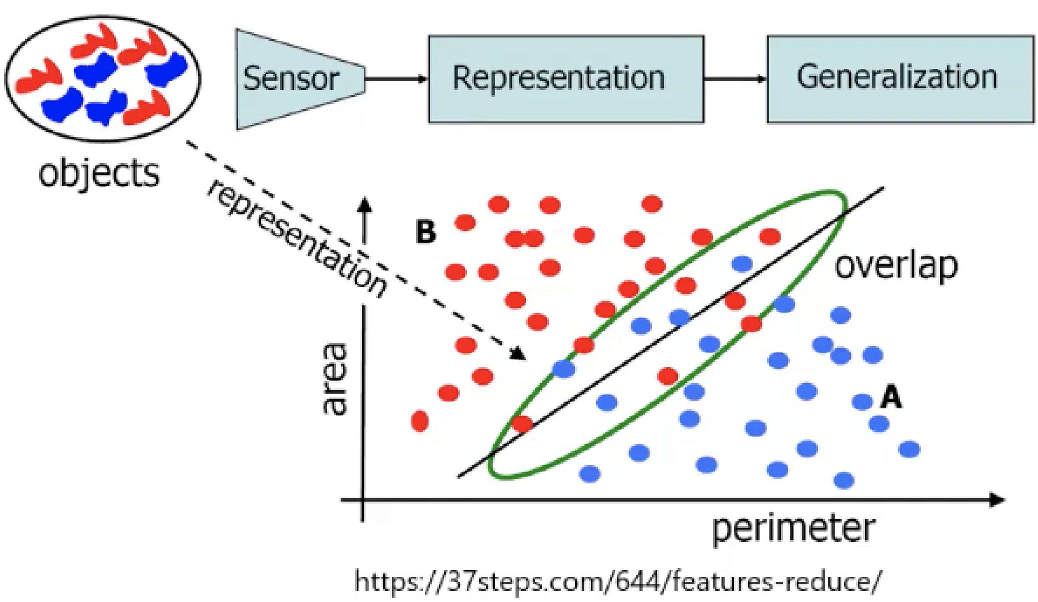
- Training data does not represent the population properly
- there is a tendency to learn random things irrespective of the real signal

Combatting Overfitting

- **cross validation**
- add a **regularization term** to the evaluation function
- penalize classifiers with more structure, thereby favoring smaller ones with less room to overfitting

efficient signal representation

- either select good attributes to train over from the dataset
- or use **projections**



we can select the black line in the middle as the new axis for which to denote the boundary for classification between the area and perimeter classes.