Data Preprocessing

IMLP Ch3.3

IMLP Ch 4.1-4.4

Preprocessing

- Scaling and Normalizing
- Imputing missing data
- Find a technique to represent categorical data
- Binning (Discretization)
- Removing Features
- Generating Polynomial Features
- Custom Transformation
- Combining Features (covered in later chapters)

Need for preprocessing

- Some models contain built-in *feature selection*, meaning that the model will only include predictors that help maximize accuracy. In these cases, the model can pick and choose which representation of the data is best.

Example

One piece of information in the data is the submission date of the grant. This date can be represented in myriad ways:

- The number of days since a reference date
- Isolating the month, year, and day of the week as separate predictors
- The numeric day of the year (ignoring the calendar year)
- Whether the date was within the school year (as opposed to holiday or summer sessions)

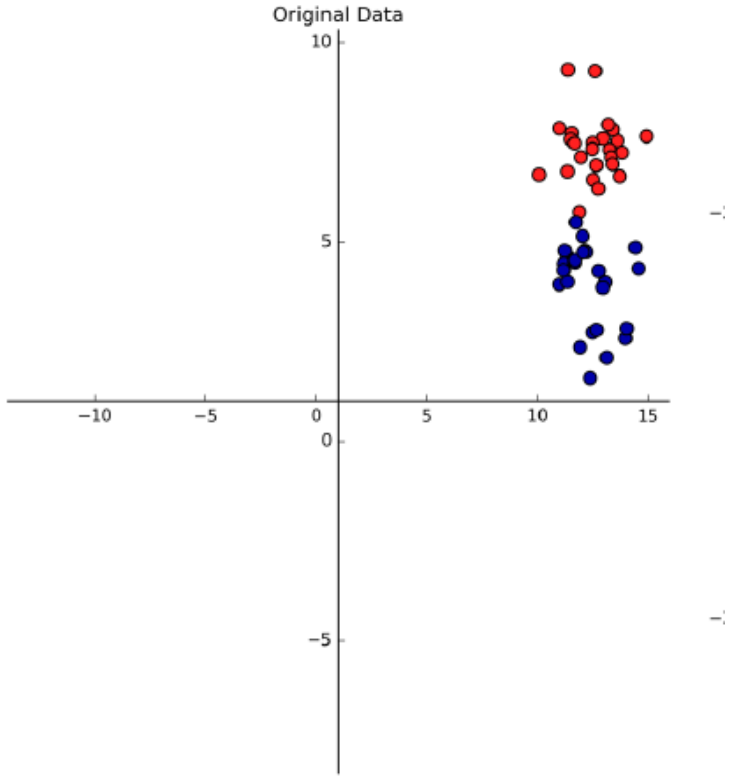
The “correct” feature engineering depends on several factors. First, some encodings may be optimal for some models and poor for others.

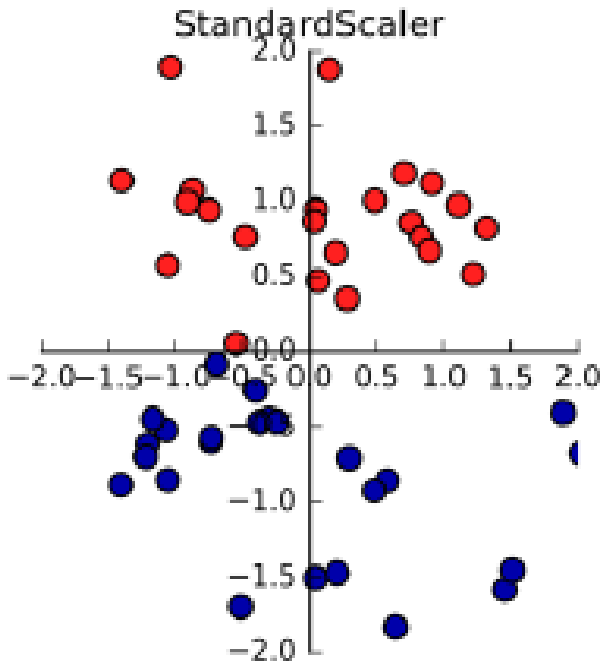
Centering and Scaling

- The most straightforward and common data transformation is to center scale the predictor variables. To center a predictor variable, the average predictor value is subtracted from all the values. As a result of centering, the predictor has a zero mean.

- Similarly, to scale the data, each value of the predictor variable is divided by its standard deviation. Scaling the data coerce the values to have a common standard deviation of one.

Standard Scalar





In scikit-learn this is called StandardScalar which makes the mean 0 and standard deviation 1.

Standard scalar

X = [**X** –mean(X)] / standard_deviation(X)

X = [3 2 5];

Mean(X) = 10/3 = 3.33

Standard Deviation = $\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$

= sqrt [((3-3.33)^2 + (2-3.33)^2 + (5-3.33)^2)/3]

= sqrt[(-.33)^2 + (-1.33)^2 + (1.67)^2))/3] =

= sqrt[(.1089 + 1.7689 + 2.7889) /3] = sqrt(4.666/3) = sqrt(1.5556) = 1.247

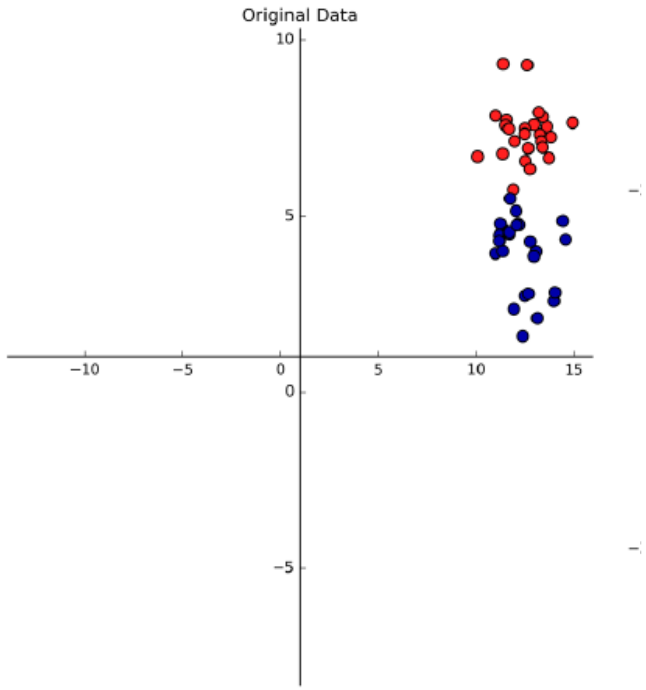
Standard Deviation = 1.247

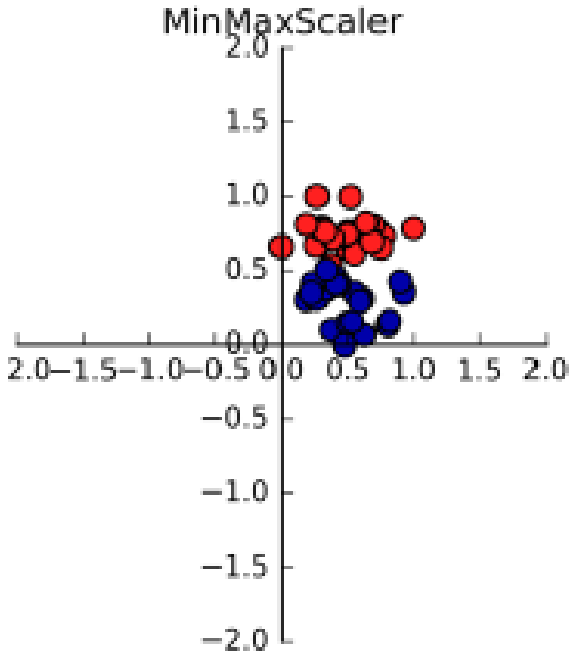
X_1 = (3-3.33)/1.247 = -0.2646 X_2 = (2-3.33)/1.247 = =1.33/1.247 = -1.066

X_3 = (5-3.33)/1.247 = 1.67/1.247 = 1.331

$\mathbf{X}_{\text{stdScaled}}$ = [-0.2646 -1.066 1.331]

MinMax Scalar





The MinMaxScaler, on the other hand, shifts the data such that all features are exactly between 0 and 1. For the two-dimensional dataset this means all of the data is contained with rectangle of (0,0), (0,1), (1,0), (1,1)

MinMax Scalar

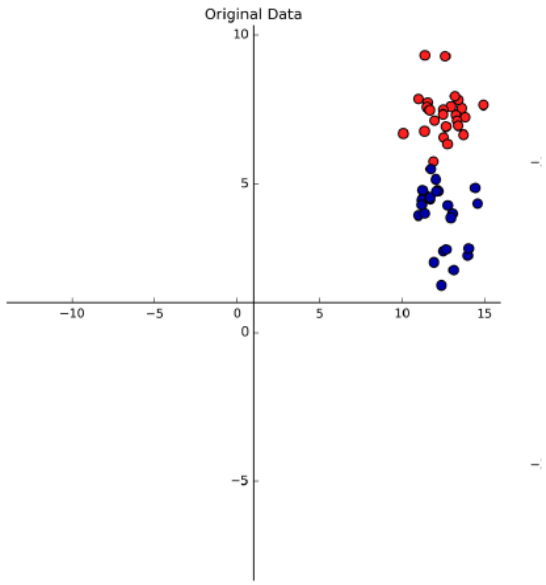
$\mathbf{X} = (\mathbf{X} - X_{min}) / (X_{max} - X_{min})$

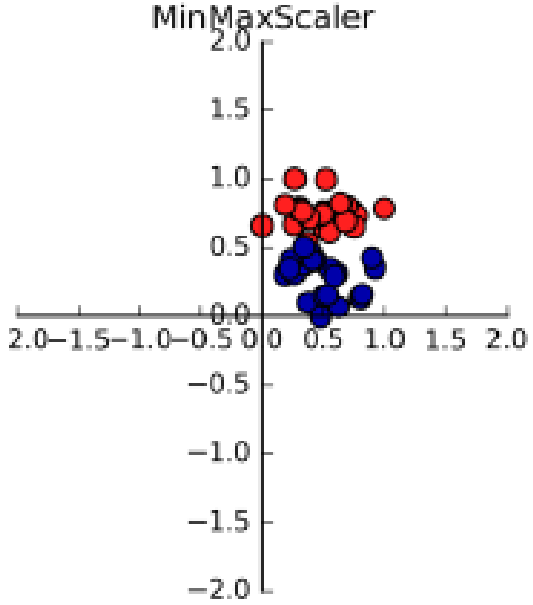
$\mathbf{X} = [3\ 2\ 5]$;
$X_{min} = 2$; $X_{max} = 5$;

$X_1 = (3-2)/(3) = 0.33$
$X_2 = (2-2)/3 = 0$
$X_3 = (5-2)/3 = 0.66$

$X_{MinMaxScaled} = [0.33\ 0\ 0.66]$

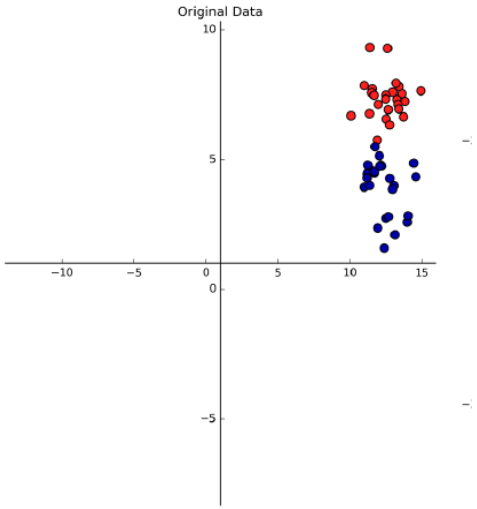
Robust Scalar

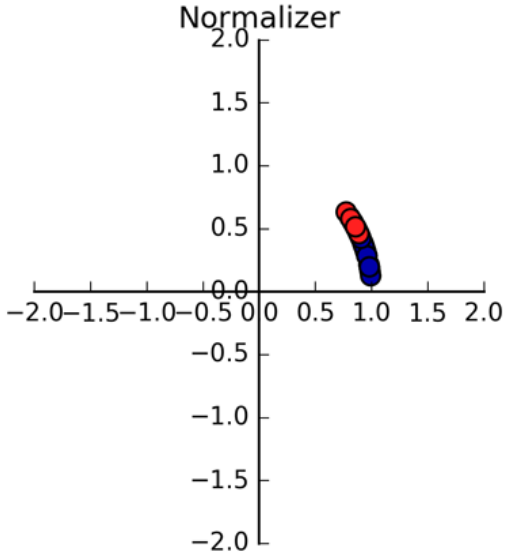




However, the RobustScaler uses the median and quartiles,1 instead of mean and variance. This makes the RobustScaler ignore data points that are very different from the rest (like measurement errors). These odd data points are also called *outliers*, and can lead to trouble for other scaling techniques.

Normalizer





Finally, the Normalizer does a very different kind of rescaling. It scales each data point such that the feature vector has a Euclidean length of 1. This means every data point is scaled by a different number (by the inverse of its length). This normalization is often used when only the direction (or angle) of the data matters, not the length of the feature vector.

Computing

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
```

```
transformed shape: (426, 30)
```

```
per-feature minimum before scaling:
```

```
[ 6.981 9.71 43.79 143.5 0.053 0.019 0. 0. 0.106 0.05 0.115 0.36 0.757 6.802 0.002
0.002 0. 0. 0.01 0.001 7.93 12.02 50.41 185.2 0.071 0.027 0. 0. 0.157 0.055]
```

```
per-feature maximum before scaling:
```

```
[ 28.11 39.28 188.5 2501. 0.163 0.287 0.427 0.201 0.304 0.096 2.873 4.885 21.98 542.2
0.031 0.135 0.396 0.053 0.061 0.03 36.04 49.54 251.2 4254. 0.223 0.938 1.17 0.291 0.577
0.149]
```

```
per-feature minimum after scaling:
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.]
```

```
per-feature maximum after scaling:
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.]
```

Test Data Transformation

You must apply the SAME transformation to test data as you do to training data.

DO THIS

```
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

DO NOT DO THIS

```
test_scaler = MinMaxScaler()
test_scaler.fit(X_test)
X_test_scaled_badly = test_scaler.transform(X_test)
```

When to scale data?

- SCALING DATA NEVER HURTS!

DOES IT HELP?

a) When using algorithms that have gradient descent technique, such as linear regression, logistic regression, SVM etc. the number of iterations will be much less for the algorithm to converge.
b) Neural Networks require data to be scaled using min-max scalar.
c) Scaling will have minimal effect (and might hurt in rare occasions due to rounding) in tree-based algorithms
d) For kNN it has no effect.
e) General rule of thumb- Depends on which algorithm you are using
f) Normalize only if using dot product in the kernel

Preprocessing

- Scaling and Normalizing
- Imputing missing data
- Find a technique to represent categorical data
- Binning
- Removing Features
- Generating Polynomial Features
- Custom Transformation
- Introduce a single feature that combines multiple features (aka *feature engineering*)

Missing Values

Patient forgetting to enter the number of children in a form during doctor's visit

Guests leaving questions blank in review form at a hotel

Library book has a check out date but not a check in date (not yet returned)

What are ways to fill in missing values?

Strategies

1. Ignore the column with missing values (Delete)
2. Fill with random value between min and max of the column
3. Fill with mean of other values in the column
4. Fill with minimum of other values in the column
5. Fill with mode (most frequently occurring data)
6. Fill with median of the values in the column
7. Fill with a given constant value of the column

Computation

The SimpleImputer class provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located. This class also allows for different missing values encodings.The following snippet demonstrates how to replace missing values, encoded as np.nan, using the mean value of the columns (axis 0) that contain the missing values:

```
fit is used to compute the averages and transform uses these to fill the missing values.
```

Computation

```
>>>>>> import numpy as np
>>> from sklearn.impute import SimpleImputer
>>> imp = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])
SimpleImputer()
>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]
>>> (imp.transform(X))
 [[4. 2. ] [6. 3.666...] [7. 6. ]]
```

Missing Values

- Other techniques include using k-nearest neighbors to fill in missing data. kNNImputer.
- Using other column values to “guess” this column. Multivariate Imputers.

Preprocessing

- Scaling and Normalizing
- Imputing missing data
- Find a technique to represent categorical data
- Binning
- Removing Features
- Generating Polynomial Features
- Custom Transformation
- Combining Features (covered in later chapters)

Dealing with Categorical Variables

- Categorical variables are those that have values in categories like (high, low, medium).
- Representing these as numbers creates problems since any math computed using these numbers is meaningless
- How do we handle categorical data?

Dealing with Categorical Variables

- Consider the dataset of adult incomes in the United States, derived from the 1994 census database.
- The task of the adult dataset is to predict whether a worker has an income of over $50,000 or under $50,000.
- The features in this dataset include the workers' ages, how they are employed (self employed, private industry employee, government employee, etc.), their education, their gender, their working hours per week, occupation, and more.
- Much of this data is categorical

Sample Categorical Data

Table 4-1. The first few entries in the adult dataset

	age	workclass	education	gender	hours-per-week	occupation	income
0	39	State-gov	Bachelors	Male	40	Adm-clerical	<=50K
1	50	Self-emp-not-inc	Bachelors	Male	13	Exec-managerial	<=50K
2	38	Private	HS-grad	Male	40	Handlers-cleaners	<=50K
3	53	Private	11th	Male	40	Handlers-cleaners	<=50K
4	28	Private	Bachelors	Female	40	Prof-specialty	<=50K
5	37	Private	Masters	Female	40	Exec-managerial	<=50K
6	49	Private	9th	Female	16	Other-service	<=50K
7	52	Self-emp-not-inc	HS-grad	Male	45	Exec-managerial	>50K
8	31	Private	Masters	Female	50	Prof-specialty	>50K
9	42	Private	Bachelors	Male	40	Exec-managerial	>50K
10	37	Private	Some-college	Male	80	Exec-managerial	>50K

Working class, gender, education, occupation and income are all categorical data. Not amenable to plug in into Logistic regression equation. There are 7 columns of data here.

One Hot Encoding (ohe)- Dummy Variables

Table 4-2. Encoding the workclass feature using one-hot encoding

workclass	Government Employee	Private Employee	Self Employed	Self Employed Incorporated
Government Employee	1	0	0	0
Private Employee	0	1	0	0
Self Employed	0	0	1	0
Self Employed Incorporated	0	0	0	1

We create four dummy variables WorkClass_Government, WorkClass_Private, WorkClass_Self and WorkClass_SelfIncorporated. We do this for each of the categorical variables.. HOW MANY ARE CREATED?

One Hot Encoding

	age	hours-per-week	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	...	occupation_ Tech-support	occupation_ Transport-moving	income_ <=50K	income_ >50K
0	39	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
1	50	13	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
2	38	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
3	53	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
4	28	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0

5 rows × 46 columns

There are 46 columns once we convert categorical variables using ohe.

Numbers can be categorical

	Categorical Feature	Integer Feature
0	socks	0
1	fox	1
2	socks	2
3	box	1

Table 4-6. One-hot encoding of the data shown in Table 4-4, encoding the integer and string features

	Integer Feature_0	Integer Feature_1	Integer Feature_2	Categorical Feature_box	Categorical Feature_fox	Categorical Feature_socks
0	1.0	0.0	0.0	0.0	0.0	1.0
1	0.0	1.0	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0	0.0	1.0
3	0.0	1.0	0.0	1.0	0.0	0.0

Computation using Pandas

```
import os
# The file has no headers naming the columns, so we pass header=None
# and provide the column names explicitly in "names"
adult_path = os.path.join(mglearn.datasets.DATA_PATH, "adult.data")
data = pd.read_csv(
adult_path, header=None, index_col=False,
names=['age', 'workclass', 'fnlwgt', 'education', 'education-num',
'marital-status', 'occupation', 'relationship', 'race', 'gender',
'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
'income'])
# For illustration purposes, we only select some of the columns
data = data[['age', 'workclass', 'education', 'gender', 'hours-per-week',
'occupation', 'income']]
```

Computation using Pandas

```
print("Original features:\n", list(data.columns), "\n")
data_dummies = pd.get_dummies(data)
print("Features after get_dummies:\n", list(data_dummies.columns))
```

Original features:

```
['age', 'workclass', 'education', 'gender', 'hours-per-week', 'occupation',
'income']
```

Features after get_dummies:

```
['age', 'hours-per-week', 'workclass_ ?', 'workclass_ Federal-gov', 'workclass_
Local-gov', 'workclass_ Never-worked', 'workclass_ Private', 'workclass_ Self-emp-
inc', 'workclass_ Self-emp-not-inc', 'workclass_ State-gov', 'workclass_ Without-
pay', 'education_ 10th', 'education_ 11th', 'education_ 12th', 'education_ 1st-
4th', ... 'education_ Preschool', 'education_ Prof-school', 'education_ Some-
college', 'gender_ Female', 'gender_ Male', 'occupation_ ?', 'occupation_ Adm-
clerical', 'occupation_ Armed-Forces', 'occupation_ Craft-repair', 'occupation_
Exec-managerial', 'occupation_ Farming-fishing', 'occupation_ Handlers-cleaners',
... 'occupation_ Tech-support', 'occupation_ Transport-moving', 'income_ <=50K',
'income_ >50K']
```

Computation with Scikit Learn (ColumnTransformer)

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import pandas as pd
import numpy as np
```

Computation with Scikit Learn (ColumnTransformer)

```
s = pd.DataFrame(data={'Category': ['A', 'A', np.nan, 'B']})
category_pipeline = Pipeline(steps=[
('imputer', SimpleImputer(missing_values=np.nan, strategy='most_frequent')),
('ohe', OneHotEncoder(sparse=False))
])
transformer = ColumnTransformer(transformers=[('category', category_pipeline ,
['Category'])],)
a = transformer.fit_transform(s)
print(a)
```

```
[[1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]]
```

Computation with Scikit Learn (make_ColumnTransformer)

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.compose import make_column_transformer
import numpy as np
```

Computation with Scikit Learn (make_ColumnTransformer)

```
df = pd.DataFrame({'Cat_Var': np.random.choice(['a', 'b'],
size=3),'Num_Var': np.arange(3)})

cat_cols = ['Cat_Var']

num_cols = ['Num_Var']

col_transformer = make_column_transformer(

(OneHotEncoder(), cat_cols),

remainder=StandardScaler())

X = col_transformer.fit_transform(df)

print(X)
```

```
[[ 1.          0.         -1.22474487]
 [ 0.          1.          0.        ]
 [ 1.          0.          1.22474487]]
```

Label Encoding

- If you need to convert text data to numeric by simply replacing label with a number you can use this..
- **Can be used ONLY for target not for data.**

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(["paris", "paris", "tokyo", "amsterdam"])
list(le.classes_)
['amsterdam', 'paris', 'tokyo']

le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1]...)
(le.inverse_transform([2, 2, 1])) ['tokyo', 'tokyo', 'paris']
```

Preprocessing

- Scaling and Normalizing
- Imputing missing data
- Find a technique to represent categorical data
- Binning
- Removing Features
- Generating Polynomial Features
- Combining Features (covered in later chapters)

Binning(Discretization)

Converting continuous variables to discrete variables

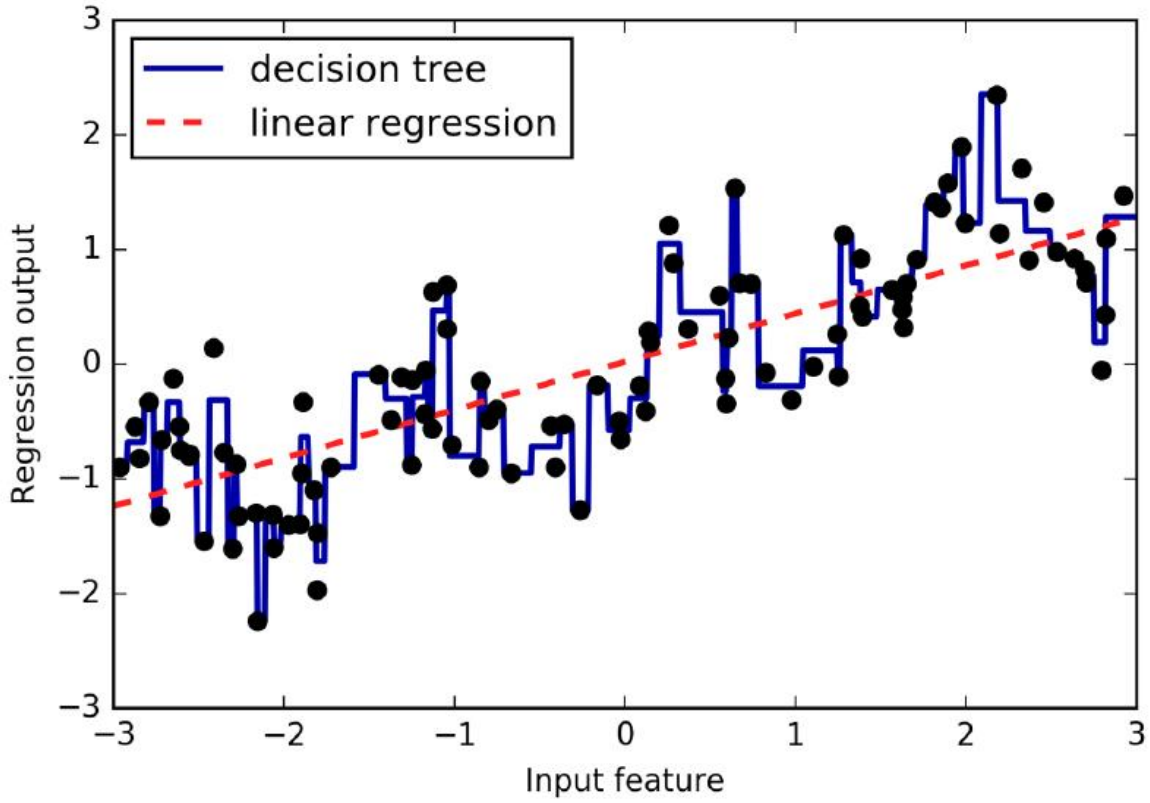


Figure 4-1. Comparing linear regression and a decision tree on the wave dataset

Putting the numbers between -3 to 3 into 10 bins

Computation (scikit-learn)

```
from sklearn.preprocessing import KBinsDiscretizer
kb = KBinsDiscretizer(n_bins=10, strategy='uniform')
kb.fit(X)
print("bin edges: \n", kb.bin_edges_)
```

```
[array([-2.967, -2.378, -1.789, -1.2 , -0.612, -0.023, 0.566, 1.155, 1.744, 2.333,
2.921])]
```

Binning

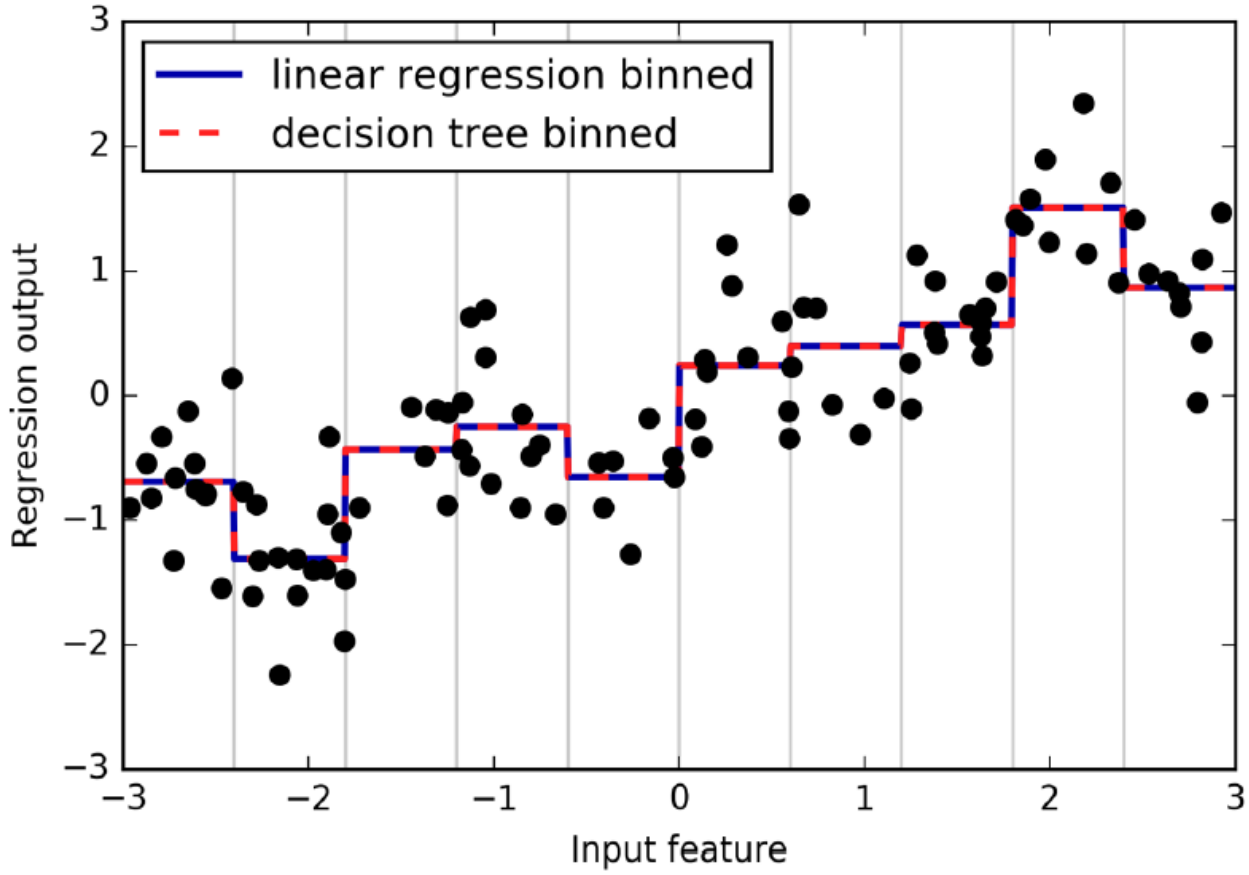


Figure 4-2. Comparing linear regression and decision tree regression on binned features

Preprocessing

- Scaling and Normalizing
- Imputing missing data
- Find a technique to represent categorical data
- Binning
- Removing Features
- Generating Polynomial Features
- Introduce a single feature that combines multiple features (aka *feature engineering*)

Removing Features

- With so many ways to create new features, you might get tempted to increase the dimensionality of the data way beyond the number of original features.
- However, adding more features makes all models more complex, and so increases the chance of overfitting.
- Removing features can lead to simpler models that generalize better.
- Removing features is done by PCA analysis that will be covered in a later chapter.
- Here we will cover three techniques for removing features

Correlation between two variables.

The features that that have low correlation (close to zero defined by a threshold) are removed. Positive value of correlation indicates the variables are move together in same direction, negative correlation means the variables move in opposite directions and zero indicates no correlation.

Correlation is defined as covariance divided by product of standard deviations.

Correlation=ρ = $\frac{\mathrm{cov}(X,Y)}{\sigma_x . \sigma y}$,

Covariace, cov(x,y) is given by cov(x,y) = $\frac{\sum_{i=1}^{i=n}(x_i-\bar{x}) .(y_i-\bar{y})}{(n-1)}$

x_i = a given x value in the data set

$\bar{x}$= the mean, or average, of the x values

y_i = the y value in the data set that corresponds with x_i

$\bar{y}$ = the mean, or average, of the y values

n = the number of data points

Computation

- Computing correlation between two variables using python (scipy).
- It is called Pearson's correlation.

```
Pearson's correlation coefficient = covariance(X, Y) /
(stdv(X) * stdv(Y))
```

Standard Deviation=σ_x = $\frac{(\sum_{i=1}^{i=n}(x_i-\bar{x})}{(\mathbf{n-1})}$

x_i=value of the i^{th} point in the data set
$\bar{x}$=mean value of the data set
n=the number of data points in the data

```
from scipy.stats import pearsonr
corr, _ = pearsonr(data1, data2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 0.888
```

Univariate Feature Selection

- In univariate statistics, we compute whether there is a statistically significant relationship between each feature and the target.
- Then the features that are related with the highest confidence are selected. also known as *analysis of variance* (ANOVA).
- This is based on variance within the data.
- ANOVA is *univariate*, meaning that they only consider each feature individually.

Computation

```
from sklearn.datasets import load_breast_cancer
from sklearn.feature_selection import SelectPercentile
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
X_w_noise, cancer.target, random_state=0, test_size=.5)
# use f_classif (the default) and SelectPercentile to select 50% of features
select = SelectPercentile(percentile=50)
select.fit(X_train, y_train)
# transform training set
X_train_selected = select.transform(X_train)
```

Output

```
X_train.shape: (284, 80)
X_train_selected.shape: (284, 40)
```

Model-Based Feature Selection

- Model-based feature selection uses a supervised machine learning model to judge the importance of each feature, and keeps only the most important ones.
- The supervised model that is used for feature selection doesn’t need to be the same model that is used for the final supervised modeling.

Computation

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
select = SelectFromModel( RandomForestClassifier(n_estimators=100,
random_state=42),
threshold="median")
select.fit(X_train, y_train)
X_train_l1 = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_l1.shape: {}".format(X_train_l1.shape))
```

Output

```
X_train.shape: (284, 80)
X_train_l1.shape: (284, 40)
```

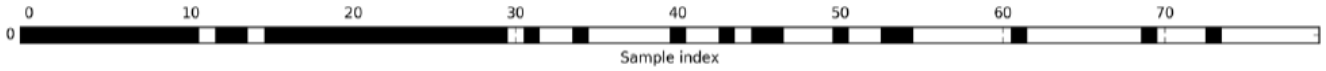


Figure 4-10. Features selected by SelectFromModel using the RandomForestClassifier

Iterative Feature Selection

- In univariate testing we used no model, while in model-based selection we used a single model to select features.
- In iterative feature selection, a series of models are built,

with varying numbers of features.

- There are two basic methods:
  1. starting with no features and adding features one by one until some stopping criterion is reached, or
  2. starting with all features and removing features one by one until some stopping criterion is reached.
- One of the techniques is called Recursive Feature Elimination
- This is a very time consuming technique

Computation

```
from sklearn.feature_selection import RFE
select = RFE(RandomForestClassifier(n_estimators=100, random_state=42),
n_features_to_select=40)
select.fit(X_train, y_train)
# visualize the selected features:
mask = select.get_support()
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Sample index")
plt.yticks(())
```

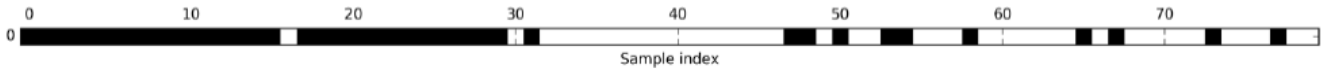


Figure 4-11. Features selected by recursive feature elimination with the random forest classifier model

Utilizing Expert Knowledge

How can experts help?

- Add a feature
- Identify binning parameters
- Find things that are completely out of range
- Categorical data fits in the right categories
- Help fill in missing data
- Representation of the data (dates, years)

Preprocessing

- Scaling and Normalizing
- Imputing missing data
- Find a technique to represent categorical data
- Binning (Discretization)
- Removing Features
- Generating Polynomial Features
- Custom Transformation
- Combining Features (covered in later chapters)

Generating Polynomial Features

- Often it's useful to add complexity to a model by considering nonlinear features of the input data. We show two possibilities that are both based on polynomials: The first one uses pure polynomials, the second one uses splines, i.e. piecewise polynomials.

- X_1, X_2 will be transformed to (1, X_1, X_1^2, X_1.X_2, X_2^2

Generating Polynomial Features

```
import numpy as np >>> from sklearn.preprocessing import
PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1], [2, 3], [4, 5]])
>>> poly = PolynomialFeatures(2)
>>> poly.fit_transform(X)

array([[ 1., 0., 1., 0., 0., 1.],
       [ 1., 2., 3., 4., 6., 9.],
       [ 1., 4., 5., 16., 20., 25.]])
```

Preprocessing

- Scaling and Normalizing
- Imputing missing data
- Find a technique to represent categorical data
- Binning (Discretization)
- Removing Features
- Generating Polynomial Features
- Custom Transformation
- Combining Features (covered in later chapters)

Custom Transformation

- One can use transformation using a function like
- Log(x), tan(x), etc. etc. using FunctionTransformer on scikit-learn

Preprocessing

- Scaling and Normalizing
- Imputing missing data
- Find a technique to represent categorical data
- Binning (Discretization)
- Removing Features
- Generating Polynomial Features
- Custom Transformation
- Combining Features (covered in later chapters)