# Supervised Learning

## Libraries

## Load Data

## Pre-Processing Data

### Column dtypes

df.dtypes

#### Dummy Variables (One Hot Encoding)

dummy = pd.get\_dummies(df, columns=df.columns[(df.dtypes == 'object') | (df.dtypes == 'category')])

df = pd.DataFrame(data=dummy)

df.head()

OR

df['Gender'].replace(to\_replace=['male','female'], value=[0,1],inplace=True)

df.head()

#### Object -> Integer

df = df[pd.to\_numeric(df['BareNuc'], errors='coerce').notnull()]

df['BareNuc'] = df['BareNuc'].astype('int')

df.dtypes

### Feature Scaling

Ensures all features are weighed the same in the algorithm.

#### Standardization (Recommended)

from sklearn.preprocessing import StandardScaler

X= preprocessing.StandardScaler().fit(X).transform(X)

X

# X = pd.DataFrame(data=X, columns=df.columns[1:])

# X.head()

= 0

= 1.

Chart, histogram

Description automatically generated

#### Normalization

from sklearn.preprocessing import MinMaxScaler

X = MinMaxScaler().fit\_transform(X)

X

# X = pd.DataFrame(data=X, columns=df.columns[1:])

# X.head()

## Training, Validating, and Testing the Model

from sklearn.model\_selection import train\_test\_split

X\_train, X\_val\_and\_test, y\_train, y\_val\_and\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=4)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_val\_and\_test, y\_val\_and\_test, test\_size=0.5, random\_state=4)

print(X\_train.shape, X\_val.shape, X\_test.shape, y\_train.shape, y\_val.shape, y\_test.shape)

Training Accuracy: Accuracy on the Training data set (overfit/underfit)

Out-of-Sample Accuracy: Accuracy on future data

1. Train and Test Model on the Same Data Set
   * ↑ Training Accuracy
   * ↓ Out-of-Sample Accuracy
2. Train 70% / Validation 15% / Test 15%
   * --- Acceptable Training Accuracy
   * ↑ Higher Out-of-Sample Accuracy

### K-Fold Cross Validation

A picture containing diagram

Description automatically generated

## Algorithm

### Regression

Prediction Set (1xn)

Predicted Dependant Value

Actual Dependant Value

Mean of all Actual Dependant Values

Parameters Matrix (nx1)

n Number of features

Feature Set (1xn)

Feature i

#### Algorithms

##### Linear Regression

from sklearn import linear\_model

regr = linear\_model.LinearRegression()

regr.fit (X\_train, y\_train)

y\_hat= regr.predict(X\_val)

##### Multiple Linear Regression

from sklearn import linear\_model

regr = linear\_model.LinearRegression()

regr.fit (X\_train, y\_train)

y\_hat= regr.predict(X\_val)

##### Non-Linear Regression

##### Polynomial Linear Regression

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)

x\_train\_poly = poly.fit\_transform(train\_x)

regr = linear\_model.LinearRegression()

regr.fit(x\_train\_poly, y\_train)

y\_hat = regr.predict(X\_val)

#### Optimization

##### Error Functions

Cost Function Minimizes the average error over all training examples.

Loss Function Minimizes the error of a single training example.

###### Loss Functions

Minimizes the error of a single training example.

0 - ꚙ Negatively Orientated

More robust to outliers. Less stable.

0 - ꚙ Negatively Orientated

Less robust to errors. More stable.

0 - ꚙ Negatively Orientated

, otherwise

Best. Quadratic for small errors. Linear for high errors.

###### Cost Functions

Cost Function Minimizes the average error over all training examples.

0 - ꚙ Negatively Orientated

More robust to outliers. Less stable.

0 - ꚙ Negatively Orientated

Less robust to errors. More stable.

0 - ꚙ Negatively Orientated

##### Optimization Algorithms

|  |  |
| --- | --- |
| **Gradient Descent** | **Ordinary Least Squares (Normal Equation)** |
| m>10k | m<10k |
| Kn2 Algorithm Complexity | Kn3 Algorithm Complexity |
| No Regularization | Regularization (non-invertible) |
| Iterative | Analytical |
| Must chose Learning Rate | No Learning Rate |
| Feature Scaling can be used | No need for Feature Scaling |

###### Gradient Descent (>10k)



###### Ordinary Least Squares (Normal Equation) (<10k)

#### Prediction

# Evaluation/Accuracy

Predicted Dependant Value for case i

Actual Dependant Value for case i

Mean of Actual Dependant Value for case i

m Number of data points

## Residual Error

Standard error.

0 - ꚙ Negatively Orientated

## Absolute

Standard error.

0 - ꚙ Negatively Orientated

0 - ꚙ Negatively Orientated

0 - 1 Negatively Orientated

## Square

Exponentially increases error. Larger errors are more significant.

0 - ꚙ Negatively Orientated

0 - ꚙ Negatively Orientated

0 - 1 Negatively Orientated

0 - ꚙ Negatively Orientated

## Explained Variance Score

1 - 0 Positively Orientated

## Coefficient of Determination (R2 Score)

0 - ꚙ Negatively Orientated

0 - ꚙ Negatively Orientated

0 - 1 Negatively Orientated

Coefficient of Determination 1 - 0 Positively Orientated

# Feature Selection

# Regularization

The objective of regression is to minimize the Cost Function a model. To avoid over fitting a model, we add regularization to a model. Regularization adds a Tuning Parameter (λ) to the cost function. The Tuning Parameter is multiplied by the Coefficient Matrix βj.

# Data Visualization

## Bar Graph

import seaborn as sns

bins = np.linspace(df.Principal.min(), df.Principal.max(), 10)

g = sns.FacetGrid(df, col="Gender", hue="loan\_status", palette="Set1", col\_wrap=2)

g.map(plt.hist, 'Principal', bins=bins, ec="k")

g.axes[-1].legend()

plt.show()

Chart, bar chart

Description automatically generated