Simon Fraser University
School of Mechatronic Systems Engineering

MSE491 - Application of Machine Learning in Mechatronic Systems
Spring 2021

Lab 1 – Regression

Amin Kabir
kabir@sfu.ca

Differentiate between Machine Learning and Expert Systems

- An expert system uses ***if-then*** statements when doing inference while an ML system projects the input data into some model space.

Example in Robotics:

1. Robot without ML → deterministic behaviors
2. Robot with ML → learn from experience

The basic difference between a robot using machine learning and a normal robot is the ability of the ML robot and its software to:

- ✓ **make decisions**
- ✓ **learn from experience**
- ✓ **adapt to its environment**
- ✓ **based on data from its sensors**

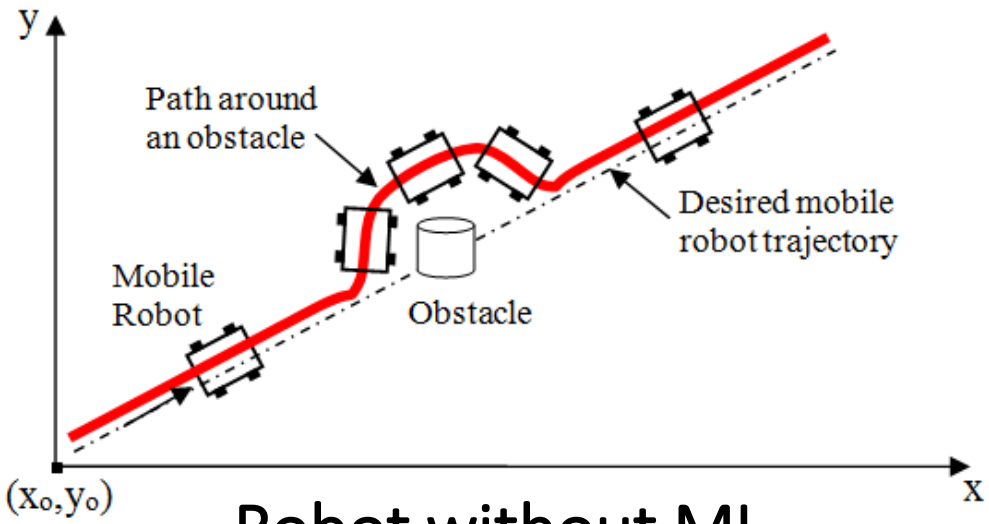
Differentiate between Machine Learning and Expert Systems

- **Robot without ML:**

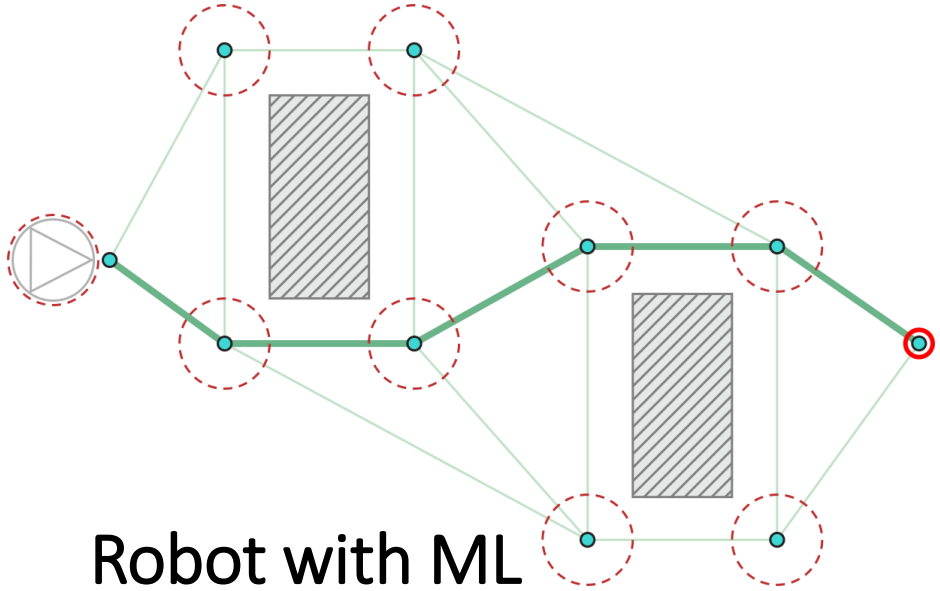
If faced with the same situation, such as encountering an obstacle, then the robot will always do the same thing, such as go around the obstacle to the left.

- **Robot with ML:**

The ML robot will change and adapt to circumstances and may do something different each time a situation is encountered. It may try to push the obstacle out of the way, or make up a new route, or change goals.



Robot without ML



Robot with ML

Scikit-Learn

https://scikit-learn.org/stable/

scikit-learn

Machine Learning in Python

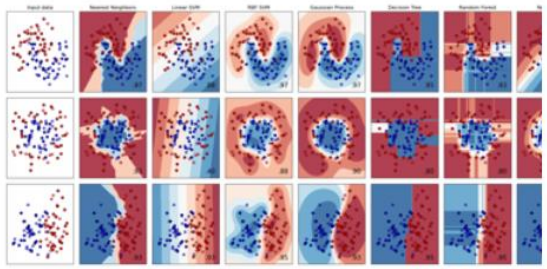
Getting Started | Release Highlights for 0.24 | GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

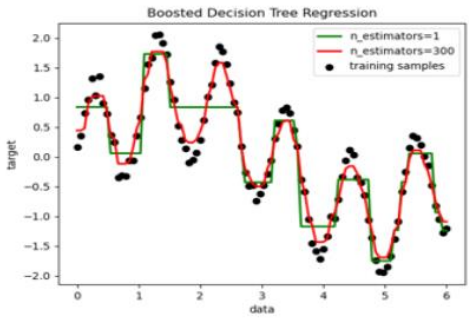


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

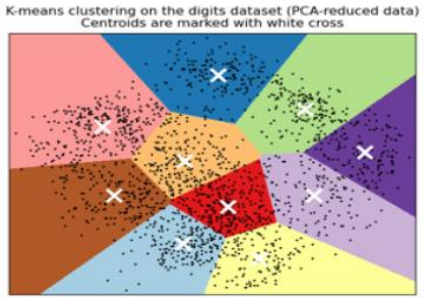


Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...



Examples

Jupyter Notebook

- The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

https://jupyter.org/install

➢Install with Conda:

conda install -c conda-forge notebook

Predict House Sales Prices

With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges you to predict the final price of each home.

https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

LOADING THE DATASET

```
# Load General Libraries
import numpy as np
import pandas as pd
import pickle
import matplotlib.pyplot as plt
```

```
# Importing the dataset
dataset = pd.read_csv('house_price.csv', header=0)
```

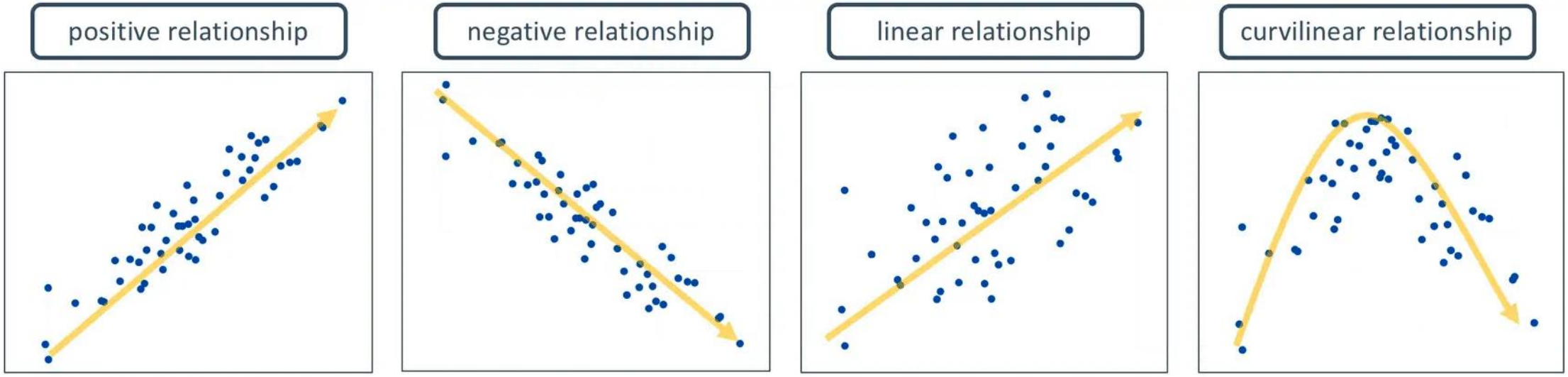
```
# Print one row randomly
dataset.sample()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...	PoolArea	PoolQC	Fence	MiscFeature	Mis
505	90	RM	60.0	7596	Pave	Grvl	Reg	Lvl	AllPub	Inside	...	0	NaN	NaN	NaN	0

1 rows × 80 columns

EXPLORE THE DATASET

Usually, once the data is collected, it must be explored to assess its conditions, including looking for trends, outliers, exceptions, incorrect, inconsistent, missing, or skewed information. This step is essential because your source data will inform your findings, so it is critical to be sure it does not contain unseen biases.



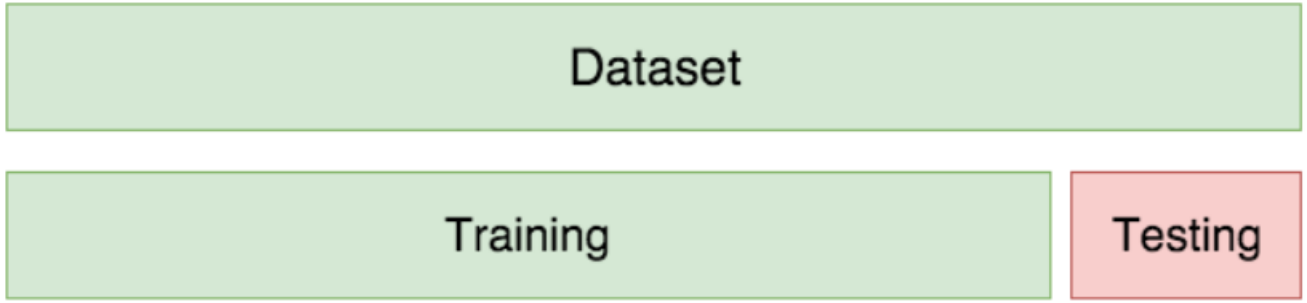
PREPROCESSING

```
# Split features and targets - X: Features, y: Targets
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1].values
```

Preprocessing Features

```
# Handling categorical features
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
cols = X.columns
num_cols = X._get_numeric_data().columns
categorical_cols = list(set(cols) - set(num_cols))
X.loc[:,categorical_cols] = X.loc[:,categorical_cols].astype(str)
for cat_col in categorical_cols:
    ind_cat = X.columns.get_loc(cat_col)
    X.iloc[:,ind_cat] = labelencoder_X.fit_transform(X.iloc[:,ind_cat])

# Create our imputer to replace missing values with the mean
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp = imp.fit(X)
X = imp.transform(X)
```

SPLIT THE DATASET INTO TRAIN & TEST




```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Save Test Set
np.savetxt("features_house_test.csv", X_test, delimiter=",")
np.savetxt("targets_house_test.csv", y_test, delimiter=",")
```

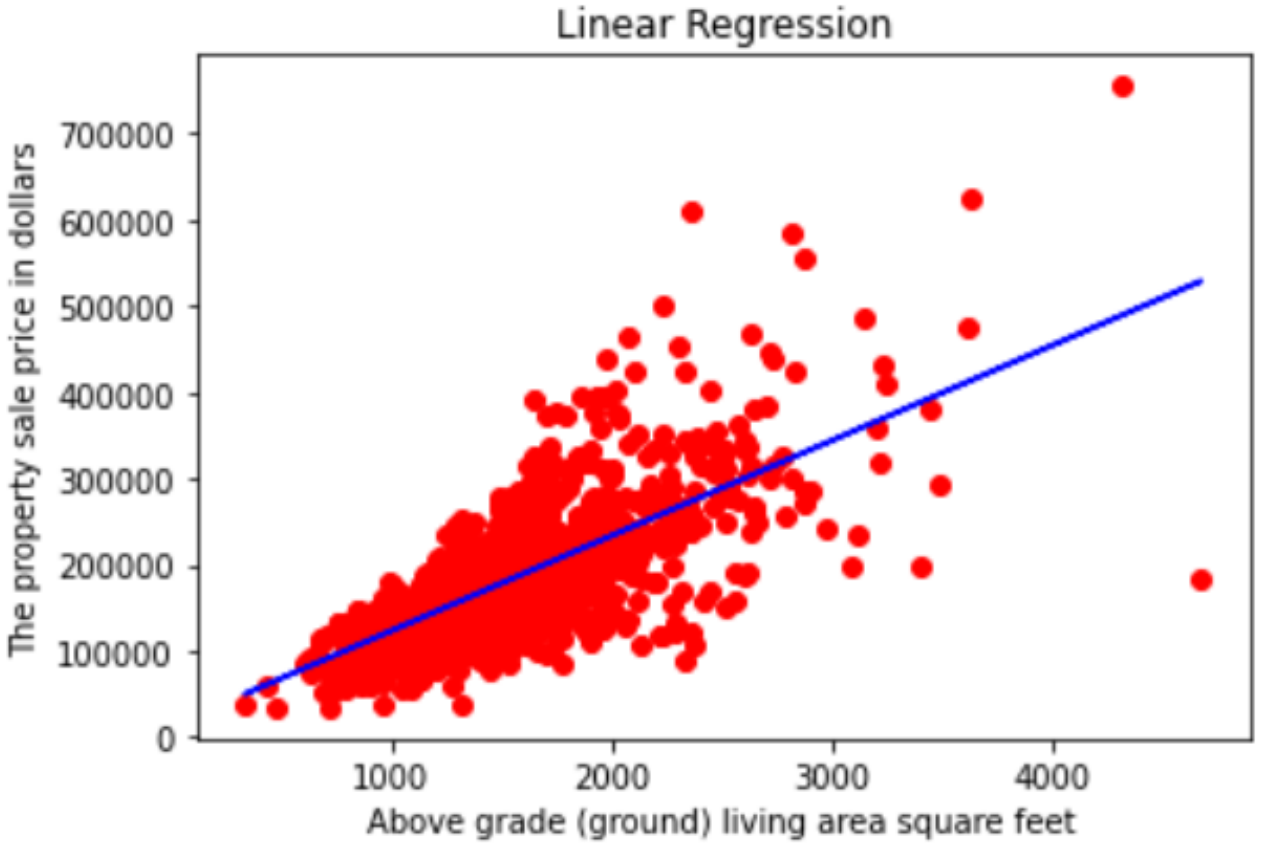
Simple Linear Regression

Simple Linear Regression

```
# Fitting Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
simp_lin_reg = LinearRegression()
```

```
### Simple Linear Regression
# Only using one feature: "Above grade (ground) living area square feet"
X1_train = X_train[:,45]
X1_train = X1_train.reshape(-1, 1)
X1_test = X_test[:,45]
X1_test = X1_test.reshape(-1, 1)
simp_lin_reg.fit(X1_train, y_train)
```

Simple Linear Regression



Simple Linear Regression

- Evaluate the performance of the trained model on the test set

```
# Predicting a random new result
import random
R_test = random.randrange(len(X_test))
Random_Test = X1_test[R_test].reshape(-1, 1)

# Predicting a new result with Linear Regression
y_pred1 = simp_lin_reg.predict(Random_Test)
print('Predicted sale price for sample %d:    %d' %(R_test,y_pred1))
print('True sale price for sample %d:         %d' %(R_test,y_test[R_test]))
```

```
Predicted sale price for sample 4:    133738
True sale price for sample 4:         88000
```

```
# Evaluation
from sklearn.metrics import r2_score
y_pred = simp_lin_reg.predict(X1_test).astype('int64')
lin_reg_r2 = r2_score(y_test, y_pred)
print('\nSimple Linear Regression - R-Squared: %f' %lin_reg_r2)
```

```
Simple Linear Regression - R-Squared: 0.433264
```

Multiple Linear Regression

Multiple Linear Regression

```
### Multiple Linear Regression - using 10 randome features
import random
R10 = random.sample(range(0,X.shape[1]),10)
X10_train = X_train[:,R10]
X10_test = X_test[:,R10]
mult_lin_reg10 = LinearRegression()
mult_lin_reg10.fit(X10_train, y_train)

### Multiple Linear Regression - using all features
mult_lin_reg = LinearRegression()
mult_lin_reg.fit(X_train, y_train)
# Save Model
pickle.dump(mult_lin_reg, open('Model_MLR.pkl', 'wb'))
```

```
# Evaluation
from sklearn.metrics import r2_score
y_pred10 = mult_lin_reg10.predict(X10_test).astype('int64')
lin_reg_r2_10 = r2_score(y_test, y_pred10)
print('\nMultiple Linear Regression using 10 randome features - R-Squared: %f' %lin_reg_r2_10)

y_pred = mult_lin_reg.predict(X_test).astype('int64')
lin_reg_r2 = r2_score(y_test, y_pred)
print('\nMultiple Linear Regression using all features - R-Squared: %f' %lin_reg_r2)
```

```
Multiple Linear Regression using 10 randome features - R-Squared: 0.483302

Multiple Linear Regression using all features - R-Squared: 0.558798
```

Polynomial Regression

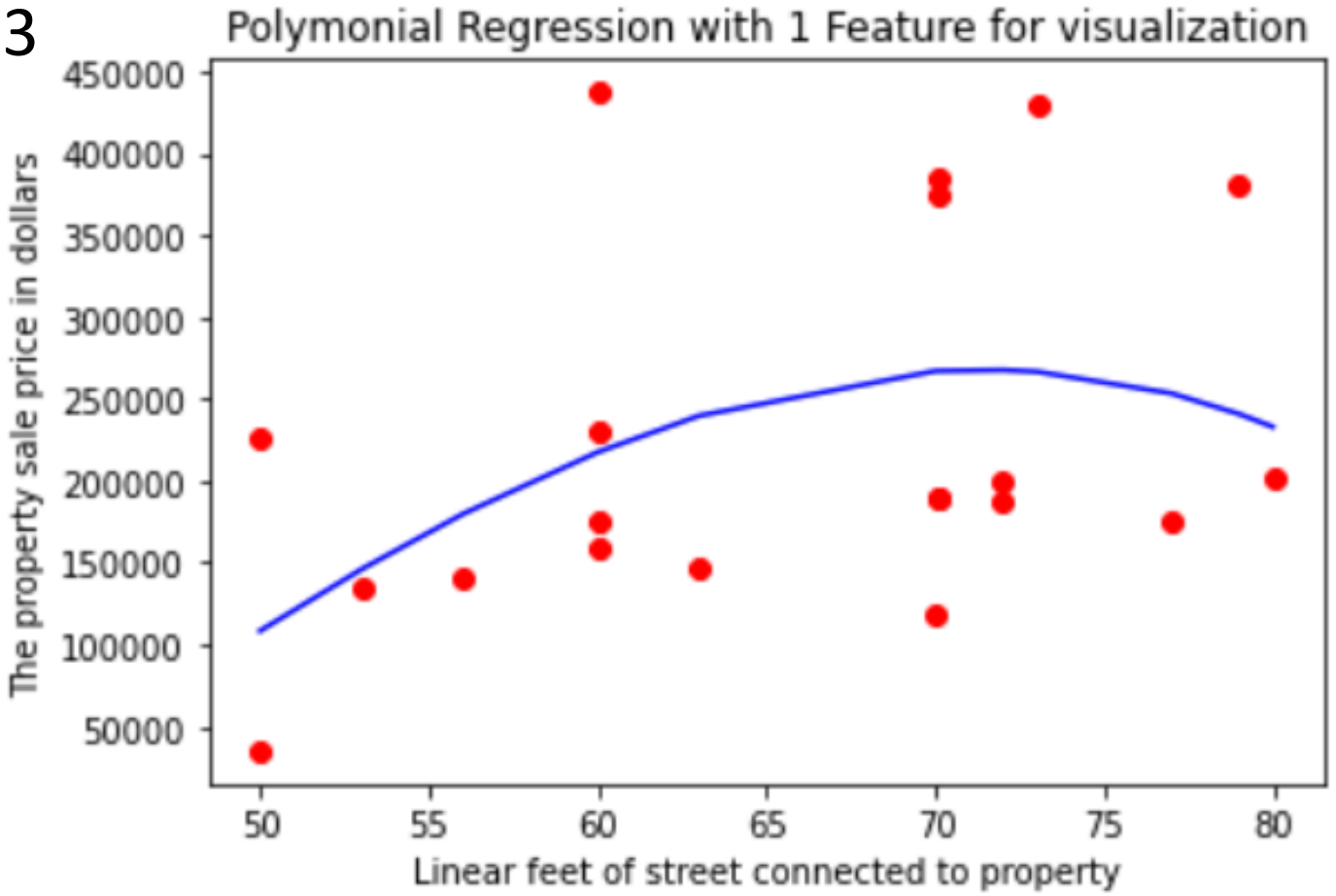
Polynomial Regression

```
#%% Polynomial Regression with 1 Feature for visualization
import random
from sklearn import preprocessing
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
```

```
# Try with polynomial of degree 3
degree = 3

# 1 feature (Linear feet of street connected to property)
# with 20 random samples for visualization
Rp = random.sample(range(0,X_train.shape[0]),20)
ypv_train = y_train[Rp]
Xpv_train = X_train[Rp,2]
Xpv_train = Xpv_train.reshape(-1, 1)

# Standardizing features by removing the mean and scaling to unit variance
scaler = preprocessing.StandardScaler()
### Polynomial Regression - using one feature with 20 samples
polyreg_scaled = make_pipeline(PolynomialFeatures(degree),scaler,LinearRegression())
polyreg_scaled.fit(Xpv_train,ypv_train)
```

Polynomial Regression

- Polynomial Degree: 3

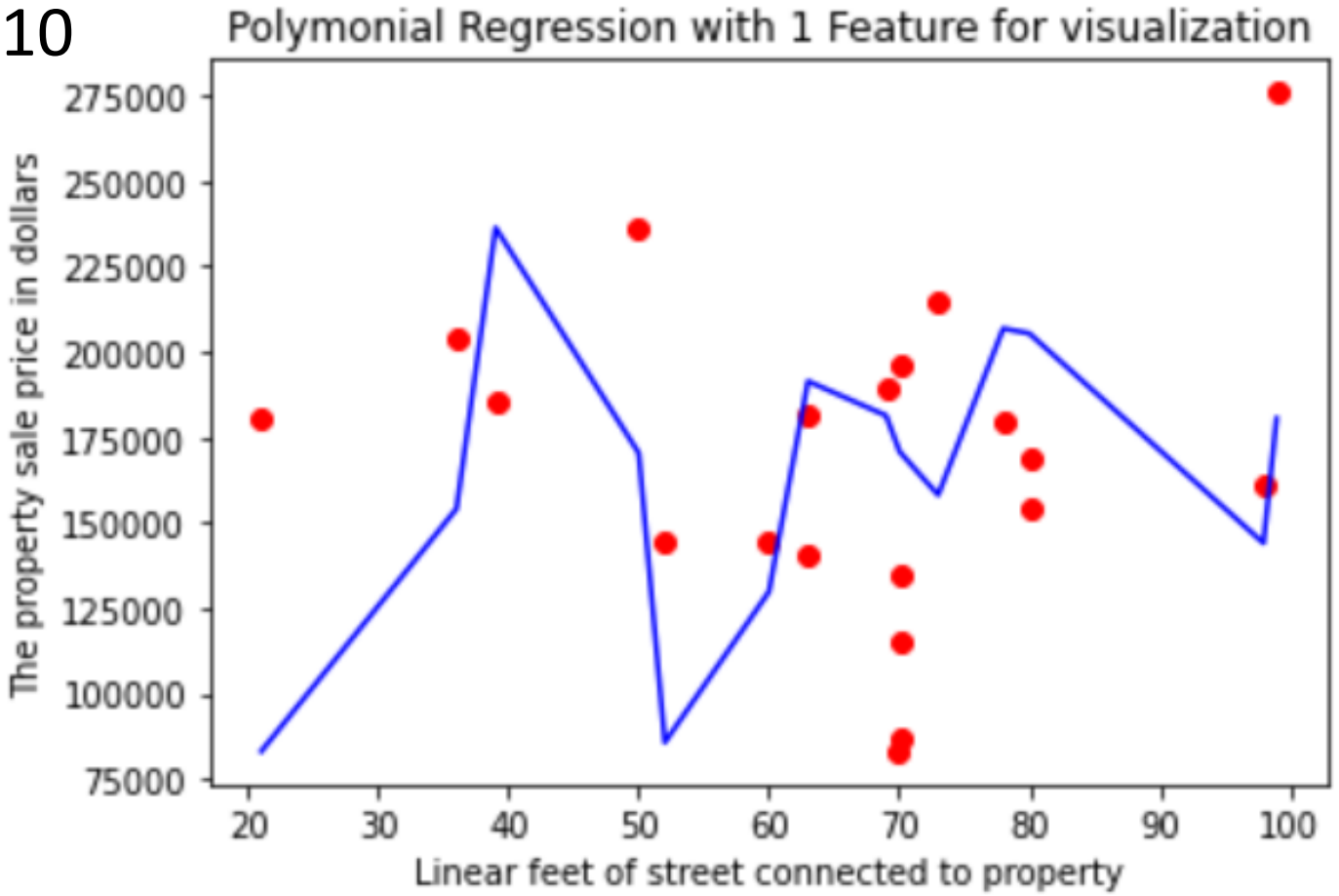


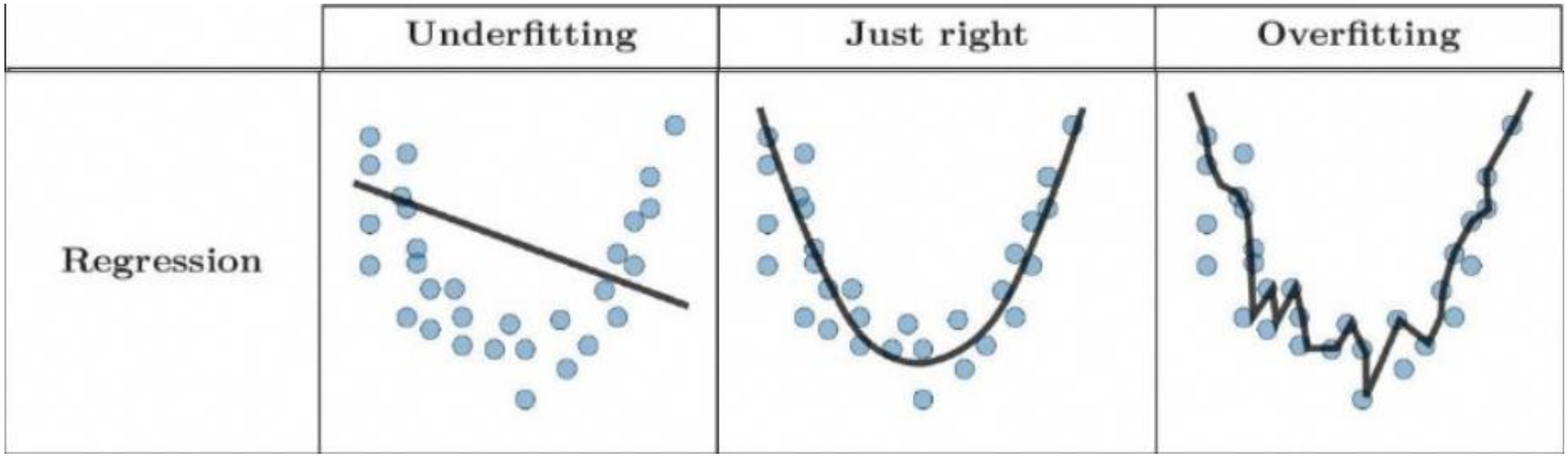
Polynomial Regression

```
# Try with a different polynomial degree (overfitting)
degree = 10

# 1 feature (Linear feet of street connected to property)
# with 20 random samples for visualization
Rp = random.sample(range(0,X_train.shape[0]),20)
ypv_train = y_train[Rp]
Xpv_train = X_train[Rp,2]
Xpv_train = Xpv_train.reshape(-1, 1)

# Standardizing features by removing the mean and scaling to unit variance
scaler = preprocessing.StandardScaler()
### Polynomial Regression - using one feature with 20 samples
polyreg_scaled = make_pipeline(PolynomialFeatures(degree),scaler,LinearRegression())
polyreg_scaled.fit(Xpv_train,ypv_train)
```

Polynomial Regression

- Polynomial Degree: 10
- **Overfitting**




Underfitting
Just right
Overfitting
Regression