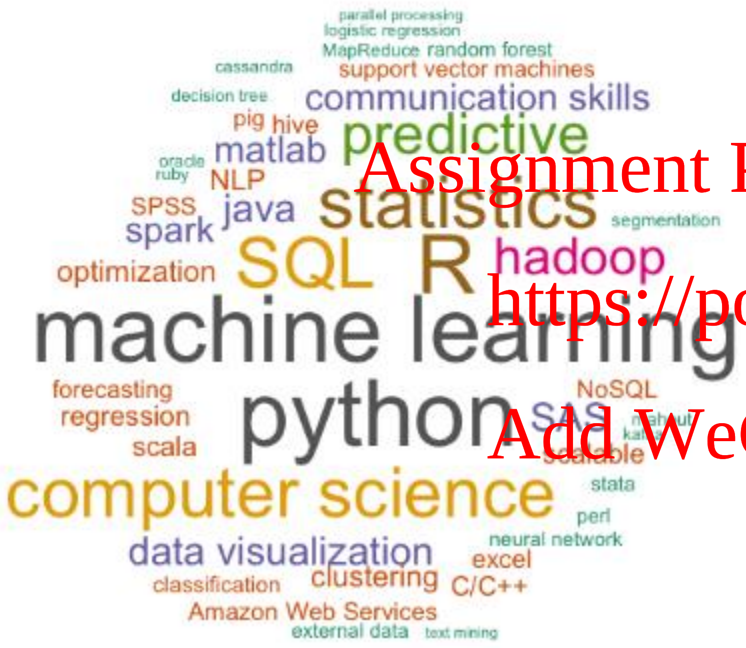
Regression - Concepts (Part 1)



Contents

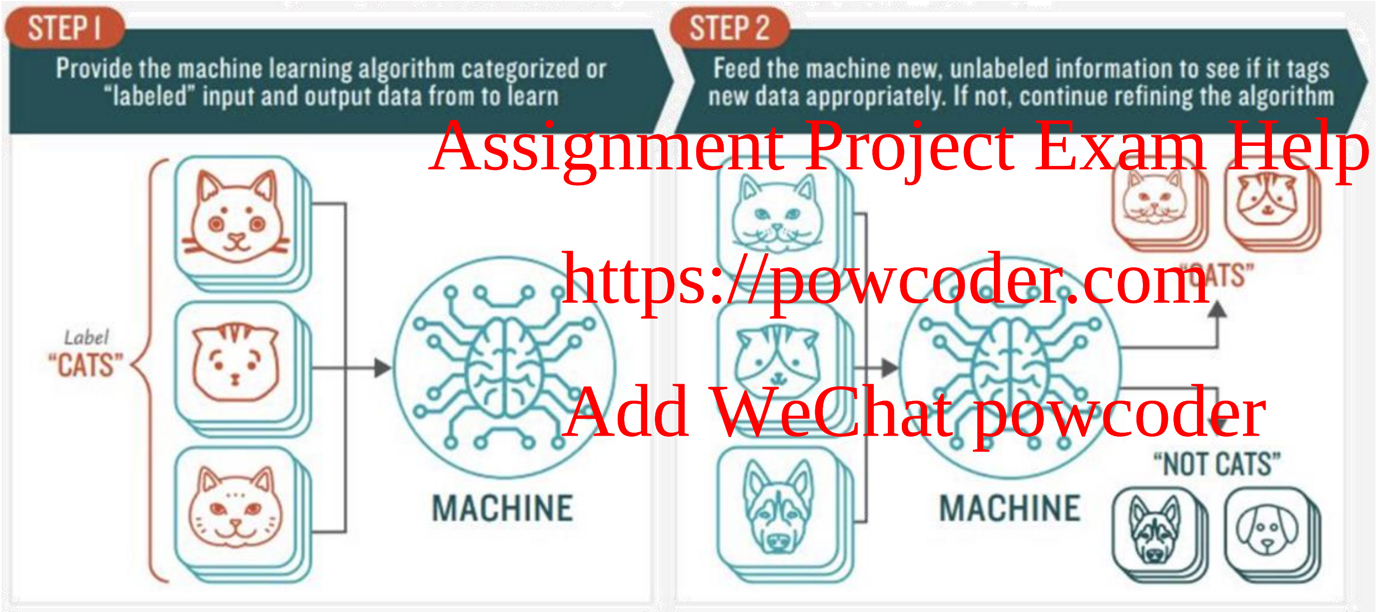
- Supervised Learning
- Linear Regression
- Polynomial Regression
- Regularized Regression
- Time Series

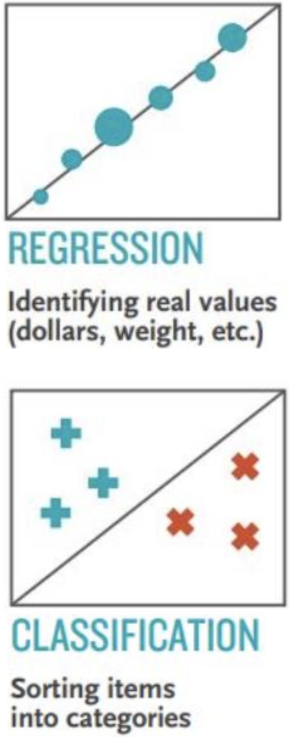
Supervised Learning

Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from **labeled training data** consisting of a set of training examples. Each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).

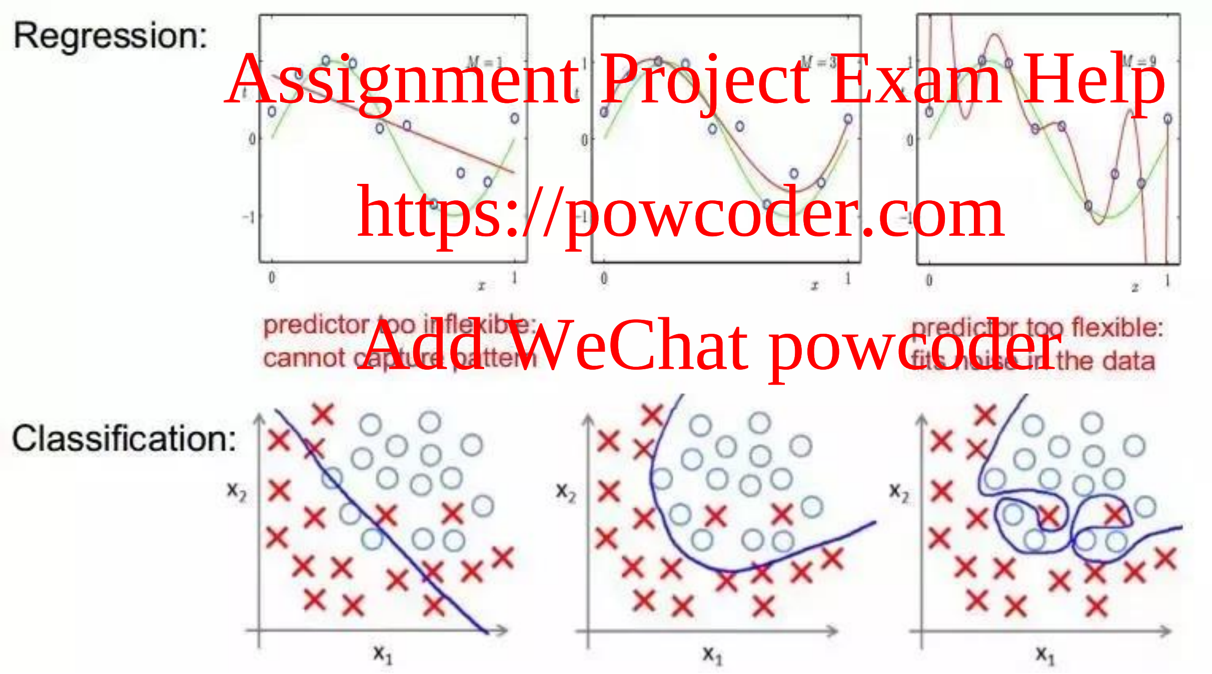
Supervised learning can be used in regression problems and classification problems

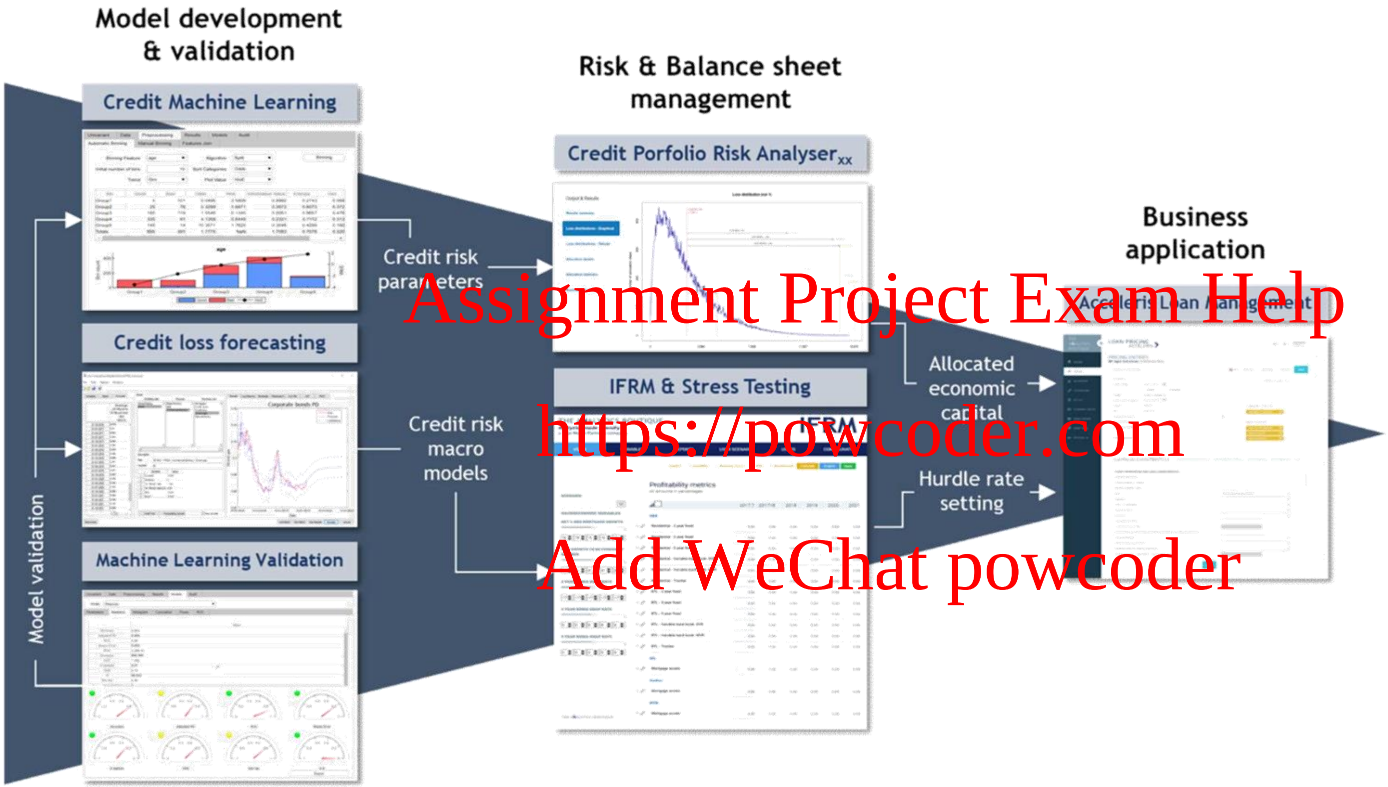




Supervised learning algorithms are provided with ***historical data*** *and asked to* ***find the relationship*** *that has the* ***best predictive power***

Regression predicts along a continuous set of possible outcomes while classification finds the category of the highest probability among a number of finite categories

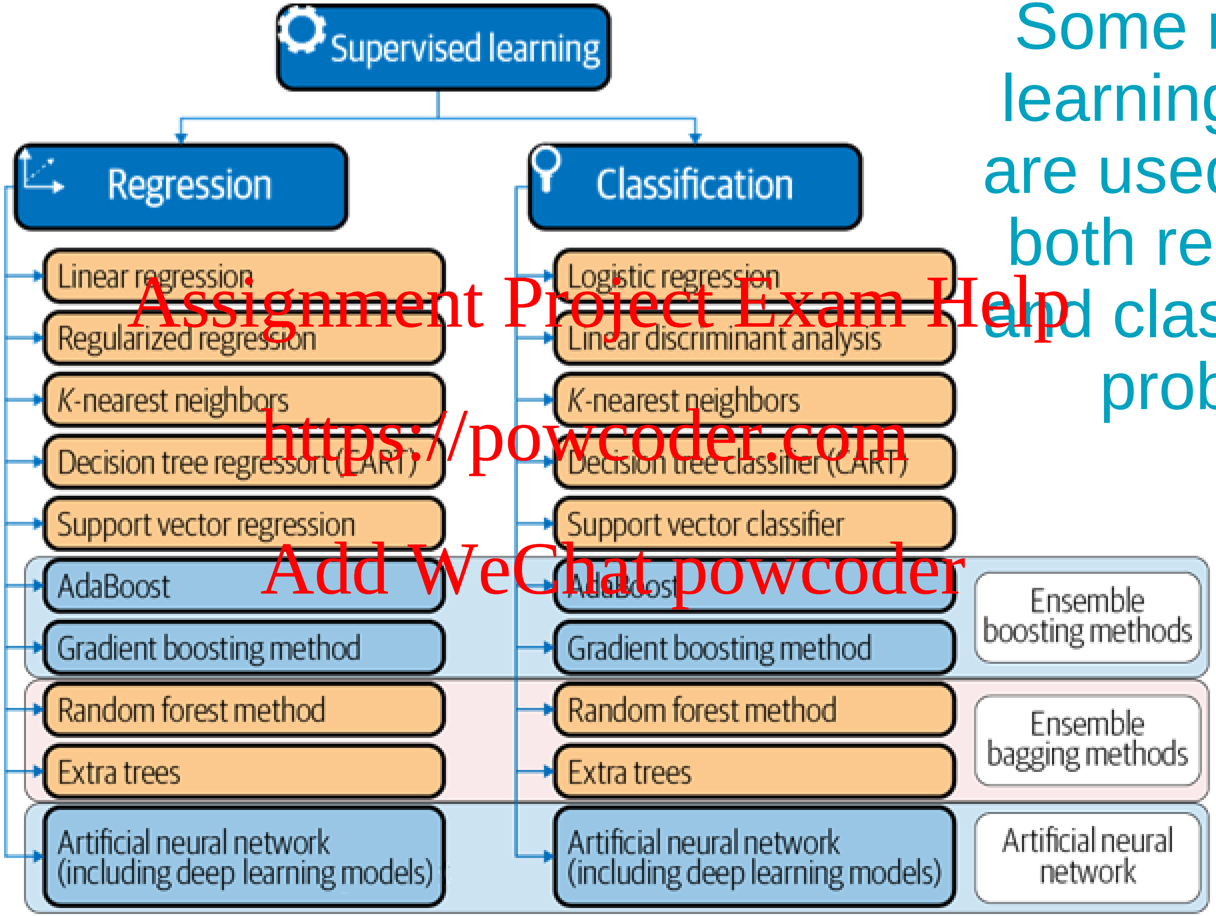




In the context of ***finance***, ***supervised learning*** *models represent one of the* ***most-used*** *class of machine learning models.*

Many algorithms that are widely applied in algorithmic trading rely on supervised learning models because they can be ***efficiently trained***, *they are relatively* ***robust to noisy financial data***, *and they have strong links to the theory of finance.*

Some machine learning models are used to solve both regression and classification problems

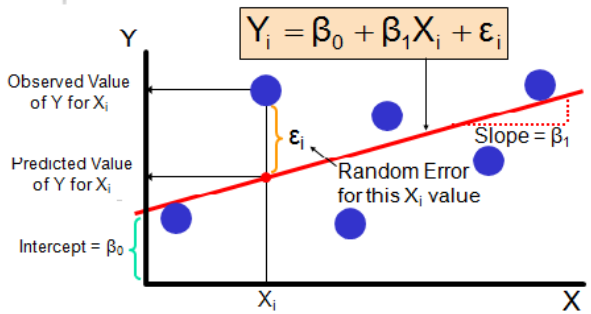


Linear Regression

Linear Regression

Linear regression is a linear model that assumes a linear relationship between the input variables (x) and the single output variable (y). The goal of linear regression is to train a linear model to predict a new y given a previously unseen x with as little error as possible.

Hyperplane

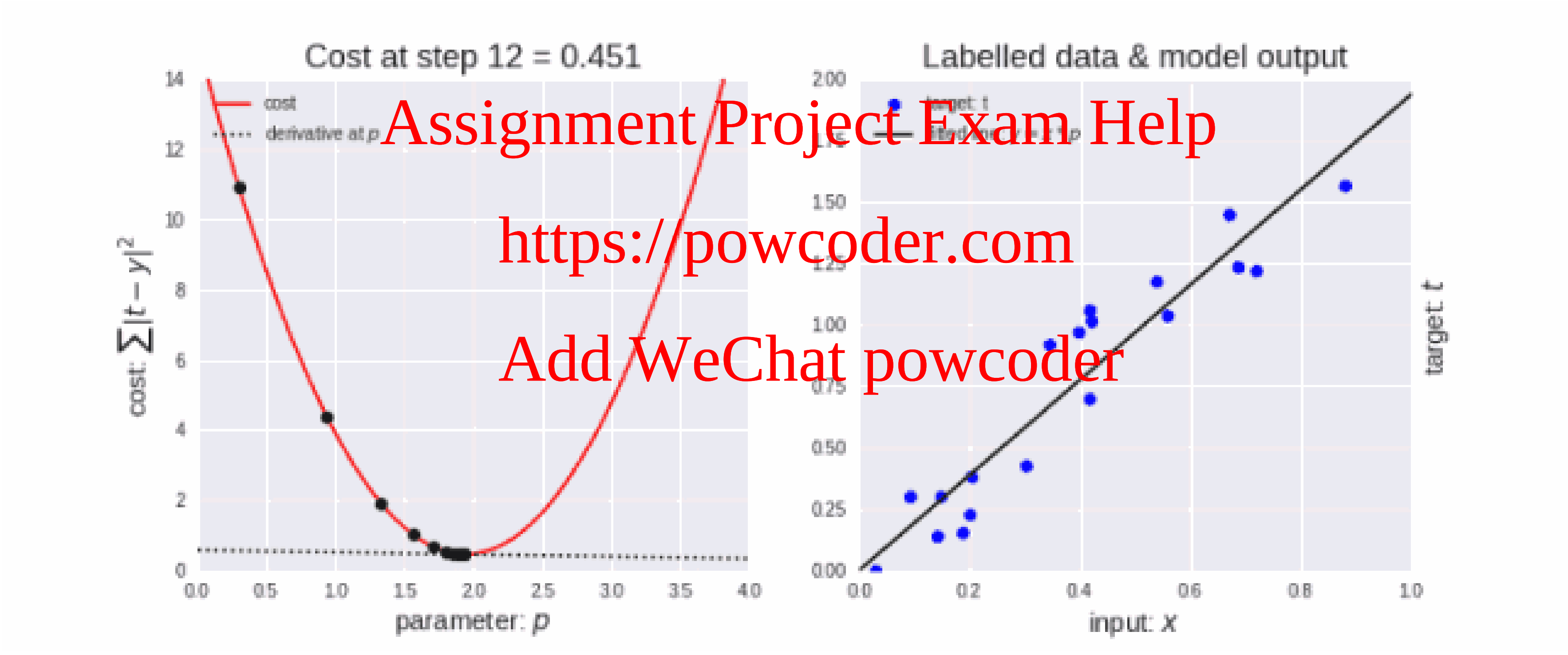


- A linear function $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_i\, x_i$
- β_0 represents the intercept with the y-axis
- $\beta_1 \cdots \beta_i$ are the coefficients of the regression
- The coefficients are estimated by minimizing the sum of the squared deviations between the observed y and the predicted y, **Residual Sum of Squares** (**RSS**)

$$RSS = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{m} \beta_j\, x_{ij}\right)^2$$

  - Residues refer exclusively to the differences between dependent variables and estimations from linear regression

RSS is a parabolic function and the best hyperplane occurs at the bottom of the parabola



Different measures of error are used with linear regression

Score	Formula	Remarks
Mean absolute error	$MAE = \frac{1}{n}\sum_{i=1}^{n}\lvert predicted_i - actual_i\rvert$	Average of absolute errors of all the data points.
Mean squared error	$MSE = \frac{1}{n}\sum_{i=1}^{n}(predicted_i - actual_i)^2 = \frac{RSS}{n}$	Average of the squares of the errors of all the data points. A good practice is to keep the MSE low and the R^2 score high.
Median absolute error	$MedAE = median(\lvert predicted_i - actual_i\rvert)$	The median of all the errors. Robust to outliers.
Explain variance score	$ExpVar = 1 - \frac{Variance\{actual_i - predicted_i\}}{Variance\{actual_i\}}$	Measures how well the model can account for the variation in the dataset. A good practice is to keep the MSE low and the R^2 score high.
R^2 score (coefficient of determination)	$R^2 = 1 - \frac{\sum_{i=1}^{n}(predicted_i - actual_i)^2}{\sum_{i=1}^{n}(predicted_i - \frac{1}{n}\sum_{i=1}^{n} actual_i)^2}$	Measures how well the unknown sample will be predicted by the model. A score near 1 means that the model is able to predict the data very well.

Linear regression with Python (1)

Import the relevant libraries

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, explained_variance_score
```


Load the Boston house prices dataset as an array of features and an array of target

```
features, target = load_boston(return_X_y = True)
features = features[:, 12:13]
```


Show first 5 rows of features

```
features[0:5]
```

```
array([[4.98],
       [9.14],
       [4.03],
       [2.94],
       [5.33]])
```

Show first 5 rows of target

```
target[0:5]
```

```
array([24. , 21.6, 34.7, 33.4, 36.2])
```

Linear regression with Python (2)

Create a linear regressor

```
regression = LinearRegression()
```


Fit the linear regressor and return the model

```
model = regression.fit(features, target)
```


Show the y-intercept of the regression line

```
model.intercept_
```

```
34.55384087938311
```


Show the coefficients of the regression line – the slope
Since only one feature is involved, the array stores only one coefficient

```
model.coef_
```

```
array([-0.95004935])
```

Linear regression with Python (3)

Make predictions using the trained model

```
prediction = model.predict(features)
```


Display the data points and the hyperplane

```
plt.figure(figsize=(12, 8))
```



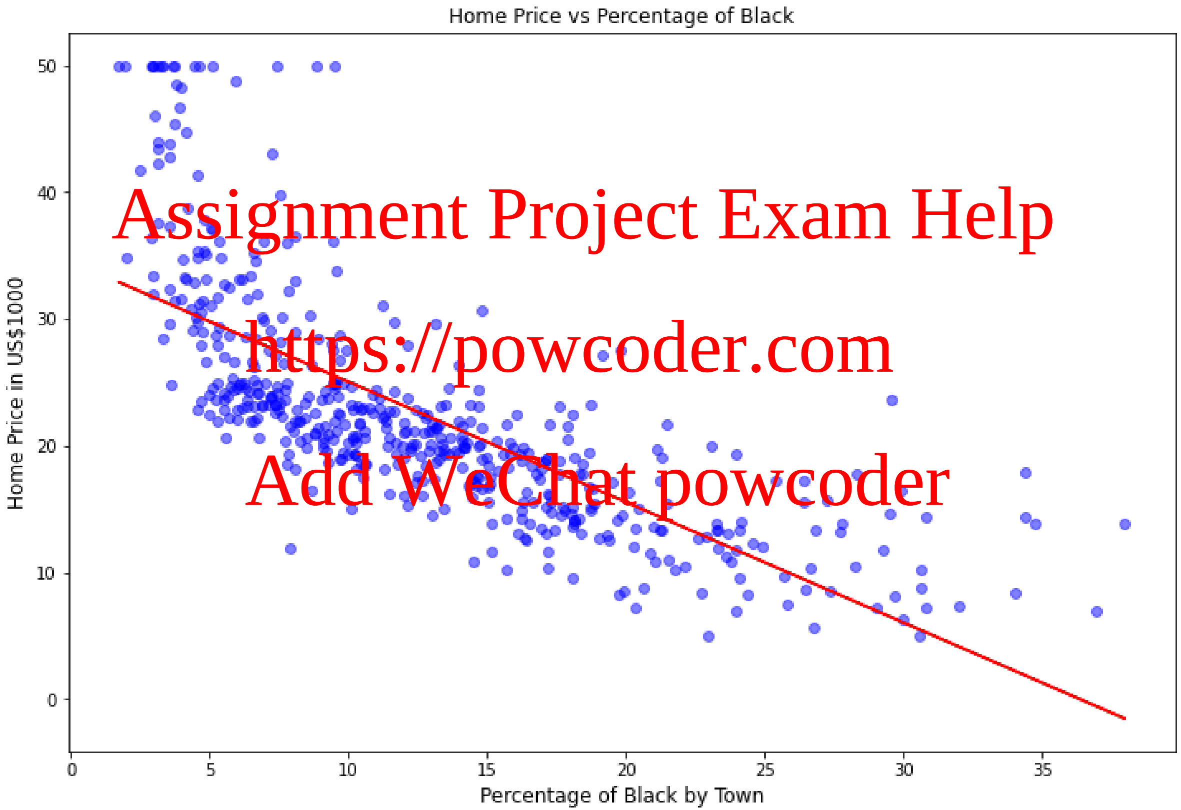
```
plt.scatter(features, target, alpha=0.5, color='blue')
plt.plot(features, prediction, color="red")
```



```
plt.title('Home Price vs Percentage of Black')
plt.xlabel('Percentage of Black by Town', fontsize=12)
plt.ylabel('Home Price in US$1000', fontsize=12)

plt.show()
```

Linear regression with Python (4)



Linear regression with Python (5)

```
# Show the MSE of the hyperplane
print('Mean Squared Error: %.2f'
      % mean_squared_error(target, prediction))
```

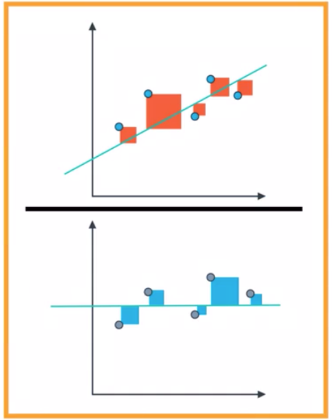
```
Mean Squared Error: 38.48
```



```
# Show the R2 score of the hyperplane
# The score suggests that 54% of the variation in home price is explained by the black %
print('Coefficient of Determination (R^2 Score): %.2f'
      % r2_score(target, prediction))
```

```
Coefficient of Determination (R^2 Score): 0.54
```

R^2 Score

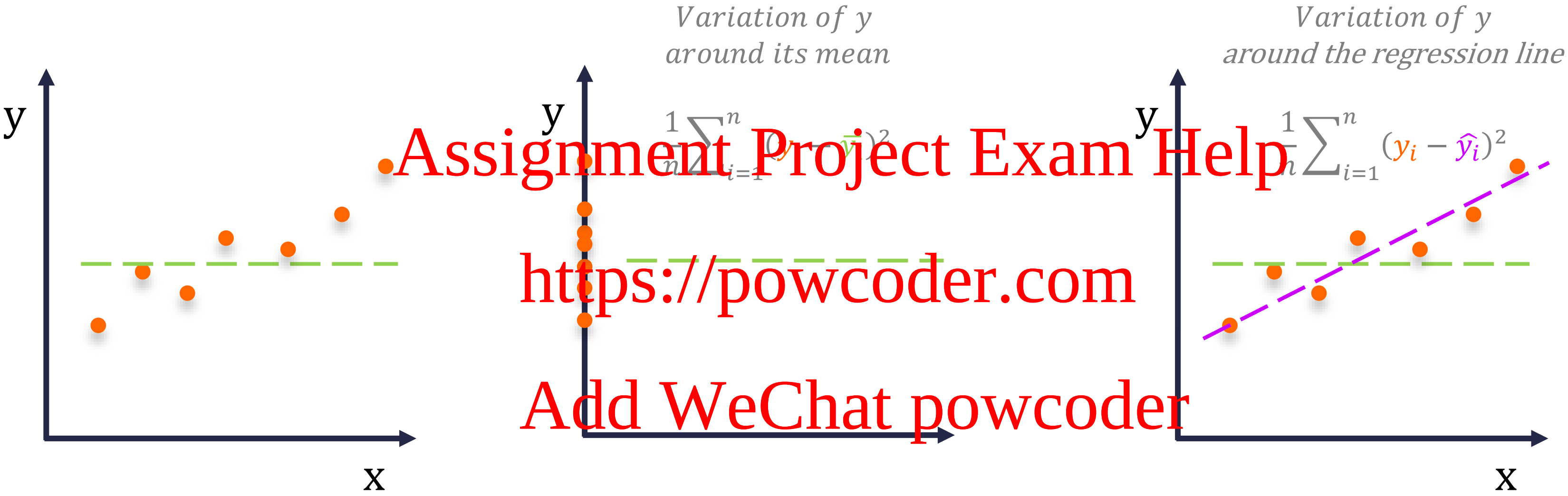
$R^2 = 1 -$ 

- It represents the proportion of the difference or variance in statistical terms for a dependent variable which can be explained by an independent variable or variables

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{\sum_{i=1}^{n}(y_i - \bar{y_i})^2}$$

  - To what extent the variance of one variable explains the variance of the second variable
    - If R^2 of a model is 0.50, approximately half of the observed variation can be explained by the model's inputs
- It determines how well data will fit the regression model

R^2 score compares the change of variation to the dependable variable after the inclusion of an independent variable



x is an independent variable that determines the value of the dependable variable y

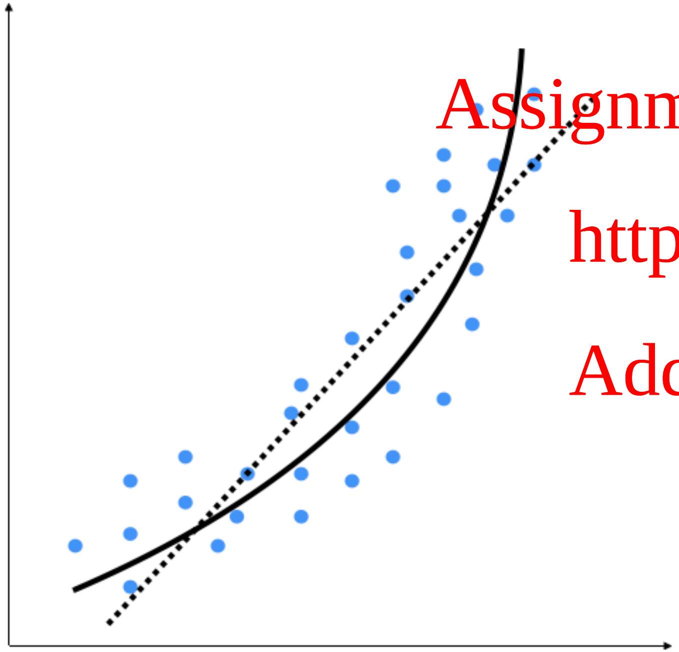
The variation of the dependable variable without considering any dependable variable

$$R^2 = \frac{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2 - \sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

R^2 represents the percentage change of variation with the introduction of variable x

Polynomial Regression

Polynomial regression should be applied where the relationship is curvilinear

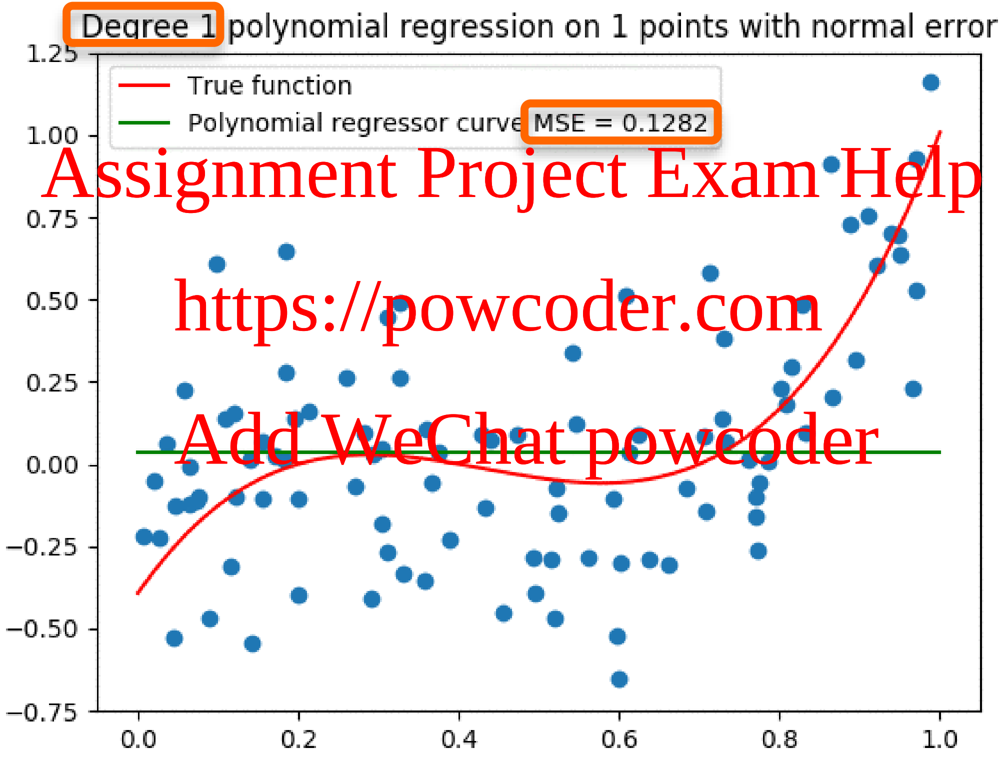


- The linear model (dotted line) is not the best fit to the data points, the polynomial regression model (solid line) provides a better fit
- Polynomial combinations of the same or different features - feature interaction - can return new features that capture the curviness
- A simple & easy way to model curves, without needing to create big non-linear models

Polynomial Regression

With polynomial regression, the prediction model may have independent variables appearing in degrees equal to or greater than two to fit the data with a curved hyperplane. Polynomial regression is usually used when the relationship between variables looks curved.

As the curviness of the model increases, it gets more accurate



Polynomial regression with Python (1)

Import the relevant libraries

```
from sklearn.datasets import load_boston
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```


Load the Boston house prices dataset as an array of features and an array of target

```
features, target = load_boston(return_X_y = True)
features = features[:, 12:13]
```


Construct new features via feature interaction

```
interaction = PolynomialFeatures(degree=2,
                                 include_bias=False,
                                 interaction_only=False)
features_interaction = interaction.fit_transform(features)
```

Polynomial regression with Python (2)

Create a linear regressor

```
regression = LinearRegression()
```


Fit the linear regressor and return the model

```
model = regression.fit(features_interaction, target)
```


Make predictions using the new model

```
prediction = model.predict(features_interaction)
```


Show the MSE and R-squared score of the hyperplane

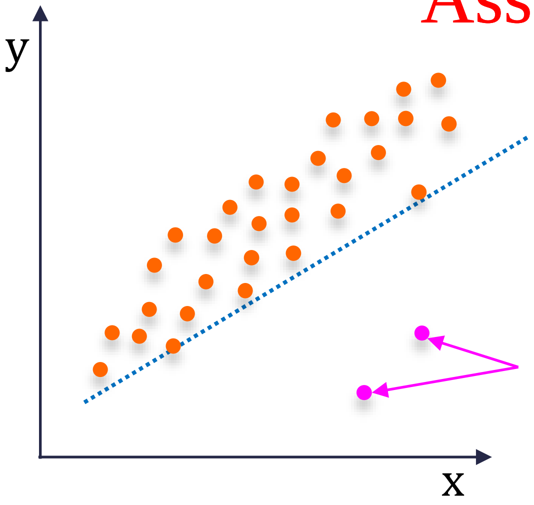
```
print('Mean Squared Error: %.2f'
      % mean_squared_error(target, prediction))

print('Coefficient of Determination (R^2 Score): %.2f'
      % r2_score(target, prediction))
```

```
Mean Squared Error: 22866274.36
Coefficient of Determination (R^2 Score): 0.84
```

Regularized Regression

Linear regression attempts to remain unbiased but leads to greater variance and suboptimal prediction accuracy



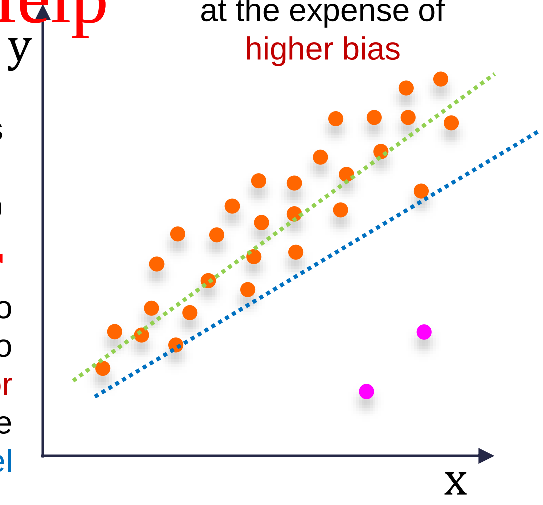
Linear regression attempts to remain unbiased by taking into considering every single data point

Linear regression is sensitive to outliers

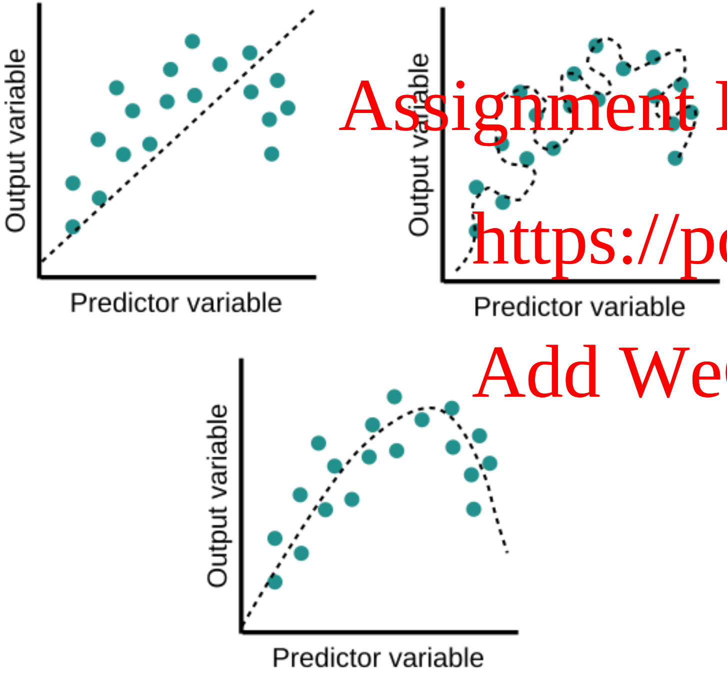
the model is suboptimal (i.e. less accurate)

the outliers tend to contribute a lot to the overall error and disrupt the entire model

the optimal model has less variance at the expense of higher bias



Linear regression with poorly selected coefficients may result in overfitting that hurts model robustness



- **Higher degree** model and **large coefficients** increase the variance significantly (while keeping bias low) leading to overfitting
- The optimal model needs to be robust meaning predicting well on training data as well as testing data (data the model has not seen during training)
- The overfitting during training is what prevents the model from being robust

Regularization

- To avoid overfitting and increase model robustness, model training needs to be **regularized** (constrained)
  - Shrink the coefficients (weights) of model features
  - Get rid of high degree polynomial features

- Linear models are typically regularized by constraining the **weights**

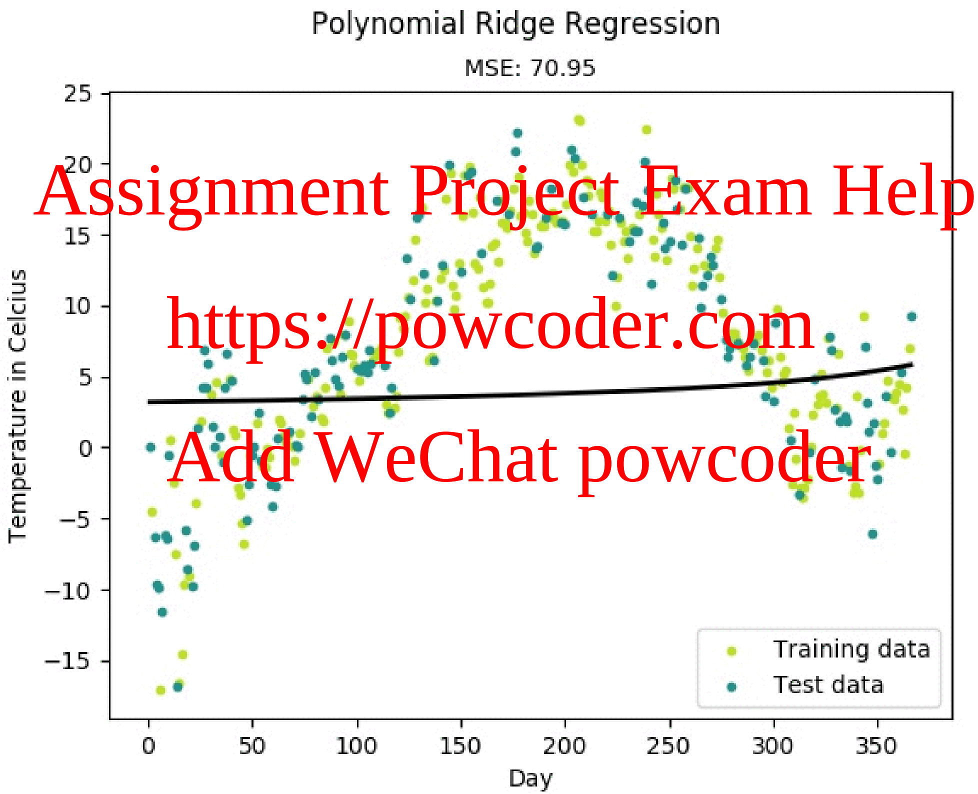
- Polynomial models are regularized by reducing the **polynomial degrees**
- A penalty term together with a regularization hyperparameter (λ) can regulate the size of the bias term in the model

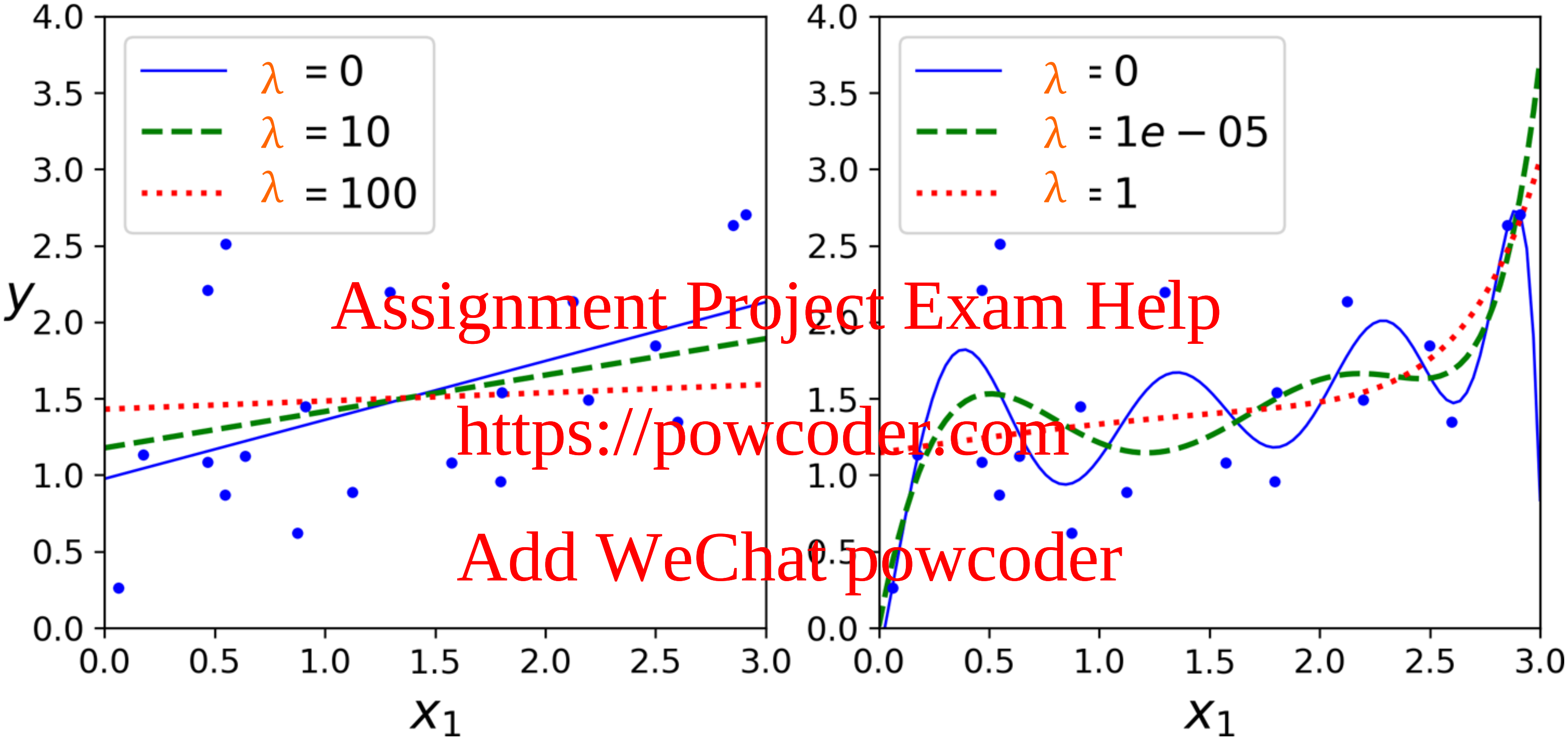
L2 Regularization / Ridge Regression

$$Cost\ Function = RSS + \lambda \cdot \sum_{j=1}^{p} \beta_j^2$$

Ridge regression adds a factor of the sum of the square of coefficients to the (RSS) cost function for linear regression. It shrinks the coefficients and helps reduce variance by introducing bias. Ridge regression can shrink the coefficients <u>asymptotically close to 0</u>, and therefore cannot reduce the number of variables. When sample sizes are relatively small, it can improve predictions made from new data by making the predictions less sensitive to the training data.

L2 regularization constrains the determination of coefficients





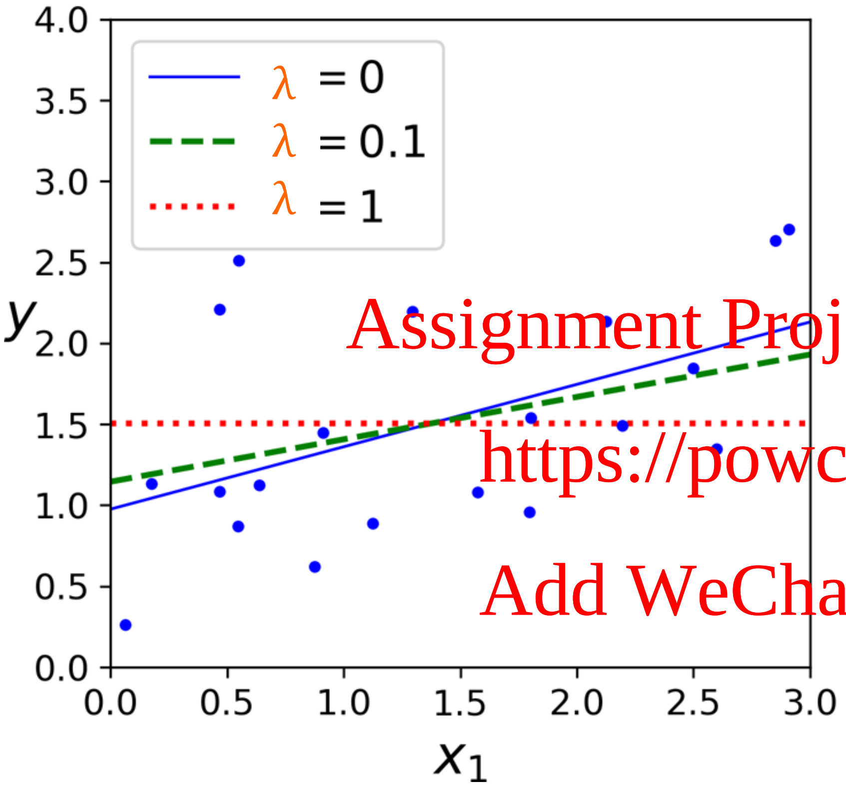
A plain Ridge model leading to linear predictions

Polynomial Regression with L2/ridge regularization flattens predictions and reduces variance while increases bias

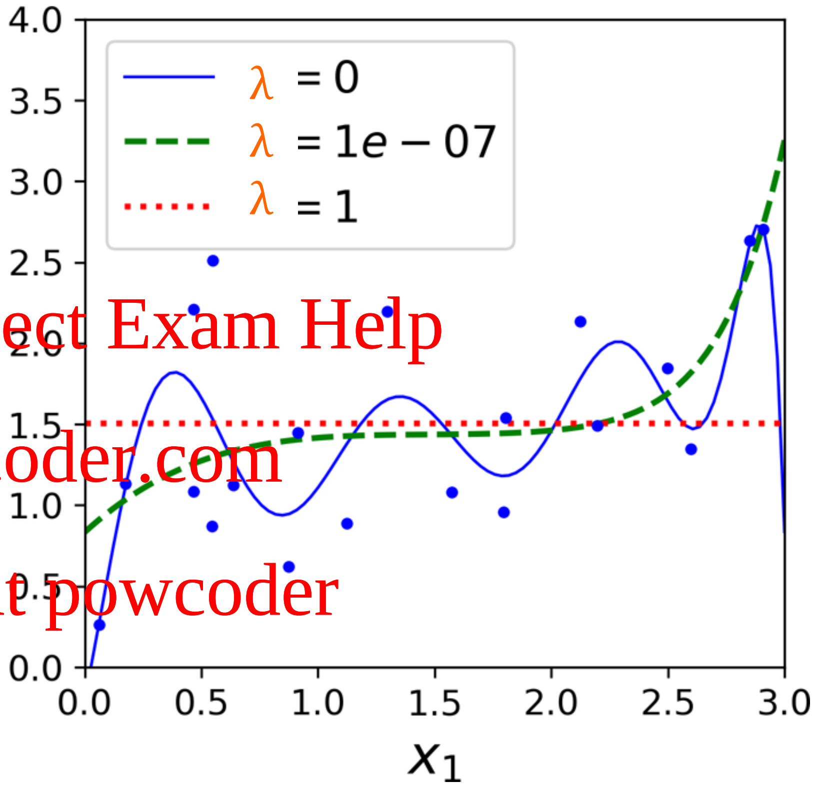
L1 Regularization / Lasso Regression

$$Cost\ Function = RSS + \lambda \cdot \sum_{j=1}^{p} |\beta_j|$$

Lasso regression adds a factor of the sum of the absolute value of coefficients to the (RSS) cost function for linear regression. The larger the value of the regularization parameter λ, the more coefficients are shrunk towards zero, even all the way to 0. Effectively, it makes predictions with new data less sensitive to the training dataset. Lasso regression not only helps in reducing overfitting, but also can help in feature selection.



A plain Lasso model leading to linear predictions



L1/Lasso regression tends to eliminate the weights of the least important features – effectively performing feature selection

Lasso regression with Python (1)

Import the relevant libraries

```
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
```


Load the Boston house prices dataset as an array of features and an array of target


```
boston = load_boston()
features = boston.data
target = boston.target
```


Standardize features

```
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
```

Lasso regression with Python (2)

```
# Create lasso regression
# Set the regularization parameter using the alpha value
regression = Lasso(alpha=0.5)
```



```
# Fit the lasso regressor
model = regression.fit(features_standardized, target)
```



```
# View coefficients
model.coef_
```



```
array([-0.11526463,  0.        , -0.        ,  0.39707879, -0.        ,
        2.97425861, -0.        , -0.17056942, -0.        , -0.        ,
       -1.59844856,  0.54313871, -3.66614361])
```

Lasso regression with Python (2)

Setting alpha to a high value will see literally none of the features being used

```
regression_a10 = Lasso(alpha=10)
model_a10 = regression_a10.fit(features_standardized, target)
model_a10.coef_
```



```
array([-0.,  0., -0.,  0., -0.,  0., -0.,  0., -0., -0., -0.,  0., -0.])
```


The practical benefit of this effect is that it means that we could include 100 features in our feature matrix and then, through adjusting lasso's α hyperparameter, produce a model that uses only 10 (for instance) of the most important features. This lets us reduce variance while improving the interpretability of our model (since fewer features is easier to explain).

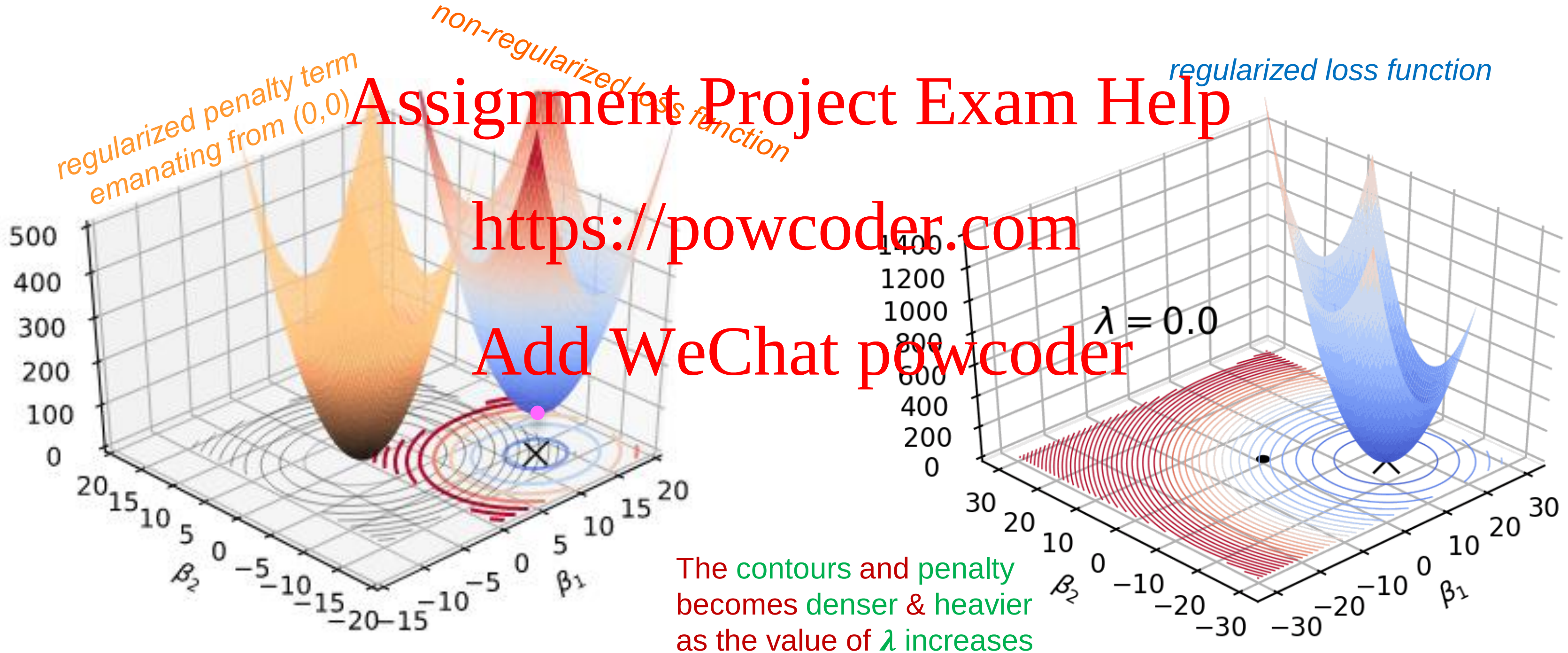
Combinations of β_1 and β_2 that produce the same error can be represented using colored contours in a 2D space

Gradient descent represented as a 3D surface in the space capturing β_1, β_2 and the error $J(\beta)$

Turning gradient descent from 3D representation to 2D representation by eliminating the error dimension

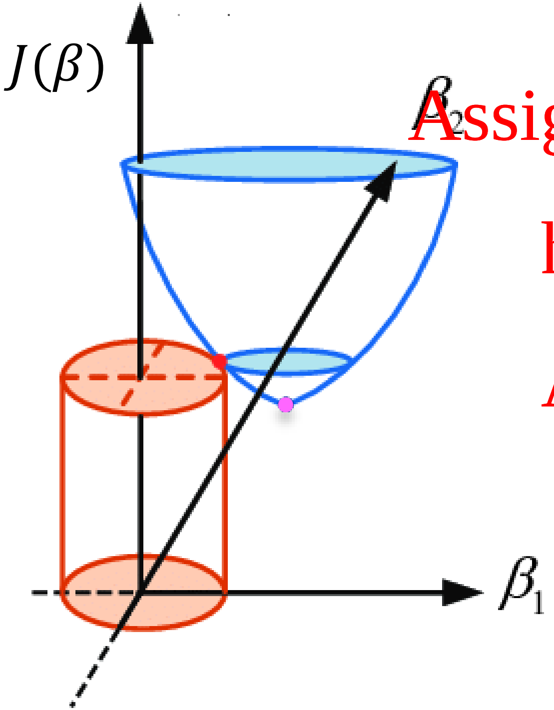
Gradient descent represented as contours in the 2D space capturing β_1 and β_2

The (increase of the) penalty term shifts the non-regularized function "bowl" upwards and its minimum towards the origin

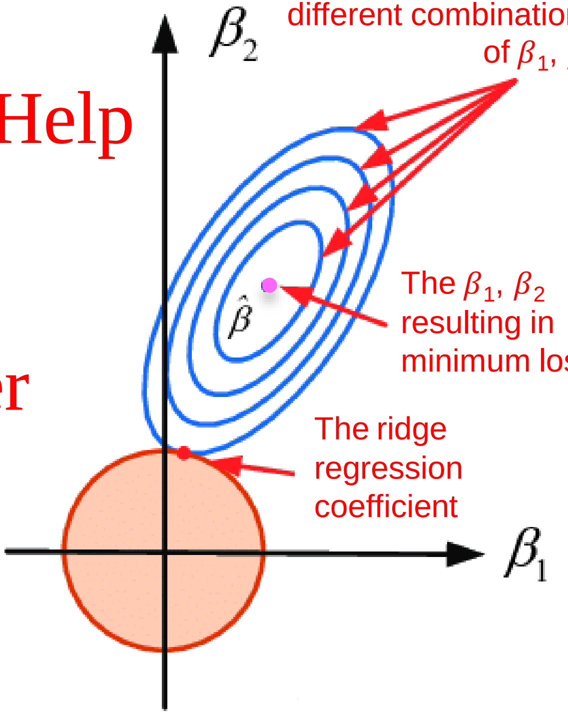


The contours and penalty becomes denser & heavier as the value of λ increases

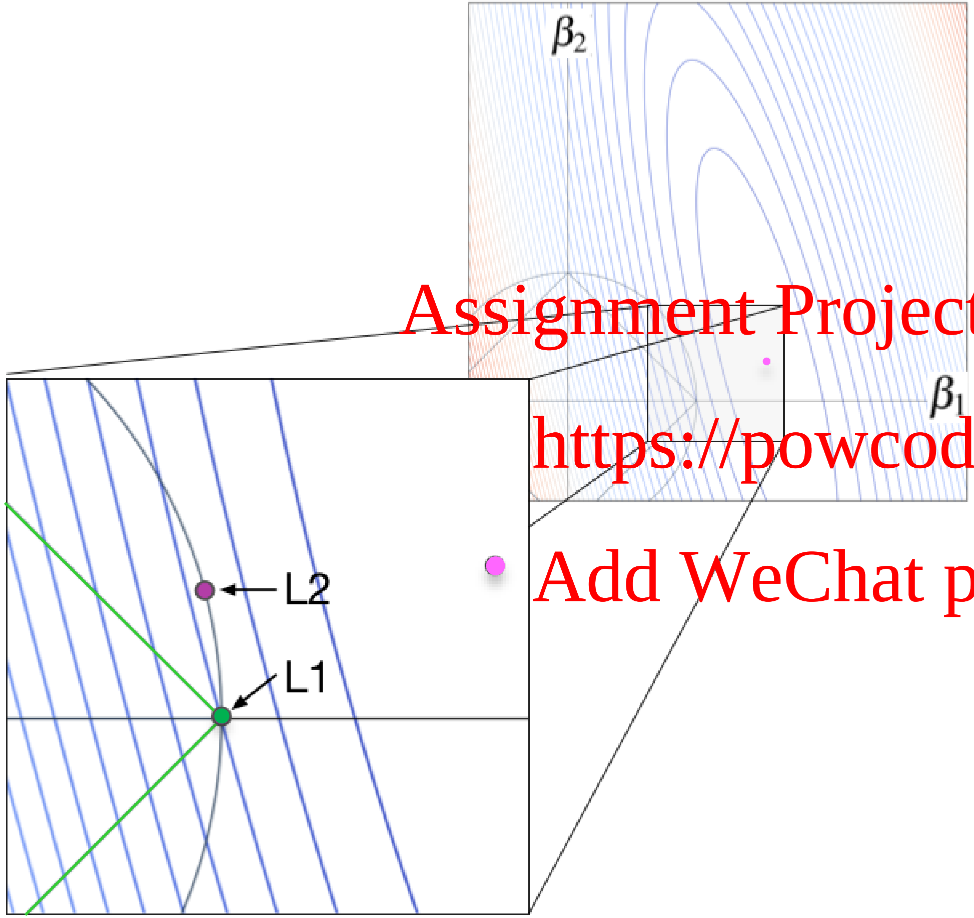
Two forces are at work when determining the minimum loss for the regularized model



- Let's use a simplified situation when a model uses hard constraints
- The hard constraint is $\beta_1^2 + \beta_2^2 < 1$ is represented by the cross section of the cylinder
- The height of the cylinder corresponds to λ
- The minimum loss is the point when the non-regularized loss "bowl" meets with the cylinder
- The larger the value of λ, the less contribution to the loss from the bowl thus pushing it up

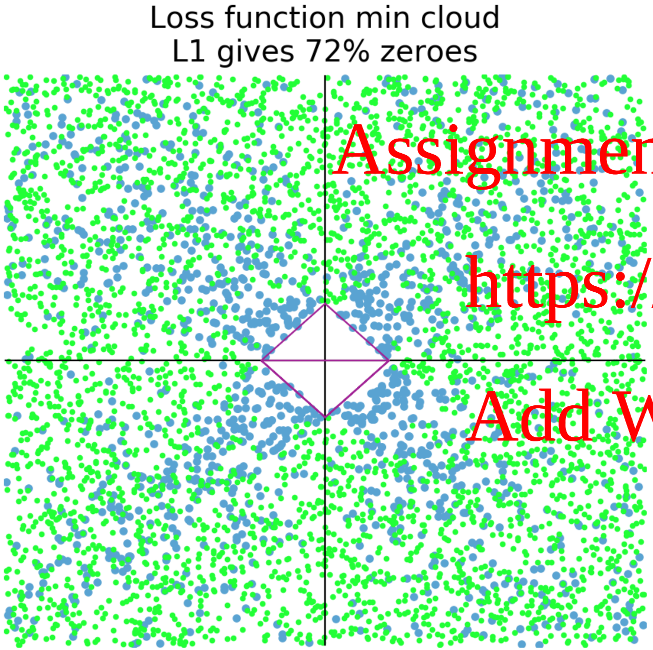


L1 encourages zero coefficients but not L2

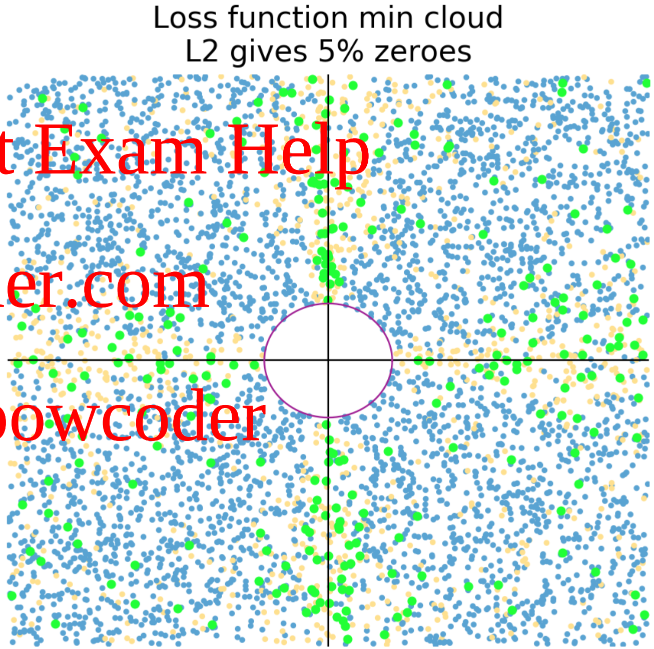


- L1 regularization encourages zero coefficients but not L2 regularization
- For L1, the optimal point is at the diamond tip, any movement away from this point increases the loss
- For L2, the optimal point is non-zero and not on the axis but can be very close to the axis

L1 regularization encourages zero coefficients



green: a loss function minimum
with **zero** regularized coefficient



blue: a loss function minimum
with **non-zero** regularized coefficient

To strike a balance between Ridge and Lasso's penalty functions, Elastic Net can be used

- Both L2/ridge and L1/lasso regression can penalize large or complex models by including coefficient values in the loss function that is to be minimized

- As a very general rule of thumb, L2/ridge regularization often produces slightly better predictions than L1/lasso regularization, but L1/lasso regularization produces more interpretable models

- To strike a balance between L2/ridge and L1/lasso's penalty functions, Elastic Net can be used, which is simply a regression model with both penalties included

Elastic Net

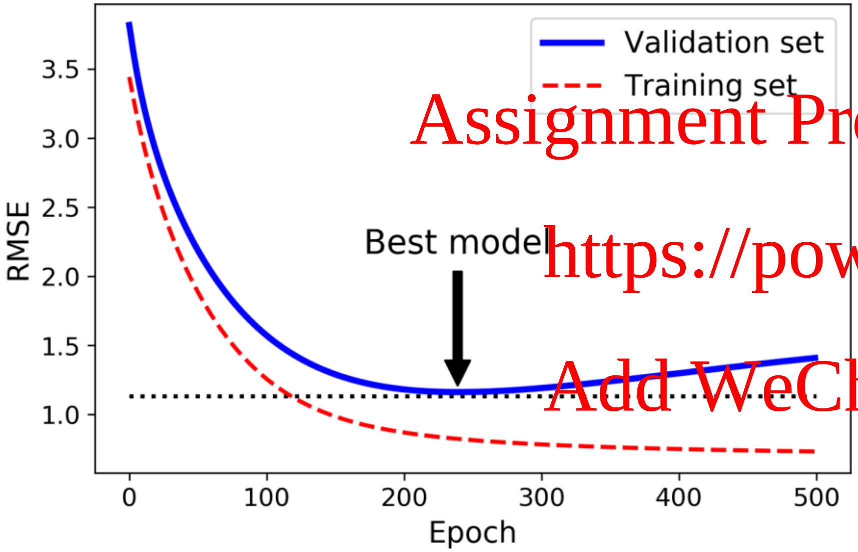
$$Cost\ Function = RSS + \lambda \cdot \left(\frac{(1-\alpha)}{2} \sum_{j=1}^{p} \beta_j^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right)$$

Elastic Nets add regularization terms to the model, which are a combination of both L1 and L2 regularization. In addition to setting and choosing a λ value, an elastic net also allows us to tune the α parameter, where $\alpha = 0$ corresponds to ridge and $\alpha = 1$ to lasso. Therefore, we can choose an α value between 0 and 1 to optimize the elastic net. Effectively, this will shrink some coefficients and set some to 0 for sparse selection.

So when to use plain Linear Regression (i.e., without any regularization), Ridge, Lasso, or Elastic Net?

- It is almost always preferable to have at least a little bit of regularization, so generally plain Linear Regression should be avoided
- **Ridge** is a good default
- If it is suspected that only a few features are useful, Lasso or Elastic Net would be preferred because they tend to reduce the useless features' weights down to zero
- In general, Elastic Net is preferred over Lasso because Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated

A very different way to regularize is to stop training as soon as the validation error reaches a minimum – early stopping



- As the epochs go by, the algorithm learns and its prediction error on the training set goes down, along with its prediction error on the testing dataset
- After a while though, the testing error stops decreasing and starts to go back up
- This indicates that the model has started to overfit the training dataset
- With **early stopping** training can be stopped as soon as the testing error reaches the minimum

Time Series

Other than historical stock price, there are other features that are generally useful for stock price prediction

Correlated Assets

- An organization depends on and interacts with many external factors, including its competitors, clients, the global economy, the geopolitical situation, fiscal and monetary policies, access to capital, and so on
- Hence, its stock price may be correlated not only with the stock price of other companies but also with other assets such as commodities, FX, broad-based indices, or even fixed income securities

Technical Indicators

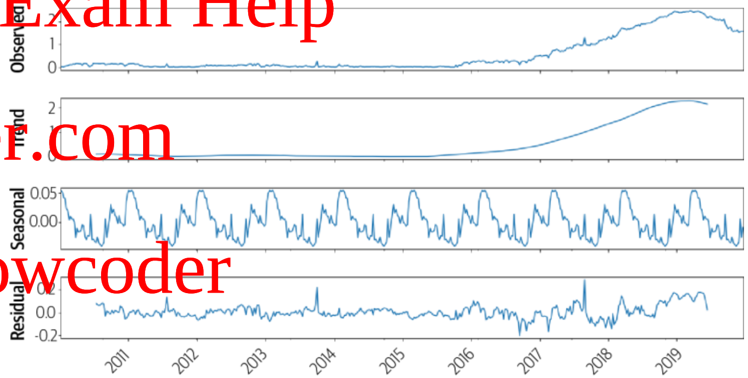
- A lot of investors follow technical indicators
- Moving average, exponential moving average, and momentum are the most popular indicators

Fundamental Analysis

- Two primary data sources to glean features that can be used in fundamental analysis
- Performance reports
  - Annual and quarterly reports of companies can be used to extract or determine key metrics, such as ROE (Return on Equity) and P/E (Price-to-Earnings)
- News
  - News can indicate upcoming events that can potentially move the stock price in a certain direction.

Time Series

A time series is a sequence of numbers that are ordered by a time index. It can be broken down into trend component (deterministic or stochastic), seasonal component (representing seasonal or cyclical variation), and the residual component.



A time series can be broken down into trend, seasonal, and residual component

Trend Component

- A consistent directional movement
- Either deterministic or stochastic
- The former provides an underlying rationale for the trend
- The latter is a random feature of a series
- Trends often appear in financial series, and many trading models use sophisticated trend identification algorithms

Seasonal Component

- Many time series contain seasonal variation
- This is particularly true in series representing business sales or climate levels
- In quantitative finance we often see seasonal variation, particularly in series related to holiday seasons or annual temperature variation (such as natural gas)

Residual Component

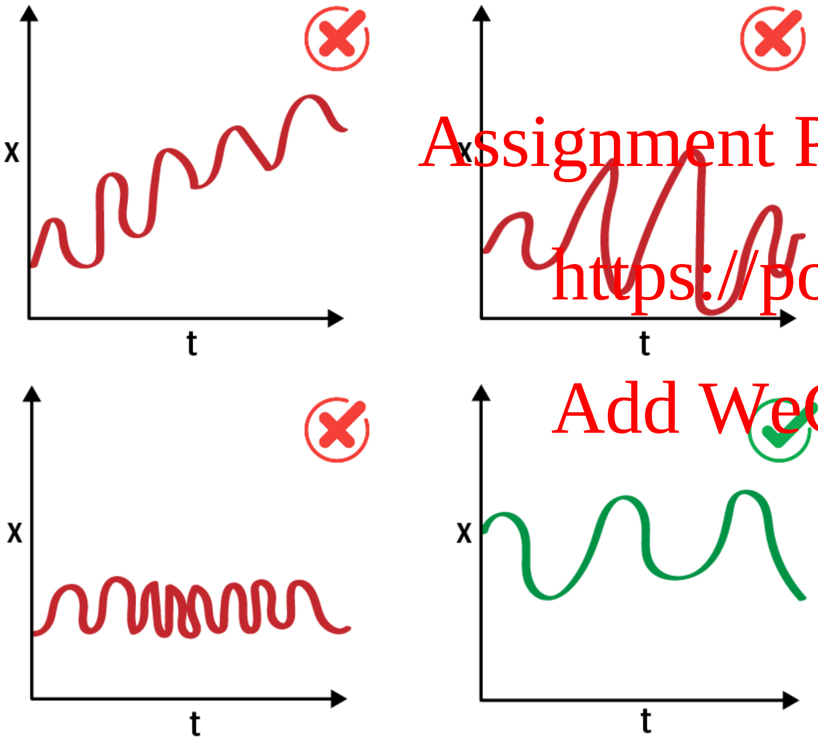
- The residual component is what is left over when the seasonal and trend components have been subtracted from the data

Autocorrelation

$$y_t = c + \emptyset_1 y_{t-1} + \emptyset_2 y_{t-2} + \cdots + \emptyset_p y_{t-p} + \epsilon_t$$

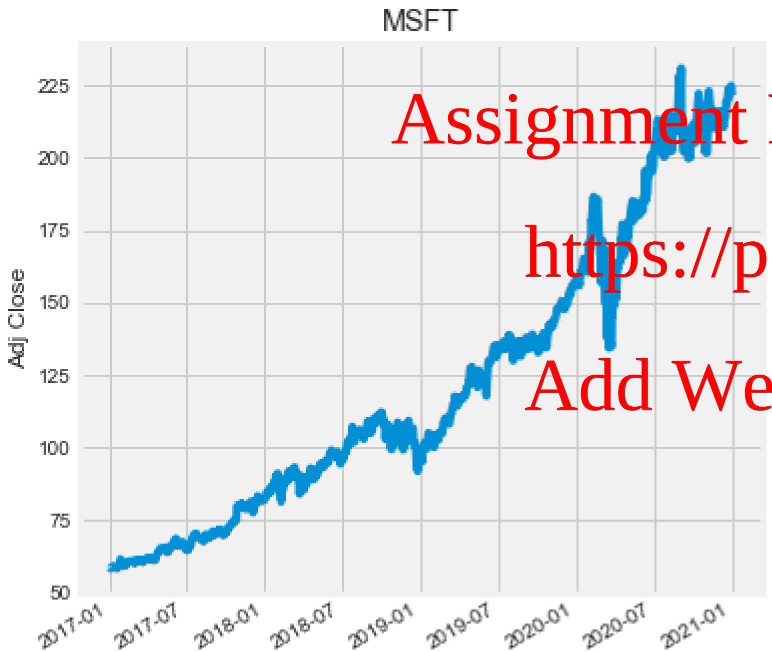
There are many situations in which consecutive elements of a time series exhibit correlation – the behavior of sequential points in the series affect each other in a dependent manner. Autocorrelation is the similarity between observations as a function of the time lag between them. Such relationships can be modeled using an autoregression model. The term autoregression indicates that it is a regression of the variable against itself.

Most statistical models require the time series to be stationary to make effective and precise predictions



- A time series is said to be stationary if its statistical properties (mean, variance, covariance) do not change over time
- Non-stationary series, as a rule, are unpredictable and cannot be modeled or forecasted
- A white noise (ϵ) series is considered stationary and has a mean of zero
- A stationary series must have a constant mean, variance & covariance
- Non-stationary series are converted to a stationary series before using time series forecast models

Generally speaking, forecasting models cannot be applied directly to pricing time series data



- On the left is the stock price data for Microsoft over a period of 4 years
- The time series is obvious non-stationary so cannot be directly used with forecasting models
  - Mean and variance are not constant
- The question therefore is how to transform this non-stationary time series to a stationary time series that we can make prediction with night

Differencing

$$y'_t = y_t - y_{t-1}$$

- Using non-stationary time series data in financial models produces unreliable and spurious results and leads to poor understanding and forecasting
- Differencing computes the difference of consecutive terms in a time series
- It can help stabilise the mean of a time series by removing changes in the level and therefore eliminating (or reducing) trend and seasonality
- The disadvantage of differencing is that it loses one observation each time the difference is taken

Time series data need to be reorganized before using supervised learning models

Time Step	Value
1	10
2	11
3	18
4	15
5	20

from 5 to 4 data points

Time Step	X	Y
1		10
2	10	11
3	11	18
4	18	15
5	15	20
6	20	

In Python, the main function to help transform time series data into a supervised learning problem is the shift() function

Traditional time series statistical models work primarily with linear functions and do not tolerate corrupt or missing data

- The traditional time series models such as ARIMA are well understood and effective on many problems

- However, these traditional methods also suffer from several limitations

- They are linear functions, or simple transformations of linear functions, and they require manually diagnosed parameters, such as time dependence, and do not perform well with corrupt or missing data

- RNN has gained increasing attention in recent years
- These methods can identify structure and patterns such as nonlinearity, can seamlessly model problems with multiple input variables, and are relatively robust to missing data

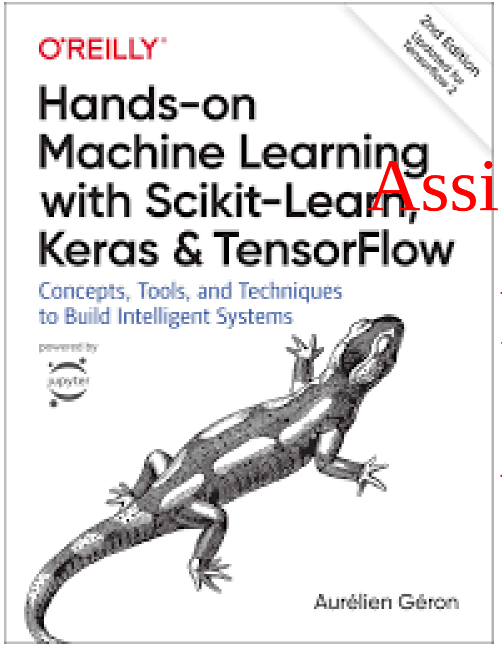
Conclusion

Regression should be deployed with regularization

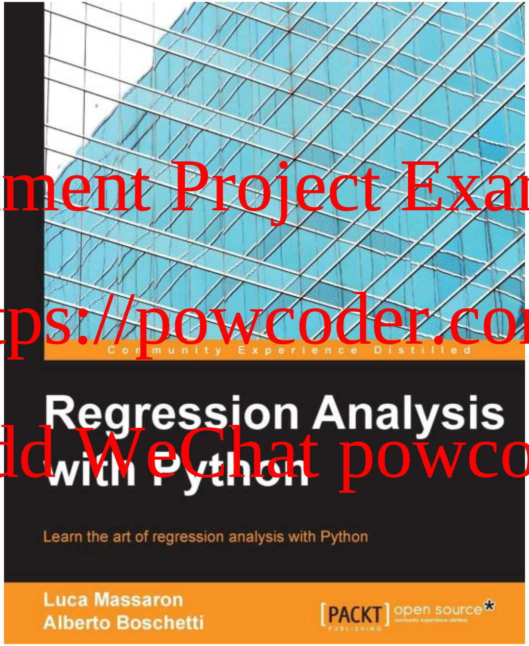
	Property	Description
1	Feature Data Types	Numeric data. Variable encoding is therefore necessary for categorical data. Normalised data is advised. For time series data, differencing is performed to make time series stationary and a number of differences (referred to as the order of integration) may be performed depending on the lag time. Train and test split should be done based on sequential sample.
2	Target Data Types	Numeric data.
3	Key Principles	Linear regression uses linear function to estimate the data points. Polynomial regression relies on feature interaction to different degrees to derive a polynomial function to estimate the data points. Regularization constrains the coefficients of the model functions using a loss function together with a penalty term. Lass regression may induce some coefficients to zero, effectively performing feature selection. Elastic Net uses both the ridge and lasso regression penalty terms.
4	Hyperparameters	Regularization parameter is used to specify the influence of the penalty term. When the value is very large, the regularization effect dominates the sum of squared loss function and the coefficients shift towards or to zero. When the regularization parameter tends toward zero, the regularized loss function tends towards the ordinary least sum of squared and coefficients exhibit big oscillations.
5	Data Assumptions	Non-parametric – no assumption about data distribution. All data are used.
6	Performance	N/A
7	Accuracy	Ridge regularization performs better.
8	Explainability	N/A

References

References



“Hands-On Machine Learning with Scikit-Learn and TensorFlow”, Aurelien Geron, O'Reilly Media, Inc., 2017



"Regression Analysis with Python”, Luca Massaron & Alberto Boschetti, Packt Publishing, 2016

"Machine Learning and Data Science Blueprints for Finance”, Hariom Tatsat, Sahil Puri & Brad Lookabaugh, O'Reilly Media, Inc., 2020

References

- "Linear Regression" (https://en.wikipedia.org/wiki/Linear_regression)
- "Introduction to Linear Regression", David M. Lane (http://onlinestatbook.com/2/regression/intro.html)
- "A Deep Dive into Regularization", Divakar Kapil, 2018 (https://medium.com/uwaterloo-voice/a-deep-dive-into-regularization-eec8ab648bce)
- "From Linear Regression to Ridge Regression, the Lasso, and the Elastic Net", Robby Sneiderman, 2020 (https://towardsdatascience.com/from-linear-regression-to-ridge-regression-the-lasso-and-the-elastic-net-4eaecaf5f7e6)
- "A Visual Explanation for Regularization of Linear Models", Terence Parr, (https://explained.ai/regularization/index.html)

THANK YOU