

CS 429- Machine Learning Algorithms and Practice

Linear Regression

Amina Asif & Gabriele Pergola

UNIVERSITY OF WARWICK

Slides adapted from Dr. Fayyaz Minhas's lecture notes

A quick recap ...



- Introduction to Machine Learning
- Linear Classification
- SVMs and Kernels
- Tree based Methods
- Model Evaluation and Comparison
- Dimensionality Reduction
  - PCA
  - t-SNE

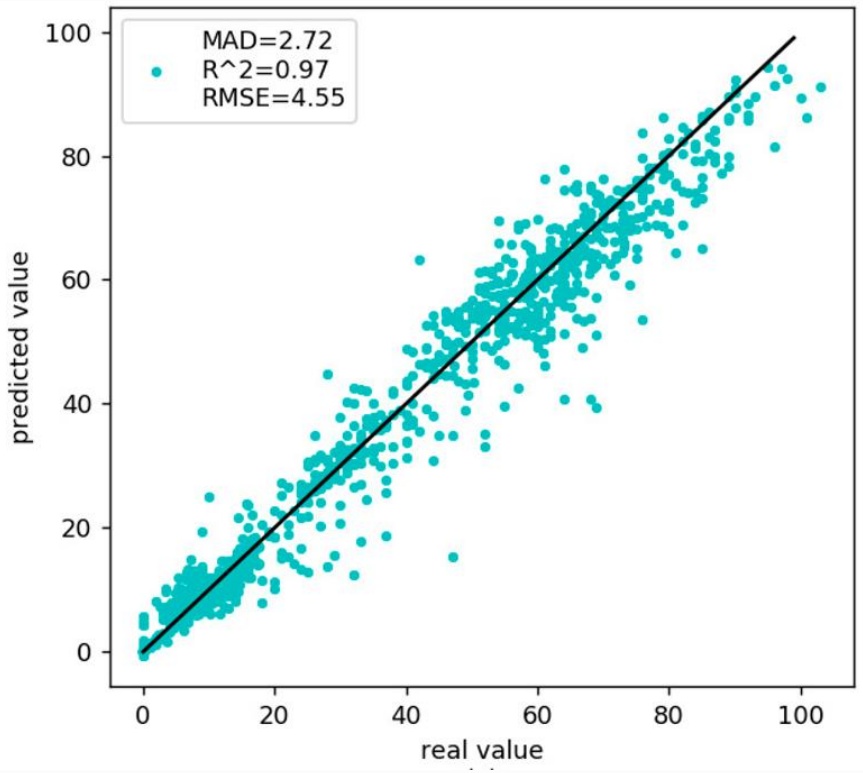
This week onwards…

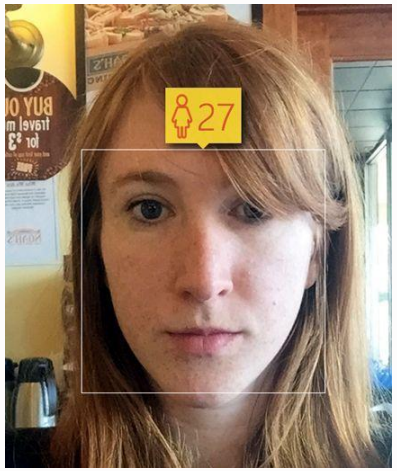


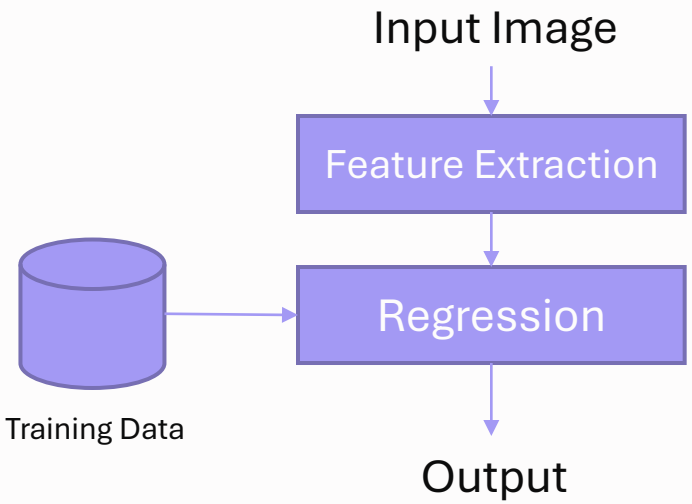
- Regression and other learning frameworks
- Deep Learning
- Coursework Assignment 2 will be released later this week
  - We will discuss it in detail in class

Problems Other than Classification

- How to model if
  - Output is a continuous variable
  - Predict age from an image







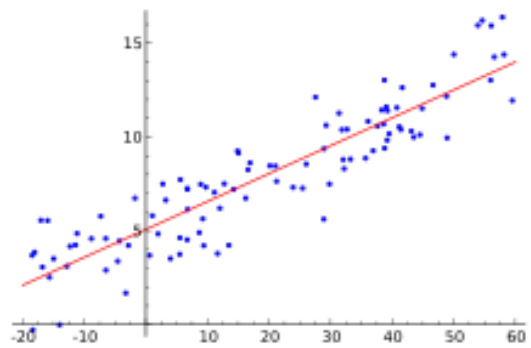
Regression

- Estimate the relationship among variables
  - Dependent variables
  - Independent variables

- Used in
  - prediction and forecasting
  - estimating effects of variables on each other

- Mathematical formulation
  - **Model:** $y = f(\boldsymbol{x}; \boldsymbol{w}) + \boldsymbol{\epsilon}$
    - **For example:** $y = wx + b + \boldsymbol{\epsilon}$

  - The Objective is to estimate the parameters $\boldsymbol{w}$ that produce outputs that are close to the given targets



x	y
1	3.5
2	4.75
3	7.05
4	9.5
…	…

Regression

- **Prediction:** Using observed data to estimate unknown or future outcomes.
- **Inference:** Understanding the relationship between variables and drawing conclusions about a population.

- Example: suppose we want to study whether the number of hours a student studies affects their exam score and we fit a linear regression model

$$y = wx + b + \epsilon$$

$y =$ exam score, $x =$ hours studied, $\epsilon =$ error

- **Prediction**
  - Focuses on estimating the outcome for new or unseen observations
  - How accurately can the model predict exam scores?
  - How well does the model perform on new data?
  - What is the expected prediction error?
  - Can we improve predictive accuracy?

- **Inference**
  - focuses on understanding and explaining relationships between variables
  - Is there a statistically significant relationship between study hours and exam scores?
  - How large is the relationship?
  - Is the regression coefficient significantly different from zero?

REO Framework – A quick recap

Representation

How we express the data and choose the mathematical structure of the model.

What can the model learn?

Evaluation

Measures how well a model performs according to a chosen criterion.

How good is the model?

Optimisation

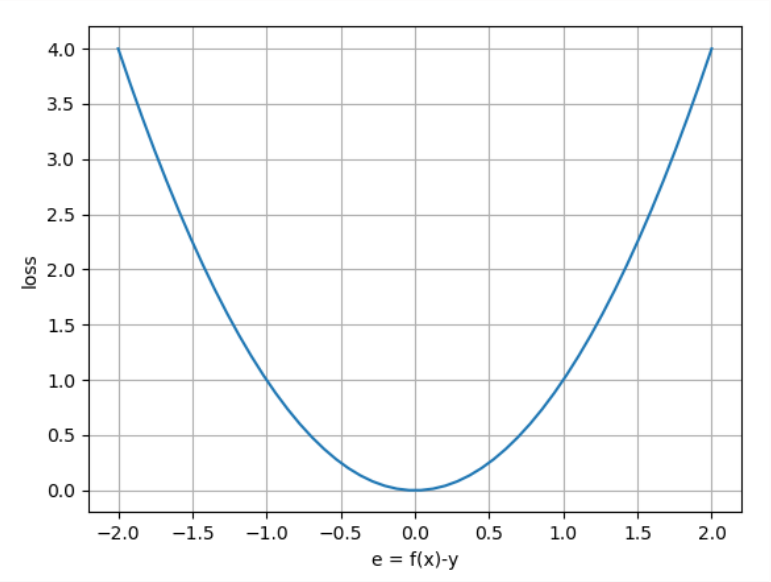
The process of finding model parameters that minimise a loss function or improve an objective.

How does the model learn?

Pedro Domingos. 2012. A few useful things to know about machine learning. Commun. ACM 55, 10 (October 2012), 78–87. https://doi.org/10.1145/2347736.2347755

SRM: Ordinary Lease Squares Regression (OLSR)

- **Representation**
  - $f(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b$
- **Evaluation**
  - Regularization: None
  - Error: Square Loss
  - $l(f(x), y) = (\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y})^2$
- **Optimization Problem**



$$\min_{w} \sum_{i=1}^{N} (\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x_i} - y_i)^2 = \|\boldsymbol{Xw} - \boldsymbol{y}\|^{\boldsymbol{2}} = (\boldsymbol{Xw} - \boldsymbol{y})^{\boldsymbol{T}}(\boldsymbol{Xw} - \boldsymbol{y})$$

$$\boldsymbol{X}_{(N\times d)}, \boldsymbol{y}_{(N\times 1)} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Taking derivative wrt w and setting to zero gives :

$$\boldsymbol{w} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-\boldsymbol{1}}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}$$

Linear Regression: Simple(st) Example

Example

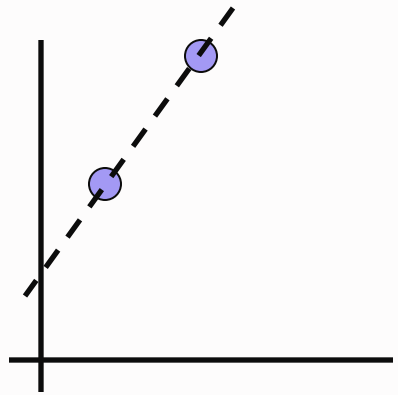
$$\boldsymbol{X} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} 3.5 \\ 4.75 \end{bmatrix}$$

Thus: $\boldsymbol{w} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} = \begin{bmatrix} 1.25 \\ 2.25 \end{bmatrix}$

Now

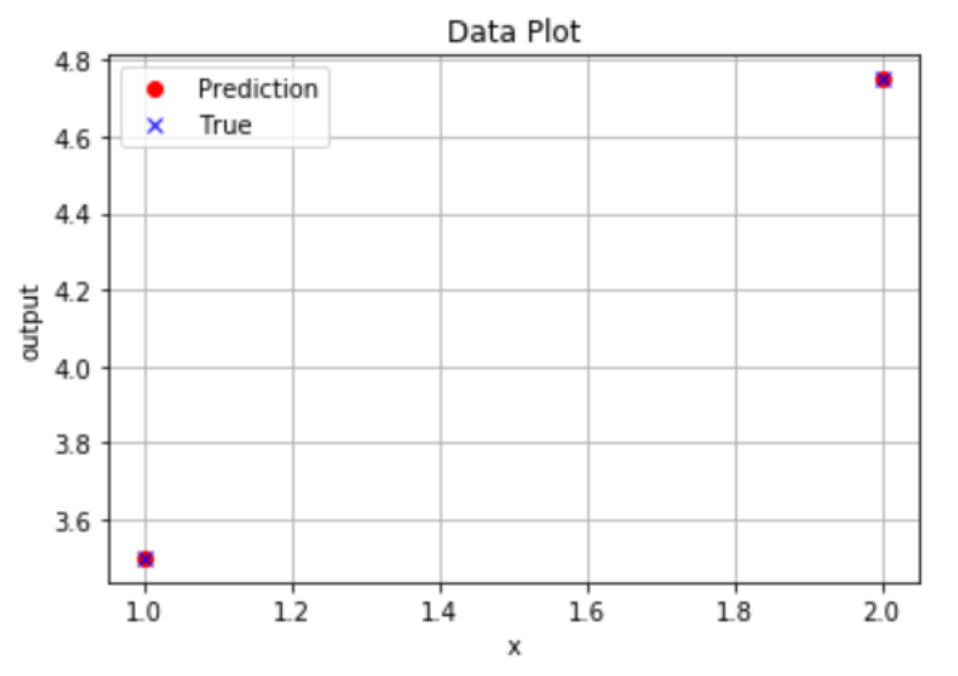
$$\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}^{(1)} = \begin{bmatrix} 1.25 \\ 2.25 \end{bmatrix}^{T} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 3.5$$

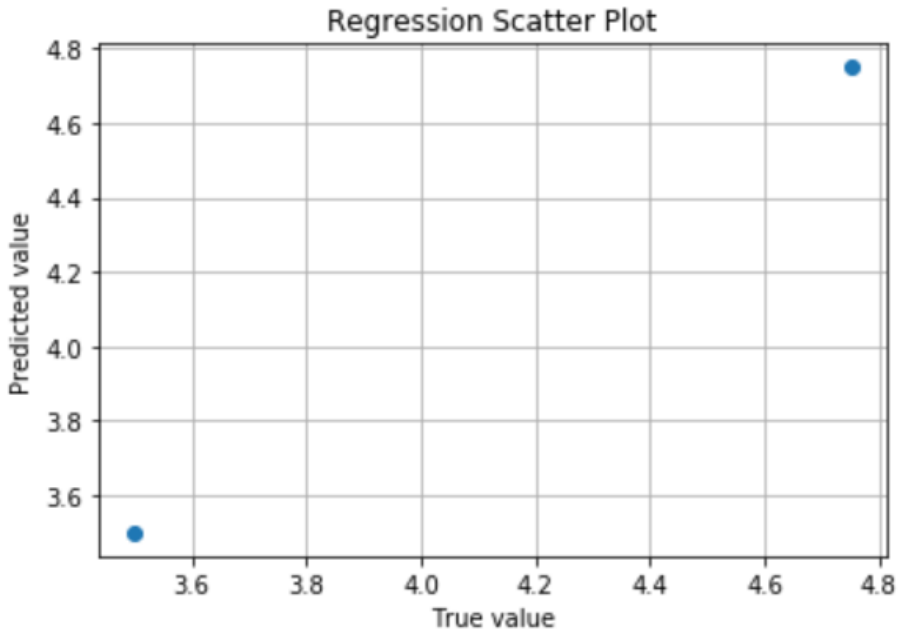
$$\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}^{(2)} = \begin{bmatrix} 1.25 \\ 2.25 \end{bmatrix}^{T} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = 4.75$$



x	y
1	3.5
2	4.75

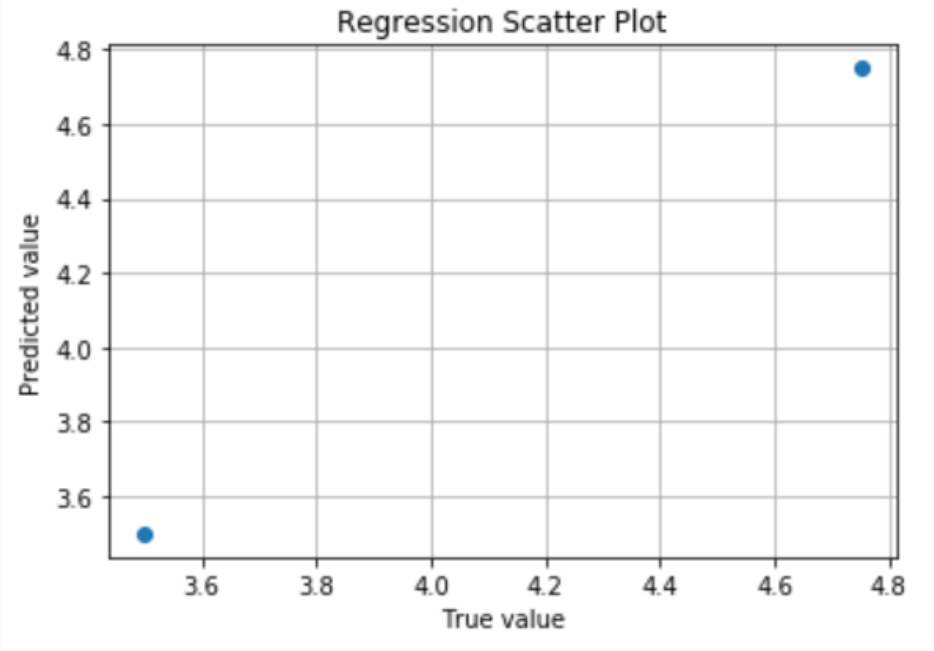
Coding






```
import numpy as np
import matplotlib.pyplot as plt
X0 = np.array([[1],[2]])
y =np.array([3.5,4.75])
X = np.hstack((X0,np.ones((X.shape[0],1)))) #append 1 to each example (for bias weights)
w = np.linalg.pinv(X)@y
f = X@w
e = f-y
L = e@e
plt.figure();plt.plot(X0,f,'ro');plt.plot(X0,y,'bx');plt.grid();plt.xlabel('x');plt.ylabel('output');plt.legend(['Prediction','True']);
plt.title('Data Plot')
plt.figure();plt.plot(y,f,'o');plt.grid();plt.xlabel('True value');plt.ylabel('Predicted value');plt.title('Regression Scatter Plot')
```

Using Sk-learn




```
from sklearn.linear_model import LinearRegression
regr = LinearRegression(fit_intercept = False).fit(X, y)
f = regr.predict(X)
print('Weights:',regr.coef_)

plt.figure();plt.plot(y,f,'o');plt.grid();plt.xlabel('True value');plt.ylabel('Predicted value');plt.title('Regression Scatter Plot')

# No need to append 1 to feature vector using below
from sklearn.linear_model import LinearRegression
regr = LinearRegression(fit_intercept = True).fit(X0, y)
f = regr.predict(X0)
plt.figure();plt.plot(y,f,'o');plt.grid();plt.xlabel('True value');plt.ylabel('Predicted value');plt.title('Regression Scatter Plot')
```

Linear Regression – Alternate Optimisation

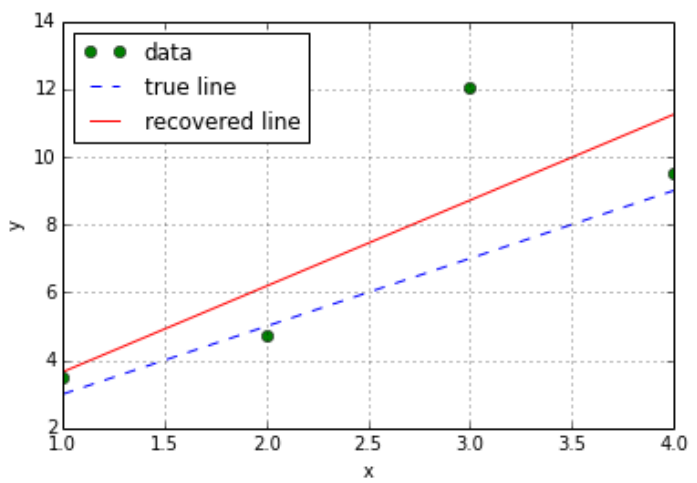
- A second way to minimize the cost function which is more broadly applicable: gradient descent.
- Gradient descent is an iterative algorithm
- We apply an update repeatedly until some criterion is met.
- We initialize the weights to something reasonable (e.g. all zeros) and repeatedly adjust them in the direction of steepest descent.
- Why?
  - GD can be applied to a much broader set of models
  - GD can be easier to implement than direct solutions, especially with automatic differentiation software
  - For regression in high-dimensional spaces, GD is more efficient than direct solution (matrix inversion is a computationally expensive operation).

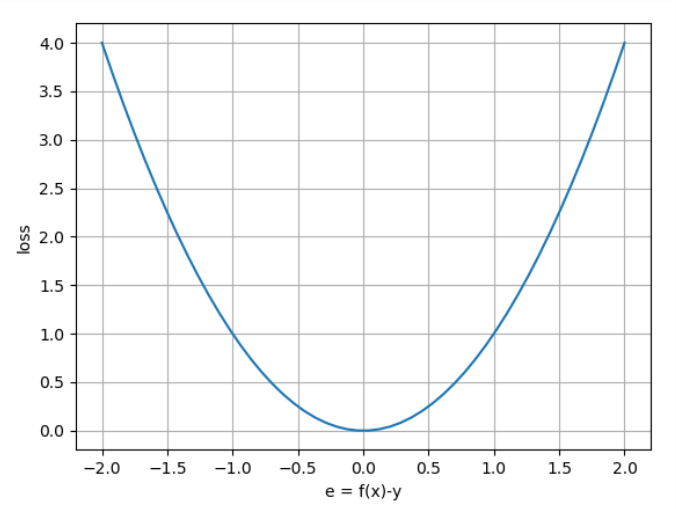
Problems with least squares solution

- Overfitting/ No regularization
  - OLS has no penalty on coefficient size, so it can fit random noise too closely.
  - With many predictors or little data, it can learn random noise.
  - We want the parameters to change slightly with change in the input data

- Sensitivity to outliers
  - Squaring errors gives unusually large errors a lot of influence.

- Cannot model non-linear data
  - A basic straight-line model cannot capture curved relationships.





Regularized Least Squares Regression

- Ridge Regression
  - Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients.
  - The ridge coefficients minimize a penalized residual sum of squares:

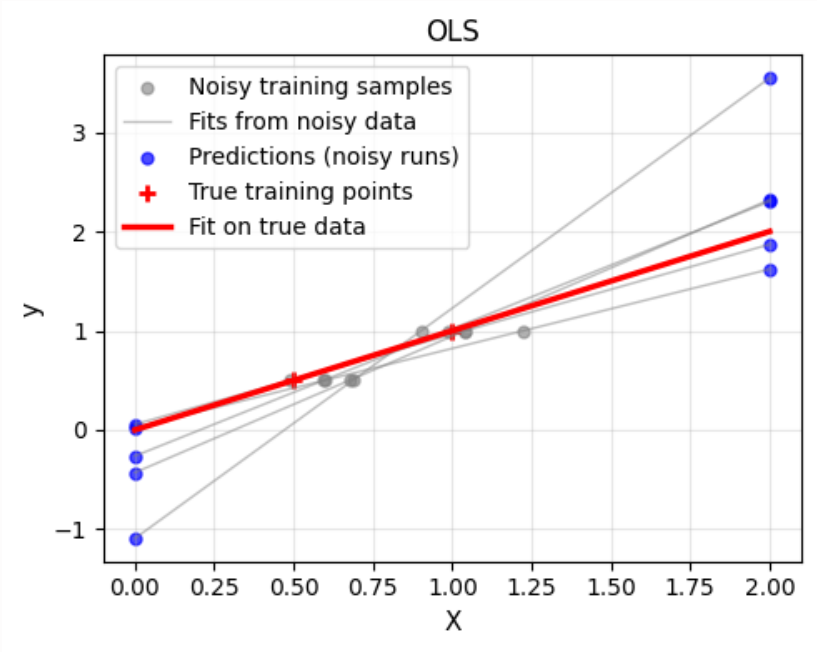
$$\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$$

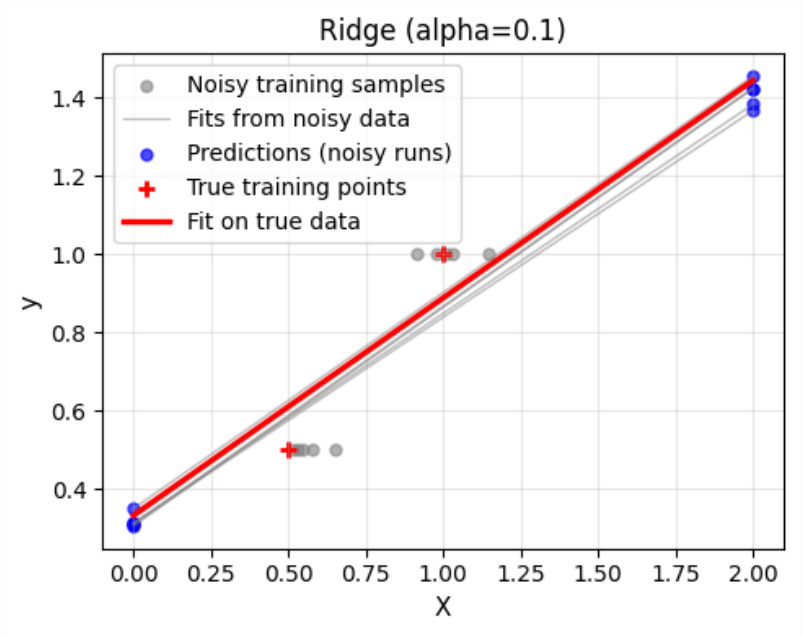
Coding

```
from sklearn import linear_model
ridge = linear_model.Ridge(alpha=a)
```

- What is the role of the hyperparameter alpha?
- How is alpha chosen?

OLS vs. Ridge Regression





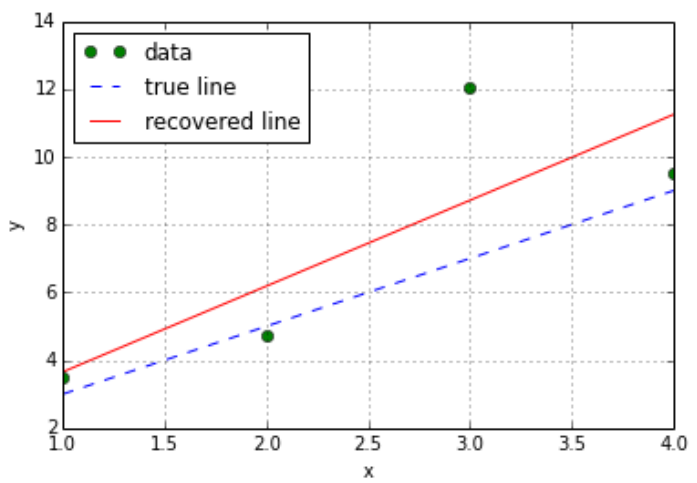
Ridge regression is basically minimizing a penalised version of the least-squared function. The penalising shrinks the value of the regression coefficients. Despite the few data points in each dimension, the slope of the prediction is much more stable and the variance in the line itself is greatly reduced, in comparison to that of the standard linear regression

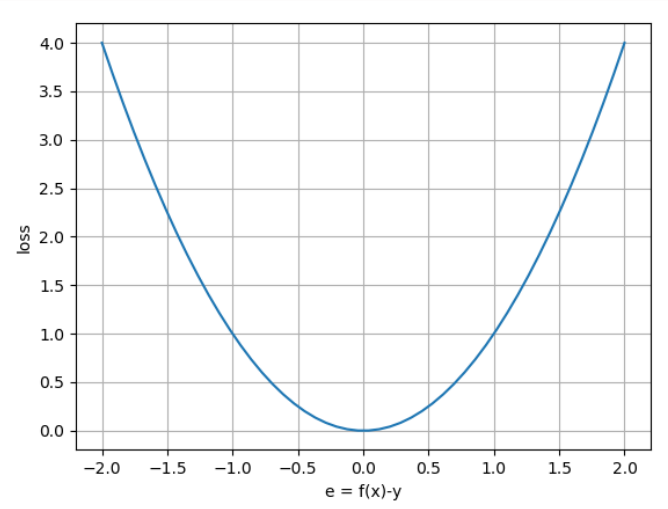
https://github.com/foxtrotmike/CS909/blob/master/regression.ipynb

https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols_ridge_variance.html#sphx-glr-auto-examples-linear-model-plot-ols-ridge-variance-py

Problems with least squares solution

- Overfitting/ No regularization
  - OLS has no penalty on coefficient size, so it can fit random noise too closely.
  - With many predictors or little data, it can learn random noise.
  - We want the parameters to change slightly with change in the input data
  - Solution: Ridge Regression
- Sensitivity to outliers
  - Squaring errors gives unusually large errors a lot of influence.
- Cannot model non-linear data
  - A basic straight-line model cannot capture curved relationships.





From ∈-insensitive loss to SVR

- ∈-insensitive loss

$$l(f(x), y) = max(0, |f(\boldsymbol{x}) - y| - \in)$$

- Support Vector Regression
  - Representation

$$f(\boldsymbol{x}) = \boldsymbol{w^T x} + b$$

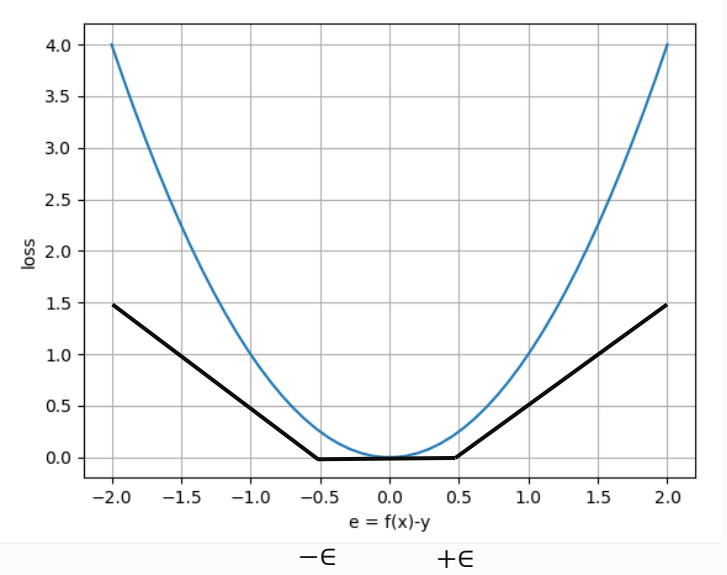
  - Evaluation

    Regularization: $\boldsymbol{w^T w}$

    Error: Epsilon insensitive loss

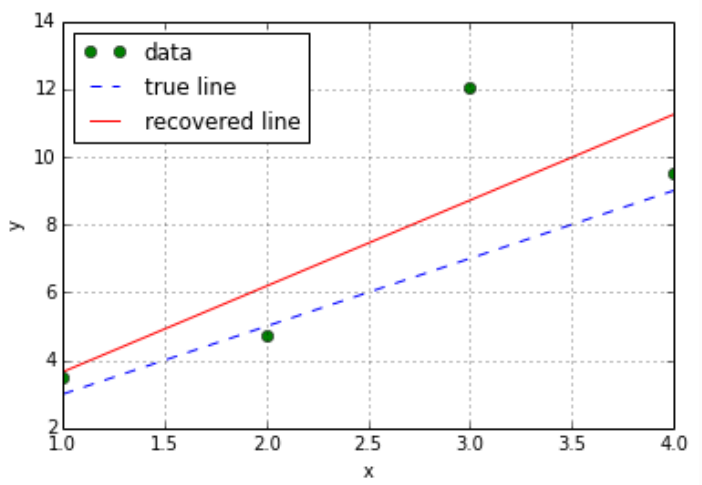
  - Optimization Problem

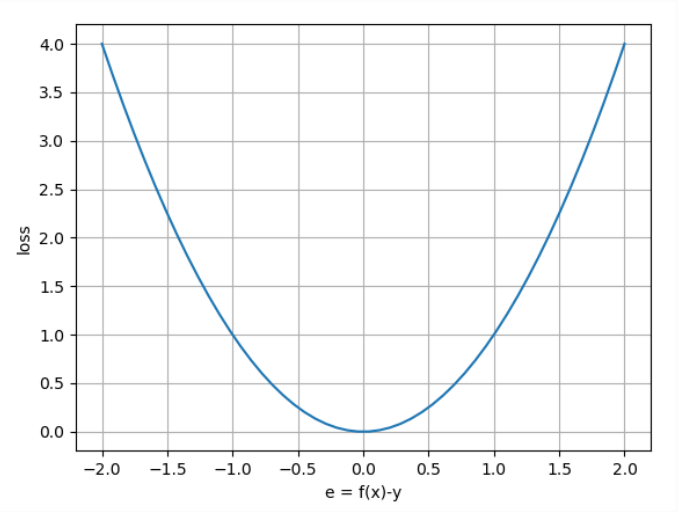
$$\min_{\boldsymbol{w},\boldsymbol{b}} \frac{1}{2}\boldsymbol{w^T w} + \frac{C}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} max(0, |f(\boldsymbol{x_i}) - y_i| - \in)$$



Problems with least squares solution

- Overfitting/ No regularization
  - OLS has no penalty on coefficient size, so it can fit random noise too closely.
  - With many predictors or little data, it can learn random noise.
  - We want the parameters to change slightly with change in the input data
  - Solution: Ridge Regression
- Sensitivity to outliers
  - Squaring errors gives unusually large errors a lot of influence.
  - Solution: Epsilon insensitive loss (SVR)
- Cannot model non-linear data
  - A basic straight-line model cannot capture curved relationships.





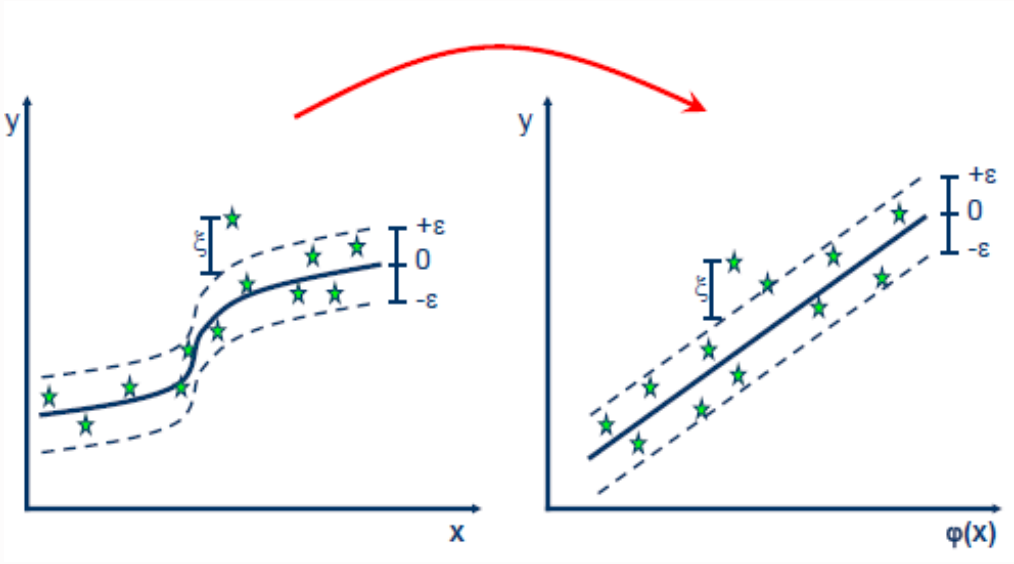
Non-linear Regression

- We handle non-linearity the same way as we did in classification
  - Transformation (e.g., polynomial expansion)

$$\boldsymbol{\phi}\colon \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \rightarrow \phi(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ x_1^2 \\ x_1 x_2 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

  - Kernels

    Fitting a line in transformed space which corresponds to an arbitrary curve in the original feature space



https://foxtrotmike.github.io/CS909/regression.html

Feature Transformation ↔ Distance Change ↔ Kernels

From lectures on SVM, we know that:

- Whenever the features are transformed
  - The distance changes
  - The dot product values changes

But **it also means that**

- **If I change the dot product, we will change the distance or in essence achieve an implicit transformation of the original feature space**
  - Kernel Trick: Replace $\boldsymbol{a^T b}$ with a Generalized Dot Product or kernel function $k(\boldsymbol{a}, \boldsymbol{b})$

Original: $k(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a^T b}$. Let's apply the transformation:

$$\boldsymbol{\phi}(a) = \boldsymbol{\phi}\left(\begin{bmatrix} a^{(1)} \\ a^{(2)} \end{bmatrix}\right) = \begin{bmatrix} a^{(1)^2} \\ a^{(2)^2} \\ \sqrt{2}a^{(1)}a^{(2)} \end{bmatrix}$$

$$\boldsymbol{\phi}(b) = \boldsymbol{\phi}\left(\begin{bmatrix} b^{(1)} \\ b^{(2)} \end{bmatrix}\right) = \begin{bmatrix} b^{(1)^2} \\ b^{(2)^2} \\ \sqrt{2}b^{(1)}b^{(2)} \end{bmatrix}$$

And compute the dot-product after transformation:

$$k_\phi(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{\phi}(\boldsymbol{a})^T \boldsymbol{\phi}(\boldsymbol{b})$$
$$= a^{(1)^2}b^{(1)^2} + a^{(2)^2}b^{(2)^2} + \sqrt{2}a^{(1)}a^{(2)}\sqrt{2}b^{(1)}b^{(2)}$$
$$= \left(a^{(1)}b^{(1)}\right)^2 + \left(a^{(2)}b^{(2)}\right)^2 + 2a^{(1)}a^{(2)}b^{(1)}b^{(2)}$$
$$= \left(a^{(1)}b^{(1)} + a^{(2)}b^{(2)}\right)^2 = \left(\boldsymbol{a^T b}\right)^2$$

The above transformation is equivalent to the polynomial kernel

Representer theorem of linear predictors

- Prediction function

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

- The Representer theorem (Scholkopf 2001) states that the weight vector of such a predictor is a linear combination of given training points as anything orthogonal to the space of training points only adds cost to the optimization:

$$\boldsymbol{w} = \alpha_1 \boldsymbol{x}_1 + \alpha_2 \boldsymbol{x}_2 + \cdots + \alpha_N \boldsymbol{x}_N = \sum_{i=1}^{N} \alpha_i \boldsymbol{x}_i$$

Thus,

$$f(\boldsymbol{x}) = b + \boldsymbol{w}^T \boldsymbol{x} = b + \left(\sum_{i=1}^{N} \alpha_i \boldsymbol{x}_i \right)^T \boldsymbol{x} = b + \sum_{i=1}^{N} \alpha_i \boldsymbol{x}_i^T \boldsymbol{x}$$

Notice how the prediction function involves only dot-products.

Generalizing the dot product to a kernel function: $k(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^T \boldsymbol{b}$

$$f(\boldsymbol{x}) = b + \sum_{i=1}^{N} \alpha_i \, k(\boldsymbol{x_i}, \boldsymbol{x})$$

Now if I change the definition of the dot-product to a generalized dot product called a kernel function, I would be able to achieve transformations!

Representer Theorem Proof

Let's assume we are given training points $\{\boldsymbol{x}_j\}_{j=1}^N$ and an optimization objective

$$\min_{\boldsymbol{w}} R(\boldsymbol{w}) + L(X, Y; \boldsymbol{w})$$

where L (loss) only depends on $\boldsymbol{w}$ through inner (dot) products $\boldsymbol{w}^T \boldsymbol{x}$ and the regularizer $R(\boldsymbol{w})$ increases with $\|w\|$ (e.g., $\frac{1}{2}\boldsymbol{w}^T \boldsymbol{w}$).

Define $\boldsymbol{w}_{\|} = \sum_{j=1}^{N} \alpha_j \boldsymbol{x}_j$ be a vector formed by an arbitrary linear combination of training points and $\boldsymbol{w}_{\perp}$ be a vector orthogonal to all such vectors.

Then, weight vector can be decomposed as: $\boldsymbol{w} = \boldsymbol{w}_{\|} + \boldsymbol{w}_{\perp}$.

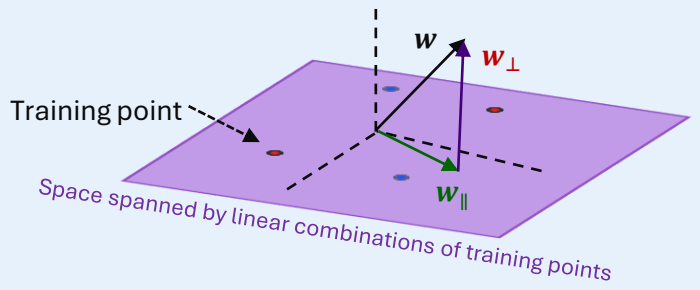
For any training point $\boldsymbol{x}_j$:

$$\boldsymbol{w}^T \boldsymbol{x}_j = (\boldsymbol{w}_{\|} + \boldsymbol{w}_{\perp})^T \boldsymbol{x}_j = \boldsymbol{w}_{\|}^T \boldsymbol{x}_j \; (\because \boldsymbol{w}_{\perp}^T \boldsymbol{x}_j = 0).$$

Hence, L is unchanged by $\boldsymbol{w}_{\perp}$.

However, as $\|\boldsymbol{w}\|^2 = \|\boldsymbol{w}_{\|}\|^2 + \|\boldsymbol{w}_{\perp}\|^2$ so any non-zero $\boldsymbol{w}_{\perp}$ will strictly increase $R(\boldsymbol{w})$. Therefore, the optimal $\boldsymbol{w}$ must have $\boldsymbol{w}_{\perp} = \boldsymbol{0}$, implying:

$$\boldsymbol{w} = \boldsymbol{w}_{\|} = \sum_{i=1}^{N} \alpha_i \boldsymbol{x}_i$$



Non-linear Regression with SVR

Using Representer Theorem & Kernel Trick

$$\min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w^T w} + \frac{C}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{max}(0, |f(\boldsymbol{x_i}) - y_i| - \epsilon)$$

Representer Theorem: $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \boldsymbol{x}_i$

$$f(\boldsymbol{x}) = \boldsymbol{w}^T x + b = \sum_{i=1}^{N} \alpha_i \boldsymbol{x}_i^T \boldsymbol{x} + b$$

$$\min_{\boldsymbol{\alpha},b} \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i \alpha_j \, \boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x}_j + \frac{C}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{max}\left(0, \left|\sum_{j=1}^{N} \alpha_i \boldsymbol{x}_i^T \boldsymbol{x}_j + b - y_i\right| - \epsilon\right)$$

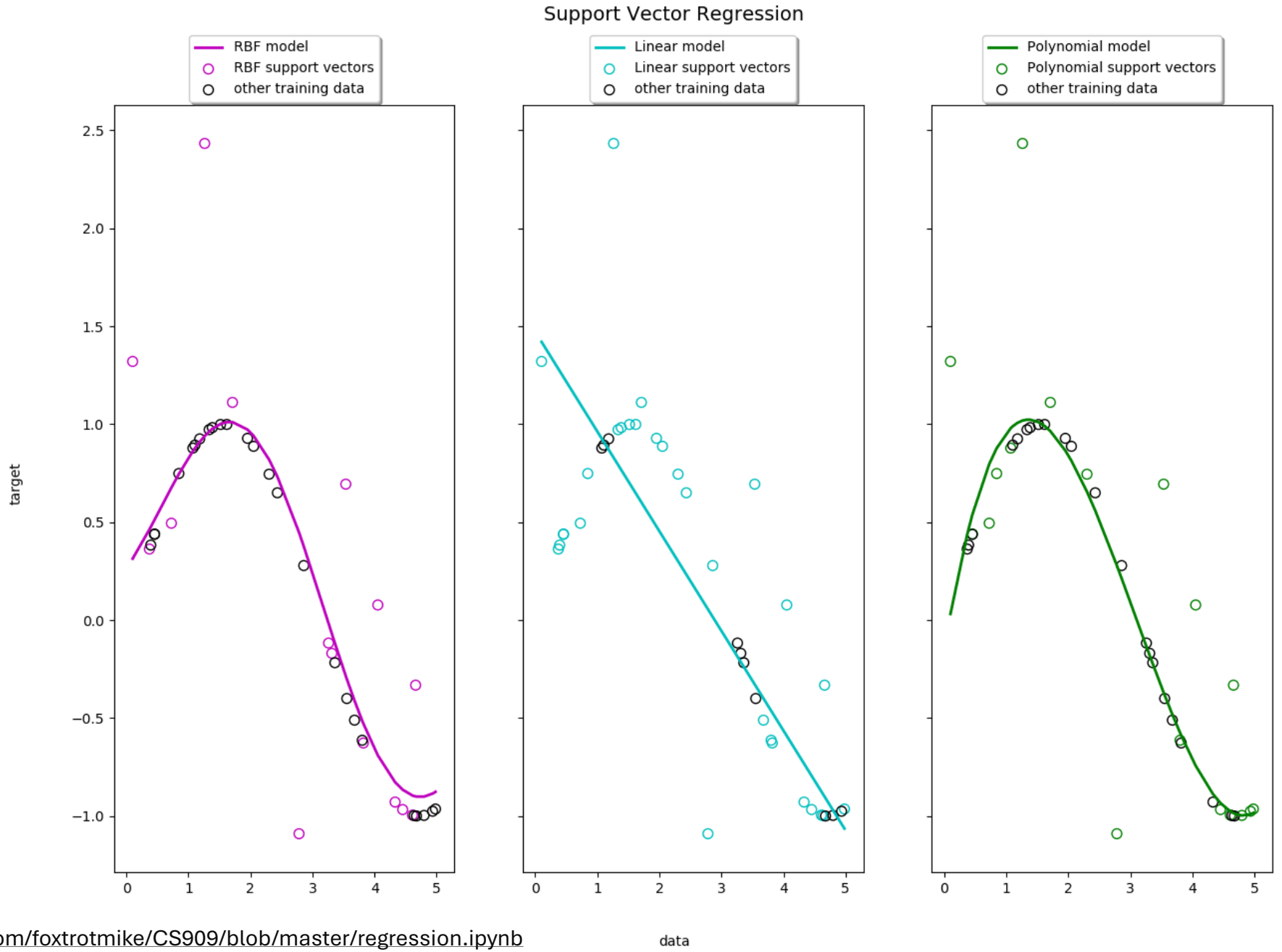
Kernel Trick: Replacing $\boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x}_j$ with $k(x_i, x_j)$

$$f(\boldsymbol{x}) = \boldsymbol{w}^T x + b = \sum_{i=1}^{N} \alpha_i \boldsymbol{x}_i^T \boldsymbol{x} + b = \sum_{i=1}^{N} k(x_i, x) + b$$

$$\min_{\boldsymbol{\alpha},b} \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i \alpha_j \, k(\boldsymbol{x_i}, \boldsymbol{x_j}) + \frac{C}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{max}\left(0, \left|\sum_{j=1}^{N} k(\boldsymbol{x_i}, \boldsymbol{x_j}) + b - y_i\right| - \epsilon\right)$$

Kernels	Functions	Parameters
Linear	$< x, x_i >$	
Polynomial	$(< x, x_i > +1)^d$	d
Radial basis function	$\exp\left(-\|x - x_i\|^2 / 2\sigma^2\right)$	σ
Sigmoid	$\tanh(b < x, x_i > + c)$	b, c

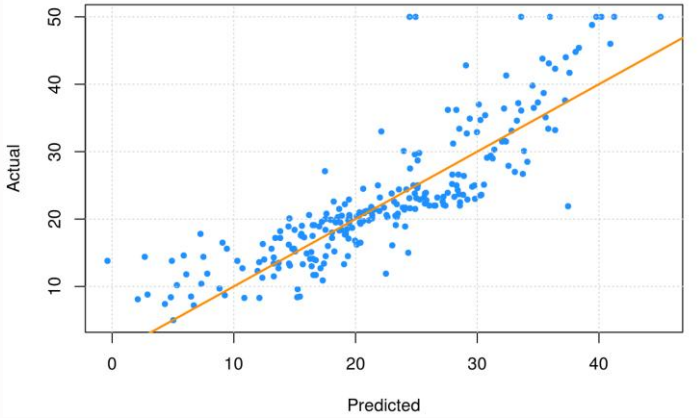
Ex



See: https://github.com/foxtrotmike/CS909/blob/master/regression.ipynb
https://scikit-learn.org/stable/auto_examples/svm/plot_svm_regression.html

Cross-validation and performance metrics

- Evaluation protocols for hyperparameter selection remain the same as in classification
- How to calculate how good we are doing?
  - Measure error or correlation between true and predicted outputs for different examples in a test set
- Error Measures
  - Sum of squared errors
  - Mean squared error
  - Mean Absolute error
  - Root mean square error
  - R^2 score
- Pearson and Spearman Correlation Coefficients



$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2$$

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|$$

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

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

$$r = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2}}$$

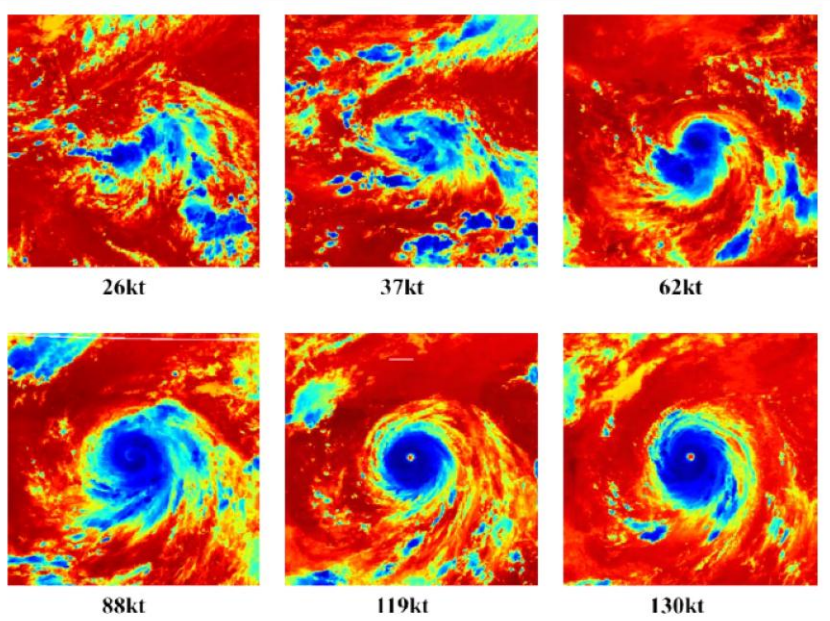
https://scikit-learn.org/stable/modules/model_evaluation.html
https://github.com/foxtrotmike/CS909/blob/master/regression.ipynb

Practical Application

Hurricane Intensity Estimation

Input: Infrared Satellite Images of Hurricanes

Output: Maximum Sustained Windspeed in knots

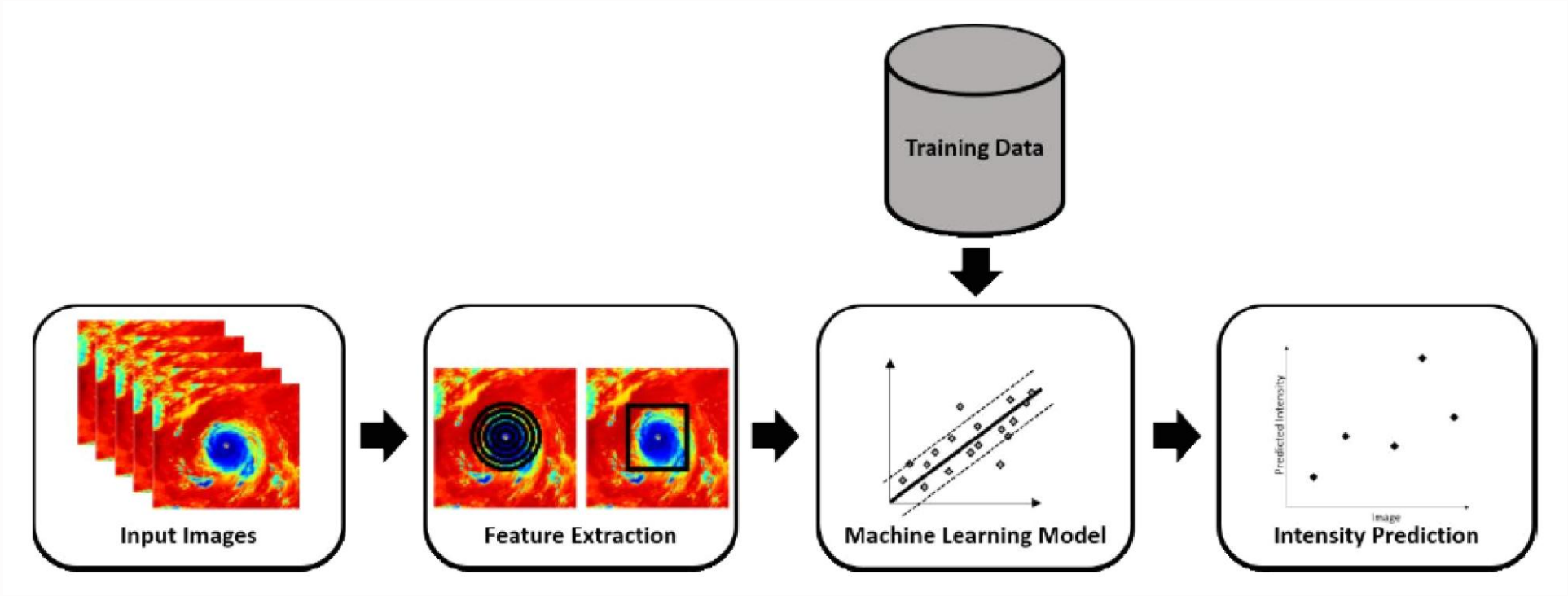






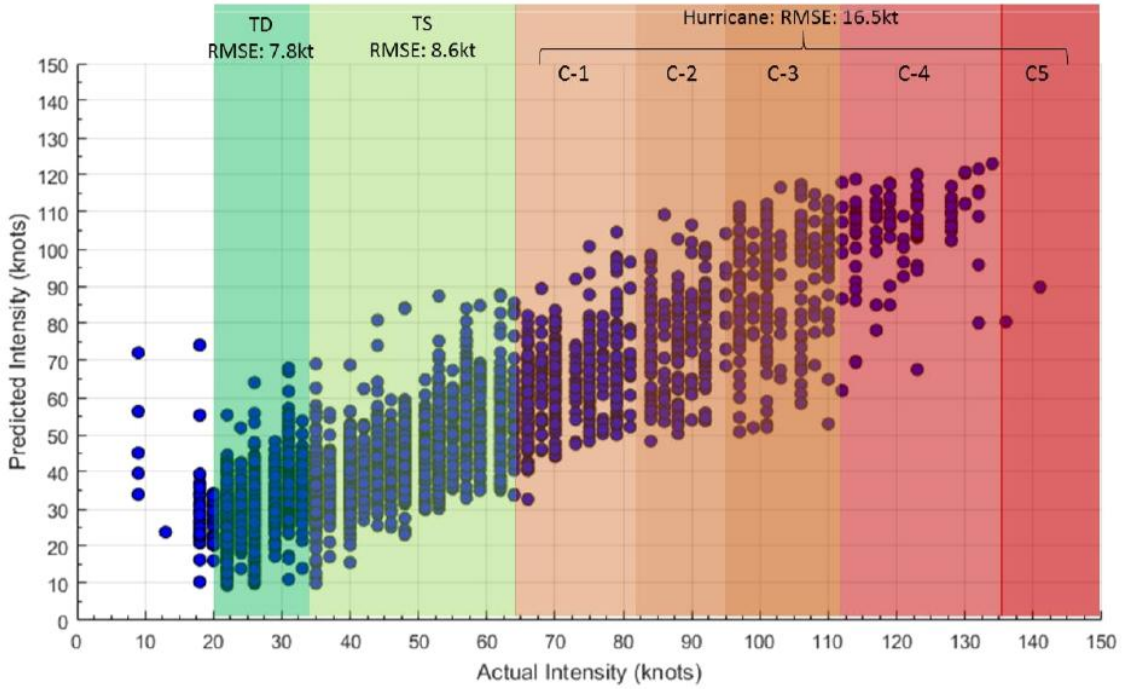
https://en.wikipedia.org/wiki/Hurricane_hunters

PHURIE ML Pipeline

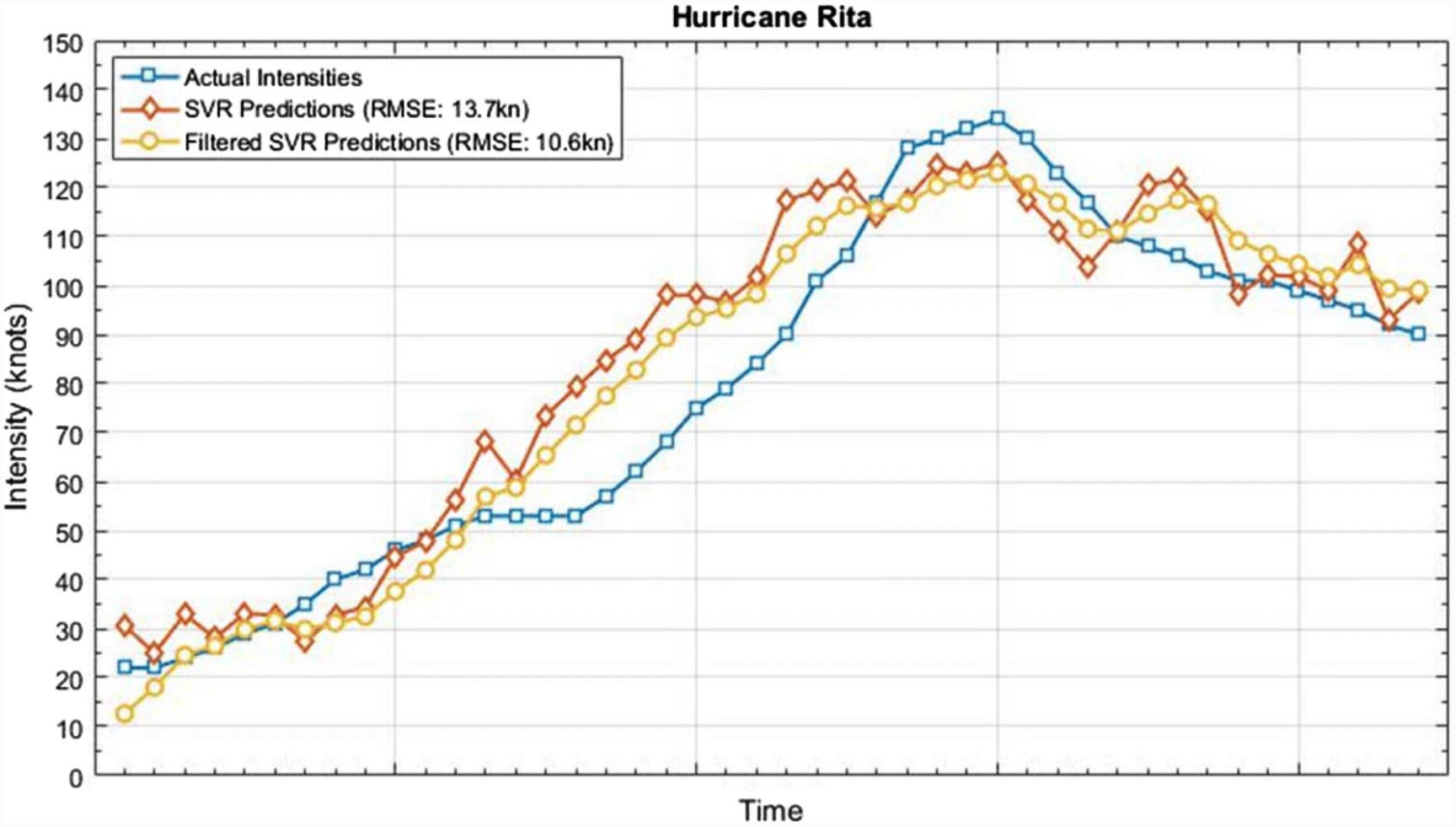


PHURIE: Hurricane Intensity Estimation from Infrared Satellite Imagery using Machine Learning, Amina Asif, Muhammad Dawood, Bismillah Jan, Javaid Khurshid, Mark DeMaria, and **Fayyaz ul Amir Afsar Minhas**, in Neural Computing and Applications, DOI: http://dx.doi.org/10.1007/s00521-018-3874-6, 2018. (Paper)

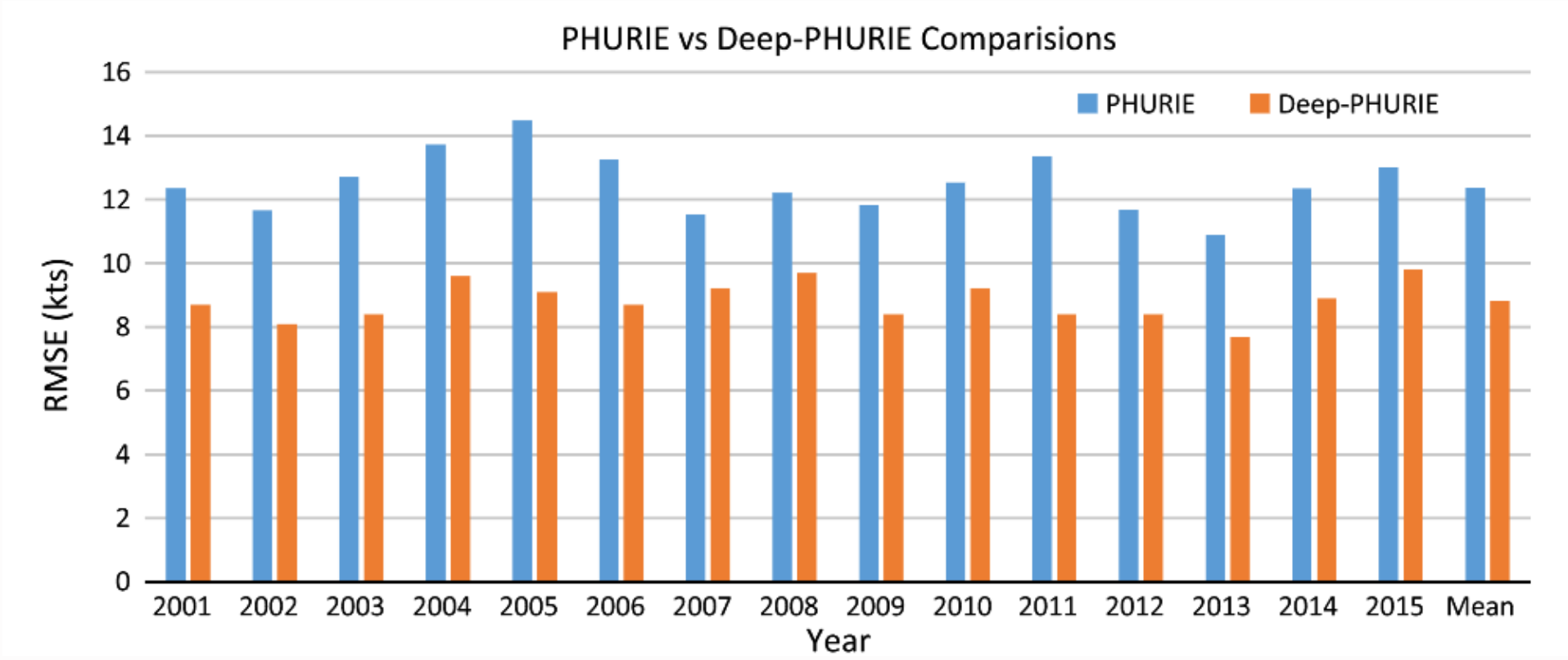
Practical Application



Method	Mean RMSE (kt)	Mean RMSE after smoothing
PHURIE: SVR	**11.2**	**9.5**
PHURIE: OLS	12.8	10.5
PHURIE: BPNN	12.0	10.1
PHURIE: XGBoost	11.3	9.8
Baseline predictor (mean)	24.3	–


Hurricane Rita
Actual Intensities
SVR Predictions (RMSE: 13.7kn)
Filtered SVR Predictions (RMSE: 10.6kn)
Intensity (knots)
Time
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150

Extension: Deep PHURIE



Deep-PHURIE: Deep Learning based Hurricane Intensity Estimation from Infrared Satellite Imagery, M. Dawood, A. Asif and Fayyaz Minhas, in Neural Computing and Applications. pp. DOI: 10.1007/s00521-019-04410-7, July 2019.

Are these enough to quantify the performance?

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2$$

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|$$

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

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

$$r = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2}}$$

- These metrics quantify the accuracy of predictions
- What would be R^2 if the predictions are perfect?
  - What would it be if we always predict the mean?

So we can quantify the accuracy…

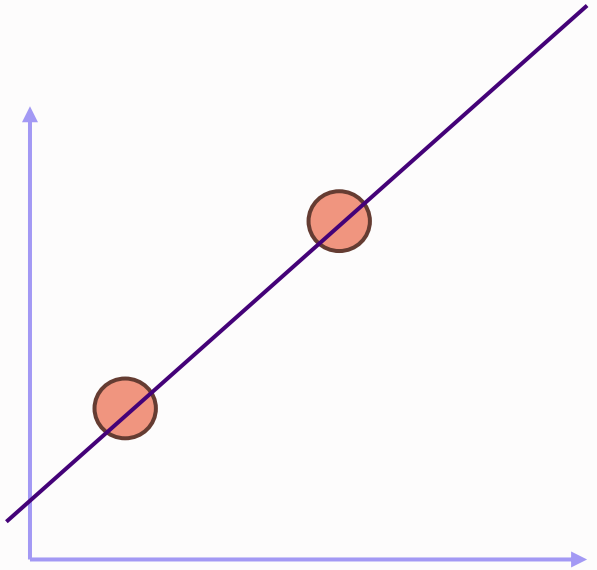
But what if someone asks how confident are we?

Why do we need to quantify confidence in our predictions?

- The probability that random chance could give us similar or better predictions!
- Why would that matter if we are getting a high enough R^2?
- If a random chance can give as predictions that are as good, the predictor isn't adding any information!
  - We can't have a lot of confidence in those predictions

R^2 =1

True for any two random points



R^2 =?

How to quantify confidence in predictions?

- P-values
  - For coefficients
    - **a predictor's p-value tells you whether there is evidence that its coefficient differs from zero**
  - For the overall model
    - **overall model p-value** tests whether the predictors, taken together, improve on simply predicting the average outcome

Statistical Inference by Regression

- Statistical Inference

  The goal of regression is to identify how a change in one variable changes the expected value of the other variable while keeping all else the same

- Linear regression: E[y|x] = b + w x

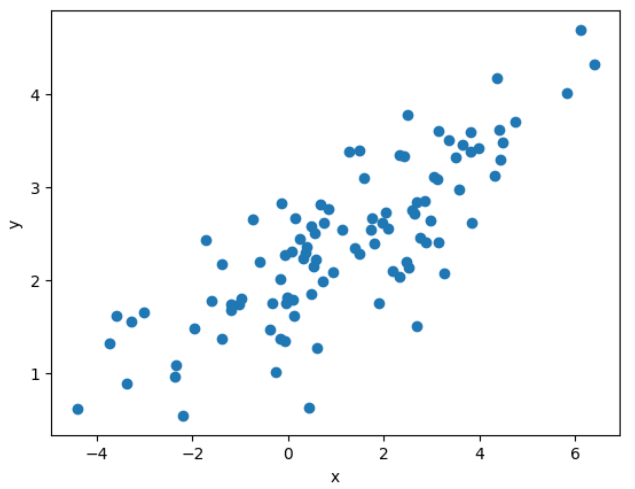
  Statistics libraries write it using the shorthand notation: $y \sim 1 + x$

  What do the coefficients tell us?

  w tells us if x and y have a (linearly) association

  We can get "p-values" for the regression coefficient and use this as a hypothesis test to test if the two variables are indeed related

  In the example in the given code (link below): For each one-unit increase in x, the expected value of y increases by 0.2970 units, holding all other variables constant. This coefficient is also statistically significant, with a very low p-value ($p < 0.05$), suggesting a strong linear relationship between x1 and y.



Synthetic Data Generation

```
# Array of 1000 values with mean = 1.5, stddev = 2.5
X = 2.5 * np.random.randn(1000) + 1.5
# Generate 1000 noise terms
noise = 0.5 * np.random.randn(1000)
# Actual values of Y
y = 2 + 0.3 * X + noise
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.722
Model:	OLS	Adj. R-squared:	0.721
Method:	Least Squares	F-statistic:	2327.
Date:	Sat, 10 Feb 2024	Prob (F-statistic):	1.57e-251
Time:	02:50:10	Log-Likelihood:	-624.91
No. Observations:	900	AIC:	1254.
Df Residuals:	898	BIC:	1263.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.0554	0.019	108.830	0.000	2.018	2.092
x1	0.2970	0.006	48.237	0.000	0.285	0.309

Omnibus:	1.085	Durbin-Watson:	2.113
Prob(Omnibus):	0.581	Jarque-Bera (JB):	1.106
Skew:	0.002	Prob(JB):	0.575
Kurtosis:	2.828	Cond. No.	3.69

https://github.com/foxtrotmike/CS909/blob/master/regression_residuals.ipynb

Questions

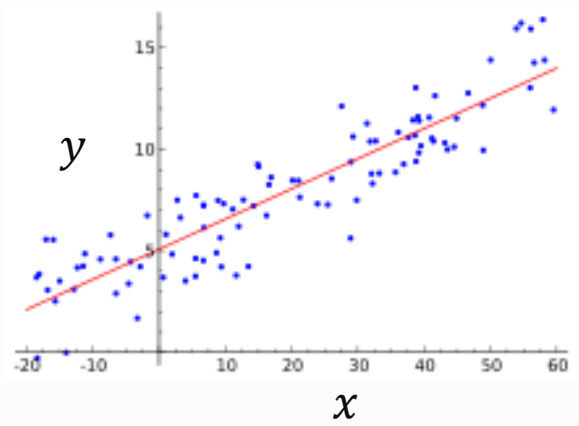
If two variables are unrelated

What would be their regression coefficients?

If two variables are positively related

What would be their regression coefficient?

Would the error be correlated with the independent variable?



Required

Reading:

- Section 13.10 in Alpaydin 2010
- http://ciml.info/ Linear Models
- Very useful: Regression with linear models
  - https://scikit-learn.org/stable/modules/linear_model.html
- Code: **Scikit-learn**, https://github.com/NICTA/revrand
- StatsModel Regression Exercises: https://www.statsmodels.org/dev/examples/index.html