ECE5984 – Applications of Machine Learning
Lecture 16 – Linear Regression

Creed Jones, PhD

Course update

- Next quiz on Thursday, March 24
- HW4 will be posted on Monday
  - Due April 5
- Project I
  - Due on Tuesday, March 22

I used an incorrect term in the Project I assignment

- I said:
- "Measure the performance as the True Positive Rate:"

$$TPR = \frac{|TestSetCorrectlyPredicted|}{|TestSet|}$$

- Unfortunately, this is not the definition of true positive rate!
  - I have some sort of mental block on this issue…

- I should have said:
- "Measure the performance as the Classification Accuracy:"

$$Accuracy = \frac{|TestSetCorrectlyPredicted|}{|TestSet|}$$

I want to show more explicitly how the gradient descent for multivariate linear regression works

- The function we are minimizing is the mean square error between the actual target variable (call it f) and the model output ($\hat{f}$):
  - $err(\boldsymbol{w}) = \sum_{x,y}\left(f(x,y) - \hat{f}(x,y)\right)^2 = \sum_{x,y}\left(f(x,y) - \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x \\ y \end{bmatrix}\right)^2 = \sum_{x,y}(f(x,y) - w_0 - w_1 x - w_2 y)^2$
- The gradient descent update equation is:

$$\begin{bmatrix} w_{0,n+1} \\ w_{1,n+1} \\ w_{2,n+1} \end{bmatrix} = \begin{bmatrix} w_{0,n} \\ w_{1,n} \\ w_{2,n} \end{bmatrix} - \gamma_n \nabla err(\boldsymbol{w}) = \begin{bmatrix} w_{0,n} \\ w_{1,n} \\ w_{2,n} \end{bmatrix} - \gamma_n \begin{bmatrix} \frac{\partial err}{\partial w_0} \\ \frac{\partial err}{\partial w_1} \\ \frac{\partial err}{\partial w_2} \end{bmatrix} = \begin{bmatrix} w_{0,n} \\ w_{1,n} \\ w_{2,n} \end{bmatrix} - \gamma_n \sum_{x,y} \begin{bmatrix} -2(f(x,y) - w_0 - w_1 x - w_2 y) \\ -2x(f(x,y) - w_0 - w_1 x - w_2 y) \\ -2y(f(x,y) - w_0 - w_1 x - w_2 y) \end{bmatrix}$$

- $$\begin{bmatrix} w_{0,n+1} \\ w_{1,n+1} \\ w_{2,n+1} \end{bmatrix} = \begin{bmatrix} w_{0,n} \\ w_{1,n} \\ w_{2,n} \end{bmatrix} - \gamma_n \sum_{x,y} \begin{bmatrix} -2(f(x,y) - w_0 - w_1 x - w_2 y) \\ -2x(f(x,y) - w_0 - w_1 x - w_2 y) \\ -2y(f(x,y) - w_0 - w_1 x - w_2 y) \end{bmatrix} = \begin{bmatrix} w_{0,n} \\ w_{1,n} \\ w_{2,n} \end{bmatrix} + 2\gamma_n \sum_{x,y} err(x,y) \begin{bmatrix} 1 \\ x \\ y \end{bmatrix}$$

Look at the update equation referred to in the book example…

$$\begin{bmatrix} w_{0,n+1} \\ w_{1,n+1} \\ w_{2,n+1} \end{bmatrix} = \begin{bmatrix} w_{0,n} \\ w_{1,n} \\ w_{2,n} \end{bmatrix} - \gamma_n \sum_{x,y} \begin{bmatrix} -2(f(x,y) - w_0 - w_1 x - w_2 y) \\ -2x(f(x,y) - w_0 - w_1 x - w_2 y) \\ -2y(f(x,y) - w_0 - w_1 x - w_2 y) \end{bmatrix}$$

$$\begin{bmatrix} w_{0,n+1} \\ w_{1,n+1} \\ w_{2,n+1} \end{bmatrix} = \begin{bmatrix} w_{0,n} \\ w_{1,n} \\ w_{2,n} \end{bmatrix} + 2\gamma_n \textstyle\sum_{x,y} err(x,y) \begin{bmatrix} 1 \\ x \\ y \end{bmatrix}$$

$$w_{j+1} = w_j + 2\gamma \sum_{\mathcal{D}} err(x,y) \begin{bmatrix} 1 \\ x \\ y \end{bmatrix}$$

$$\mathbf{w}[j] \leftarrow \mathbf{w}[j] + \alpha \underbrace{\sum_{i=1}^{n} ((t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)) \times d_i[j])}_{errorDelta(\mathcal{D},\mathbf{w}[j])}$$

Initial Weights

w[0]:	-0.146	**w**[1]:	0.185	**w**[2]:	-0.044	**w**[3]:	0.119

Example

$$\mathbf{w}[1] \leftarrow 0.185 + 0.00000002 \times 2,412,074 = 0.23324148$$

New Weights (Iteration 1)

w[0]:	-0.146	**w**[1]:	0.233	**w**[2]:	-0.043	**w**[3]:	0.121

Today's Objectives

Non-linear relationships

- Basis functions

More on Linear Regression

- Interpreting linear regression models
- Weight decay
- Categorical features

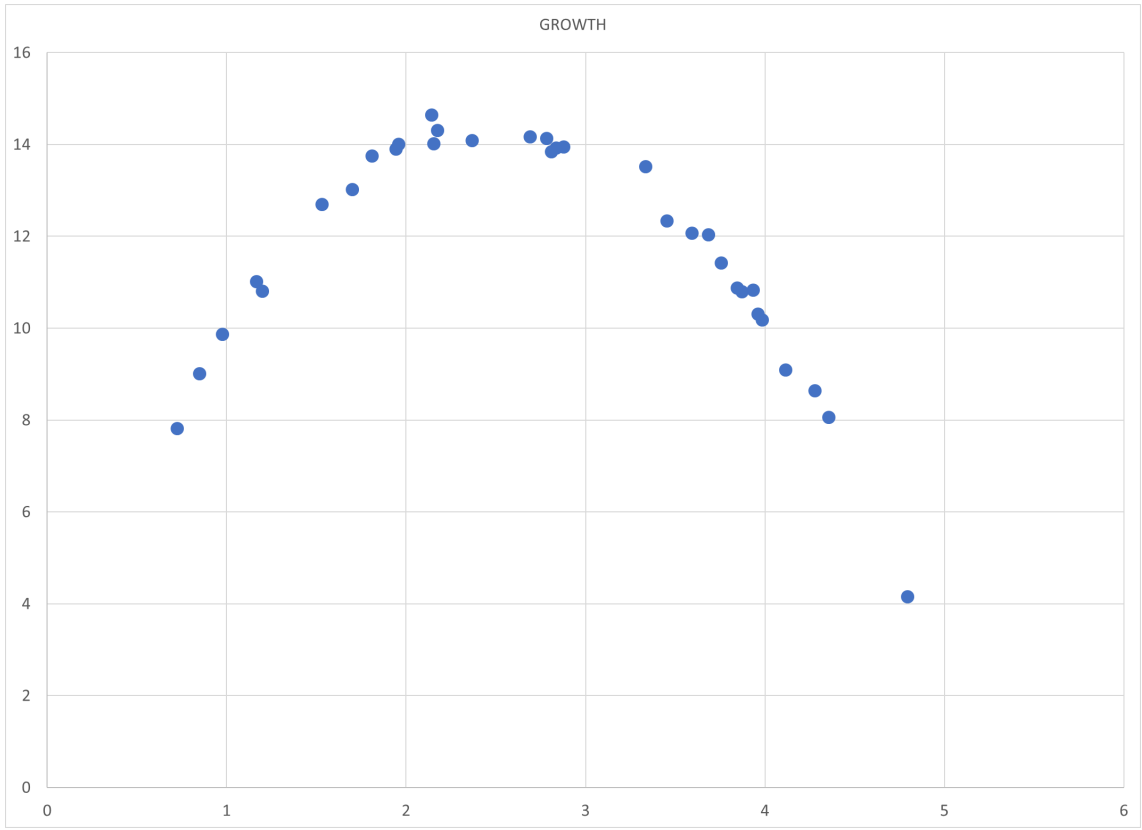
NON-LINEAR RELATIONSHIPS: VARIABLE TRANSFORMATIONS

ID	RAIN	GROWTH
1	2.153	14.016
2	3.933	10.834
3	1.699	13.026
4	1.164	11.019
5	4.793	4.162
6	2.69	14.167
7	3.982	10.19
8	3.333	13.525
9	1.942	13.899
10	2.876	13.949
11	4.277	8.643
12	3.754	11.42
13	2.809	13.847
14	1.809	13.757
15	4.114	9.101
16	2.834	13.923
17	3.872	10.795
18	2.174	14.307
19	4.353	8.059
20	3.684	12.041
21	2.14	14.641
22	2.783	14.138
23	3.96	10.307
24	3.592	12.069
25	3.451	12.335
26	1.197	10.806
27	0.723	7.822
28	1.958	14.01
29	2.366	14.088
30	1.53	12.701
31	0.847	9.012
32	3.843	10.885
33	0.976	9.876

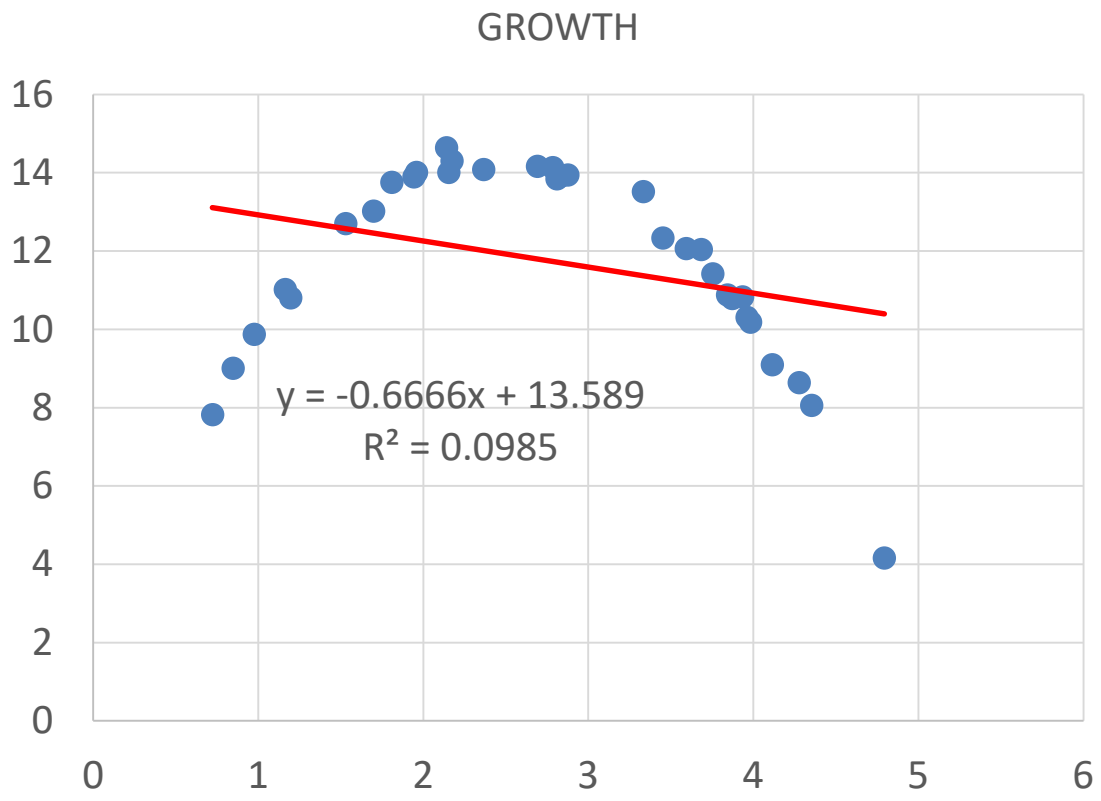
Here is a dataset of grass growth on Irish farms in July 2012; the only feature here is RAIN

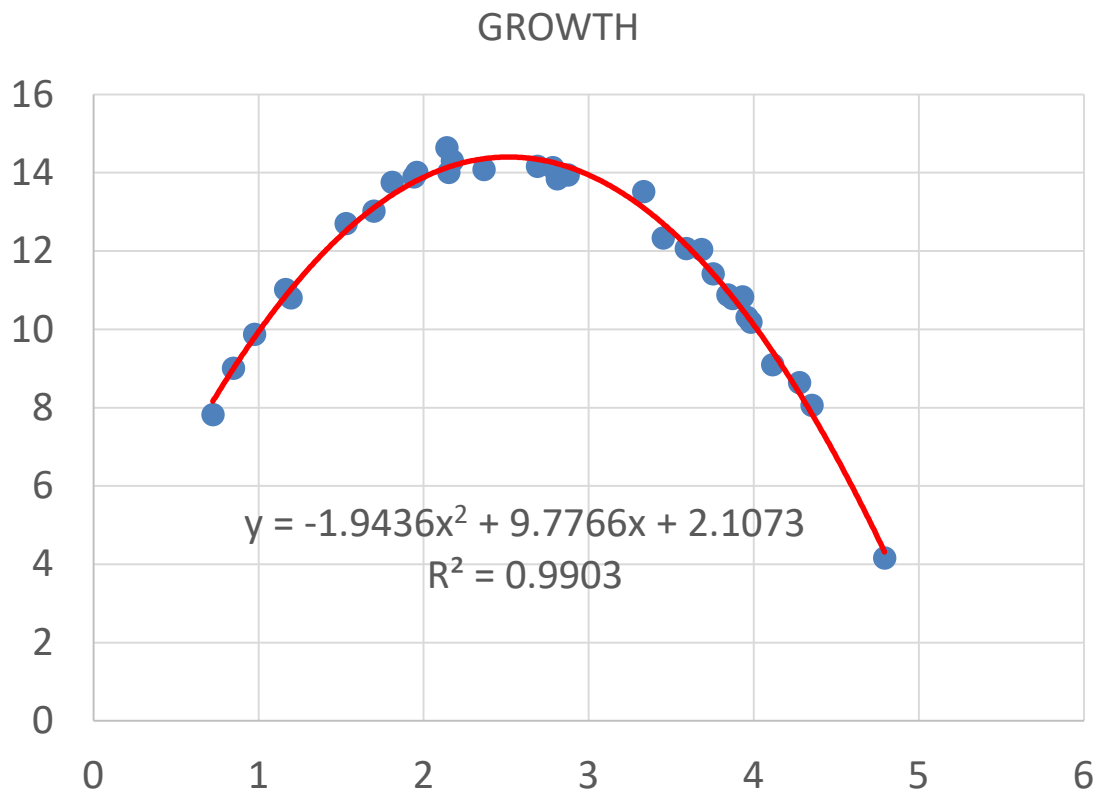
A scatterplot of GROWTH vs. RAIN strongly suggests a second-order polynomial relationship

NOTE: It's never this obvious in real life…



A linear model is horrible – but if we can allow a second-order relationship, the fit is very close





Instead of extending the model fitting to higher order equations, we modify our dataset to contain a set of *basis functions* computed on the initial features

- The model will now look like:

$$\mathbb{M}_{\boldsymbol{w}}(\boldsymbol{d}) = \sum_{k=0}^{b} \boldsymbol{w}[k] \cdot \phi_k(\boldsymbol{d})$$

- where $\phi_0 \cdots \phi_b$ are a series of basis functions that transform the input vector

- Advantages:
  - The regression approach and gradient descent algorithm are unchanged
  - We can select suitable basis functions for the problem at hand
- Disadvantages:
  - The dataset is much wider (more columns, same number of rows)
  - How do we choose basis functions?

Because we see the suggestion (!) of a second-order relationship, let's choose basis functions appropriately

$$\boldsymbol{\phi}(\boldsymbol{d}) = \begin{bmatrix} \phi_0(RAIN) \\ \phi_1(RAIN) \\ \phi_2(RAIN) \end{bmatrix} = \begin{bmatrix} 1 \\ RAIN \\ RAIN^2 \end{bmatrix}$$

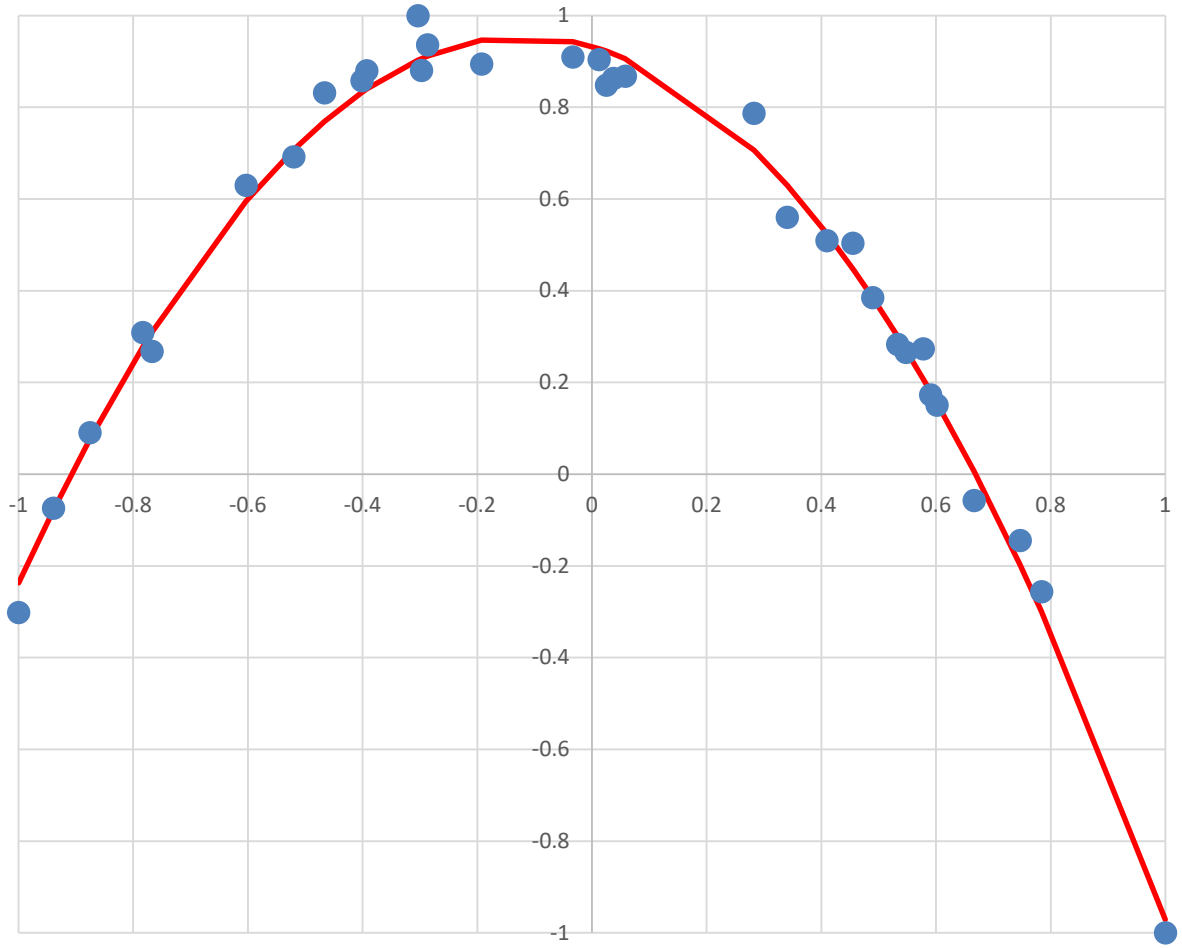
$$GROWTH = \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{d}) = w[0] \cdot \phi_0(RAIN) + w[1] \cdot \phi_1(RAIN) + w[2] \cdot \phi_2(RAIN)$$

$$GROWTH = w[0] + w[1] \cdot RAIN + w[2] \cdot RAIN^2$$

The usual linear regression procedure converges on the following model:

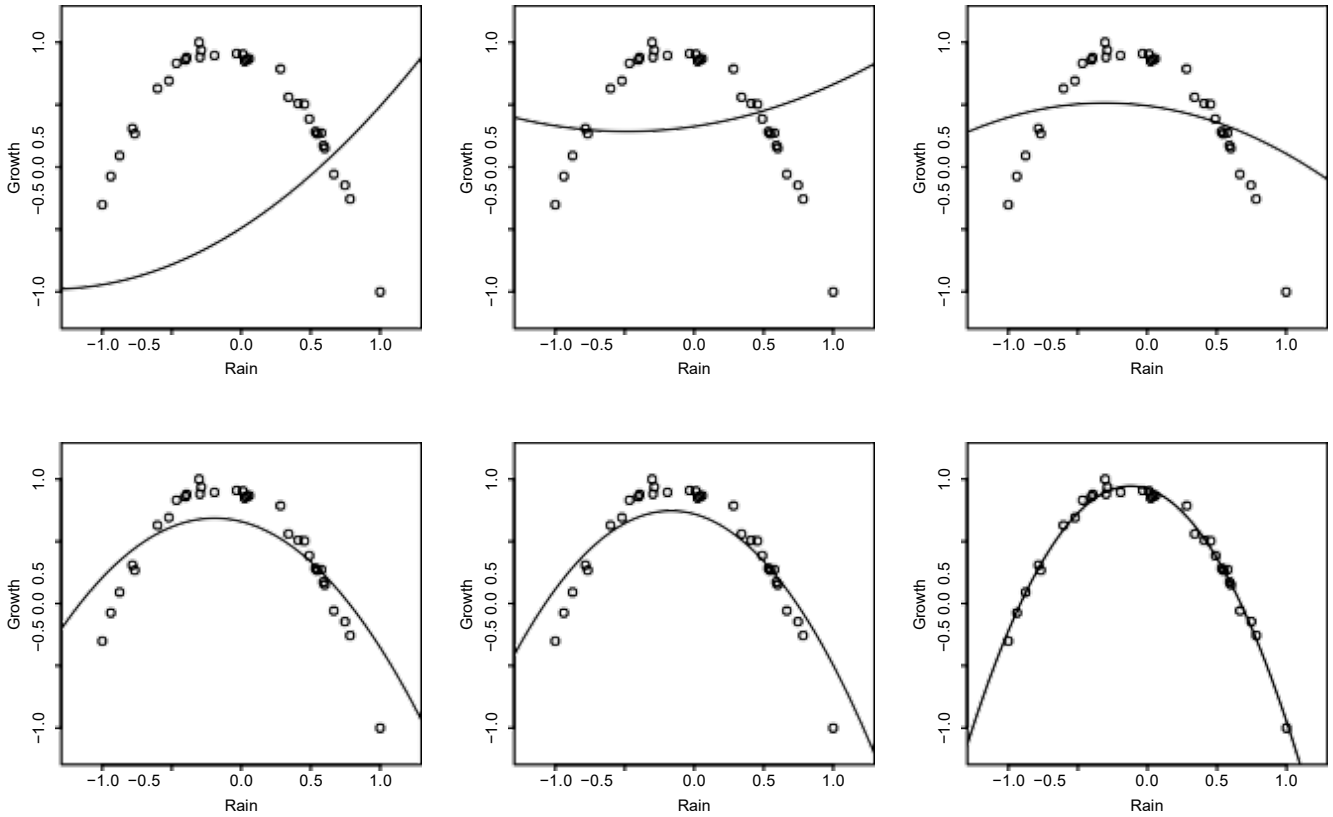
$$GROWTH = 0.3707 + 0.8475 \cdot RAIN - 1.717 \cdot RAIN^2$$

$$GROWTH = 0.3707 + 0.8475 \cdot RAIN - 1.717 \cdot RAIN^2$$



- Using the basis functions allowed a good fit to the training set
- First, apply the basis functions
- Then, scale each feature to [-1, 1]
- Note that it's not sufficient to only use the second-order term
  - The first-order term is needed as well
- This does not match the model printed in the book
  - Theirs does not fit the data as well
  - I have no idea why

The successive models that result from the gradient descent method on the transformed dataset



Linear Basis Function Models are multivariate linear models of the features and basis functions operating on those features

So, our model would be

$$\mathbb{M}_{\boldsymbol{W}}(\boldsymbol{d}) = w[0] + \sum_{j=1}^{M} w[j]\phi_j(d[j]) = \boldsymbol{w}^T \cdot \boldsymbol{\phi}(\boldsymbol{d})$$

where $\boldsymbol{w}$ is the weight vector, $\boldsymbol{d}$ is the augmented feature vector, and $\boldsymbol{\phi}$ is a vector of basis (transforming) functions chosen from a family

- how are the basis functions chosen? Stay tuned…

If we used the family of polynomials of order 1 and 2, the model for the office data set would be:

$$\mathbb{M}_{\boldsymbol{W}}(\boldsymbol{d}) = w[0] + w[0] \cdot SIZ + w[0] \cdot SIZ^2 + w[0] \cdot FLR + w[0] \cdot FLR^2 + w[0] \cdot BBR + w[0] \cdot BBR^2$$

Basis functions are a powerful addition to both linear and logistic regression

We can use basis functions for:

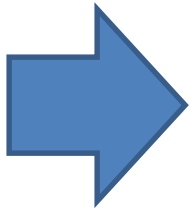
- multivariable simple linear regression models in the same way, the only extra requirement being the definition of more basis functions.
- to train logistic regression models for categorical prediction problems that involve non-linear relationships

$$\mathbb{M}_{\boldsymbol{w}}(\boldsymbol{d}) = \frac{1}{1 + e^{-\sum_0^b \boldsymbol{w}[\boldsymbol{j}]\phi_{\boldsymbol{j}}(d)}}$$

Consider the generator dataset again, using a set of basis functions for nonlinear variable transformations

- We use sklearn.preprocessing.PolynomialFeatures()
- This creates an object that will transform a dataset to include all possible products of features, up to the desired polynomial degree…
- Other alternatives include PowerTransformer(), FunctionTransformer()…

ID	RPM	VIBR	NORM_RPM	NORM_VIBR	GOOD
1	498	604	-1.0000	1.0000	0
2	517	594	-0.9246	0.9586	0
3	541	574	-0.8294	0.8758	0
4	555	587	-0.7738	0.9296	0
…	…	…	…	…	…
65	950	156	0.7937	-0.8551	1
66	956	174	0.8175	-0.7805	1
67	973	134	0.8849	-0.9462	1
68	1002	121	1.0000	-1.0000	1



ID	RPM_NM	VIBR_NM	RPM_NMSQ	VIBR_NMSQ	RPM_VIBR_NM	GOOD
1	-1.0000	1.0000	1.0000	1.0000	-1.0000	0
2	-0.9246	0.9586	0.8549	0.9189	-0.8863	0
3	-0.8294	0.8758	0.6878	0.7670	-0.7263	0
4	-0.7738	0.9296	0.5988	0.8642	-0.7193	0
…	…	…	…	…	…	…
65	0.7937	-0.8551	0.6299	0.7311	-0.6786	1
66	0.8175	-0.7805	0.6682	0.6092	-0.6381	1
67	0.8849	-0.9462	0.7831	0.8952	-0.8373	1
68	1.0000	-1.0000	1.0000	1.0000	-1.0000	1

```
""“ ECE5984 SP20 Multivariate Linear Regression
Created on Thu Feb 20 17:41:33 2020 @author: crjones4 ""“
from sklearn import linear_model as linmod
from sklearn import metrics
from sklearn import preprocessing as preproc
import pandas as pd
import numpy as np

def showStats(W, X, Yact, Ypred):
    print("R2 = %f,  MSE = %f" % (mlr.score(X, Yact), metrics.mean_squared_error(Yact, Ypred)))
    print("W: ", W)

pathName = "C:\\Data\\"
fileName = "generators.xlsx"      # read from Excel file
targetName = "GOOD"
IDName = "ID"
doScale = True
dataFrame = pd.read_excel(pathName + fileName, sheet_name='extended')
trainX = dataFrame.drop([IDName, targetName], axis=1).to_numpy()
if (doScale):
    scalerX = preproc.MinMaxScaler(feature_range=(-1, 1))
    scalerX.fit(trainX)
    trainX = scalerX.transform(trainX)
trainY = dataFrame[targetName].to_numpy()
mlr = linmod.LinearRegression()     # creates the regressor object
mlr.fit(trainX, trainY)
showStats(np.append(np.array(mlr.intercept_), mlr.coef_), trainX, trainY, mlr.predict(trainX))

poly = preproc.PolynomialFeatures(2)    # object to generate polynomial basis functions
bigTrainX = poly.fit_transform(trainX)
mlr = linmod.LinearRegression()     # creates the regressor object
mlr.fit(bigTrainX, trainY)
showStats(np.append(np.array(mlr.intercept_), mlr.coef_), bigTrainX, trainY, mlr.predict(bigTrainX))
```

```
""“ ECE5984 SP20 Multivariate Linear Regression
Created on Thu Feb 20 17:41:33 2020 @author: crjones4 ""“
from sklearn import linear_model as linmod
from sklearn import metrics
from sklearn import preprocessing as preproc
import pandas as pd
import numpy as np

def showStats(W, X, Yact, Ypred):
    print("R2 = %f,  MSE = %f" % (mlr.score(X, Yact), metrics.mean_squared_error(Yact, Ypred)))
    print("W: ", W)

pathName = "C:\\Data\\"
fileName = "generators.xlsx"      # read from Excel file
targetName = "GOOD"
IDName = "ID"
doScale = True
dataFrame = pd.read_excel(pathName + fileName, sheet_name='extended')
trainX = dataFrame.drop([IDName, targetName], axis=1).to_numpy()
if (doScale):
    scalerX = preproc.MinMaxScaler(feature_range=(-1, 1))
    scalerX.fit(trainX)
    trainX = scalerX.transform(trainX)
trainY = dataFrame[targetName].to_numpy()
mlr = linmod.LinearRegression()     # creates the regressor object
mlr.fit(trainX, trainY)
showStats(np.append(np.array(mlr.intercept_), mlr.coef_), trainX, trainY, mlr.predict(trainX))

poly = preproc.PolynomialFeatures(2)    # object to generate polynomial basis functions
bigTrainX = poly.fit_transform(trainX)
mlr = linmod.LinearRegression()     # creates the regressor object
mlr.fit(bigTrainX, trainY)
showStats(np.append(np.array(mlr.intercept_), mlr.coef_), bigTrainX, trainY, mlr.predict(bigTrainX))
```

R2 = 0.742461, MSE = 0.064385
('W: ', array([0.56690484, -0.82127102, -1.22379766]))

R2 = 0.750516, MSE = 0.062371
('W: ', array([0.63744767, 0.   , -0.80593851, -1.20532541, -0.21807308, -0.37487979, -0.31797449]))

Does including the 2nd order basis functions increase performance?

With the original dataset:

- $\mathbb{M}_W(\boldsymbol{d}) = 0.5669 - 0.8213 \cdot RPM_{nm} - 1.2238 \cdot VIBR_{nm}$
- On training set: $R^2 = 0.742461$, MSE = 0.064385

With 2nd order basis functions

- $\mathbb{M}_W(\boldsymbol{d}) = 0.6374 - 0.8059 \cdot RPM_{nm} - 1.2053 \cdot VIBR_{nm}$
  $-0.218 \cdot RPM_{nm}{}^2 - 0.3749 \cdot RPM_{nm} \cdot RPM_{nm} - 0.3179 \cdot RPM_{nm}$
- On training set: $R^2 = 0.750516$, MSE = 0.062371

Let's evaluate on a larger dataset (ccpp.xlsx)

- 6000 instances in training set, 3568 instances in test set

ID	AT	V	AP	RH	PE
1	14.96	41.76	1024.07	73.17	463.26
2	25.18	62.96	1020.04	59.08	444.37
3	5.11	39.4	1012.16	92.14	488.56
…	…	…	…	…	…

Using base features (AT, V, AP and RH) to predict PE:

- Training set: $R^2 = 0.930531$, MSE = 20.206218
- Test set: $R^2 = 0.931604$, MSE = 20.158957

Using 2nd order polynomial basis functions:

- $R^2 = 0.939488$, MSE = 17.600797
- $R^2 = 0.940909$, MSE = 17.416263

So, without knowledge of what basis functions to try, are there any general guidelines?

Popular choices for basis functions are:

- square: $\phi(x) = x^2$
- logarithm: $\phi(x) = \log(x)$
- magnitude: $\phi(x) = |x|$
- products of two features: $\phi(x, y) = xy$
  - Use with discretion – the dataset will get large
- Gaussian, Fourier, Laguerre…
- splines: see –
  - Friedman, J. (1991). Multivariate Adaptive Regression Splines. *The Annals of Statistics, 19*(1), 1-67.

What if we cannot define a nice decision boundary, even with use of basis functions?

- Perhaps the problem is poorly posed
- See if more features are available
- Check for data errors
- Use a more complex classifier tool
  - Support Vector Machines
  - we will discuss them briefly in later lectures

INTERPRETING LINEAR REGRESSION MODELS

The weights used by linear regression models indicate the effect of each descriptive feature on the predictions returned by the model

- The sign and the magnitude of the weight provide information on how the descriptive feature effects the predictions of the model.

Descriptive Feature	Weight	Standard Error	t-statistic	p-value
SIZE	0.6270	0.0545	11.504	<0.0001
FLOOR	-0.1781	2.7042	-0.066	0.949
BROADBAND RATE	0.071396	0.2969	0.240	0.816

- The sign can be interpreted directly – *sometimes*
  - The weight on the FLOOR variable is negative – higher floors command less rent
  - Even this can be misleading if features have significant correlation

- We can perform statistical testing to determine the importance of the various features

We can use the t-test to determine the importance of a descriptive feature $d[j]$ in a linear regression model

- The null hypothesis for this test is that the feature does not have a significant impact on the model. The test statistic we calculate is called the t-statistic.
  - See https://blog.minitab.com/en/statistics-and-quality-data-analysis/what-are-t-values-and-p-values-in-statistics
- The standard error can be computed as $se(\mathbb{M}) = \sqrt{\frac{\sum_{i=1}^{n}(t_i - \mathbb{M}_{\boldsymbol{w}}(\boldsymbol{d}_i))^2}{n-2}}$
- For a given feature, we calculate its se as $se(\boldsymbol{d}[j]) = \frac{se(\mathbb{M})}{\sqrt{\sum_{i=1}^{n}(\boldsymbol{d}[j] - \mu_{\boldsymbol{d}[j]})^2}}$
- The t-statistic for that feature is then $t = \frac{\boldsymbol{w}[\boldsymbol{j}]}{se(\boldsymbol{d}[j])}$

The t-test will reject variables that do not have statistically significant influence on the target variable

- Using a standard t-statistic look-up table, we can then determine the p-value associated with this test (this is a two tailed t-test with degrees of freedom set to the number of instances in the training set minus 2).
- If the p-value is less than the required significance level, typically 0.05, we reject the null hypothesis and say that the descriptive feature has a significant impact on the model; otherwise we say that it does not.

Descriptive Feature	Weight	Standard Error	t-statistic	p-value	
Size	0.6270	0.0545	11.504	<0.0001	SIGNIFICANT
Floor	-0.1781	2.7042	-0.066	0.949	NOT SIGNIFICANT
Broadband Rate	0.071396	0.2969	0.240	0.816	NOT SIGNIFICANT

Here is the report output from a multivariate linear regression in SAS

- www.sas.com
- For each variable, the routine reports:
  - The coefficient estimate
  - The standard error
  - the t-value
  - the final metric $Pr > |t|$
- In this example, only the Height is statistically significant (Age is not)
  - Assuming we use 0.05 as our p-value

Model: MODEL1
Dependent Variable: Weight

Number of Observations Read	19
Number of Observations Used	19

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	7215.63710	3607.81855	27.23	<.0001
Error	16	2120.09974	132.50623		
Corrected Total	18	9335.73684			

Root MSE	11.51114	R-Square	0.7729
Dependent Mean	100.02632	Adj R-Sq	0.7445
Coeff Var	11.50811		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-141.22376	33.38309	-4.23	0.0006
Age	1	1.27839	3.11010	0.41	0.6865
Height	1	3.59703	0.90546	3.97	0.0011

So, what does this mean? If a variable is found to be not statistically significant, do we leave it out?

- Maybe
- Maybe not
- If it increases the accuracy and generalization of the model, then use it
- High p-value tells me three things:
  - This variable may be cause of overfitting (note, I said may)
  - This variable may be a candidate for removal (more on this later)
  - This variable is not, by itself, a good predictor (so don't draw conclusions)

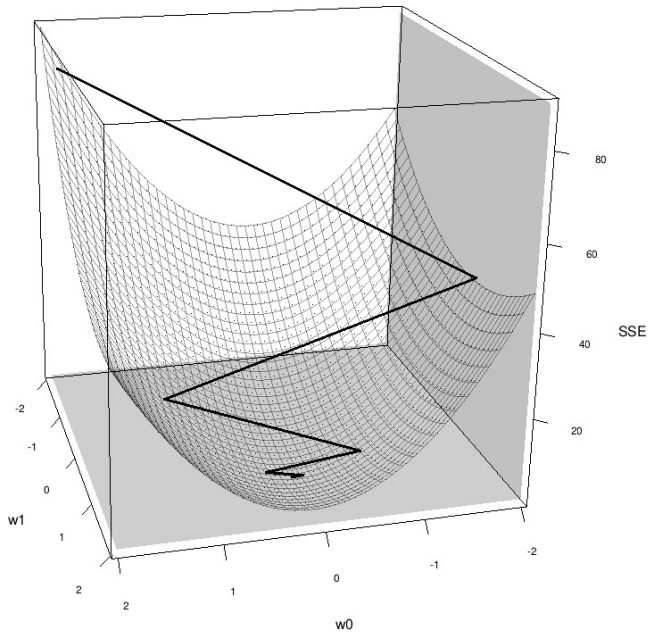
WEIGHT DECAY

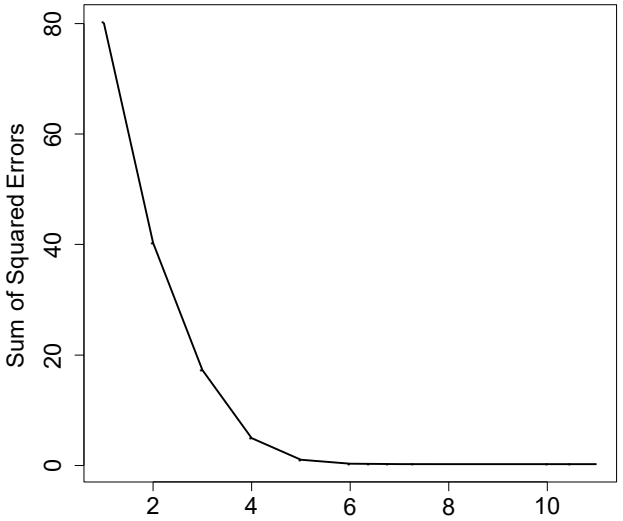
Last time, we mentioned that adaptive learning rates can be useful in gradient descent methods

- Learning rate decay allows the learning rate to start at a large value and then decay over time according to a predefined schedule.
- A good approach is to use the following decay schedule:

$$\alpha_\tau = \alpha_0 \frac{c}{c + \tau}$$

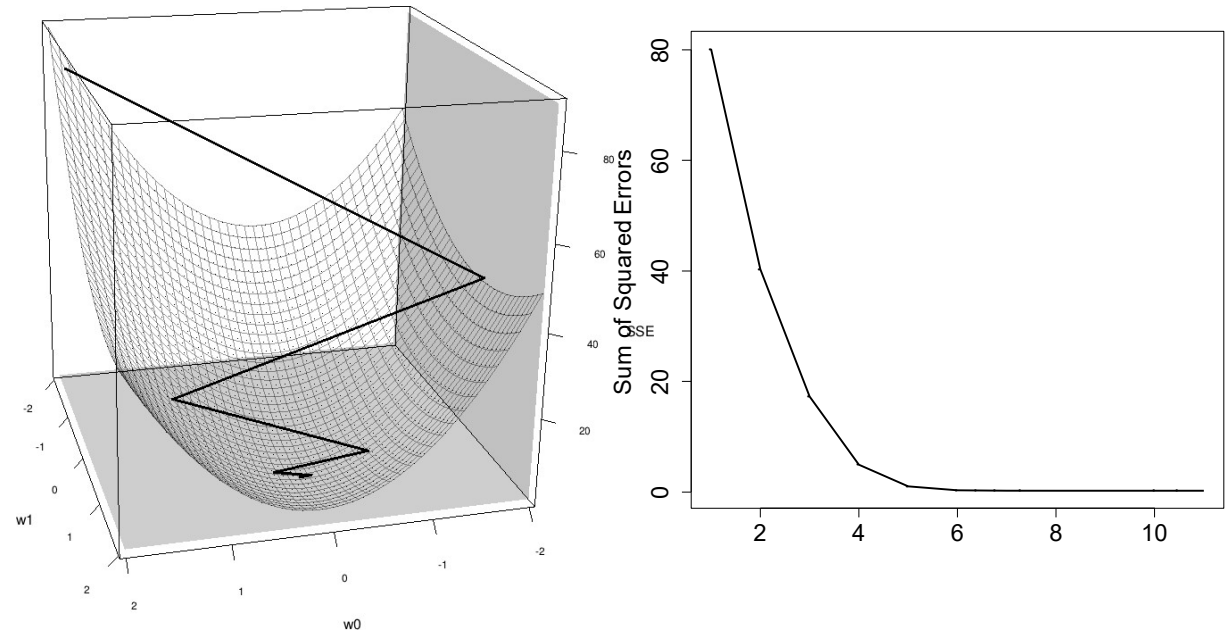
- Of course, we need to choose α_0, the initial rate, and c, the rate change



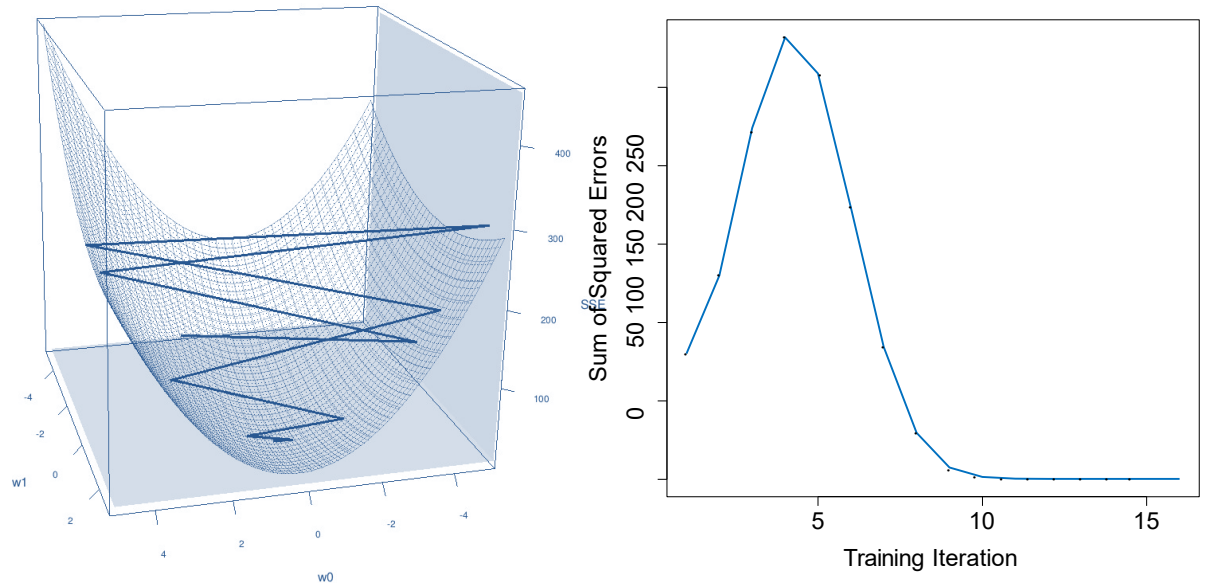


(a) The journey across the error surface for the office rent prediction problem when learning rate decay is used ($\alpha_0 = 0.18$, $c = 10$);
(b) the changing sum of squared error values

Rates that are too large will cause erratic training; rates that are too small will take too long to converge



(a) The journey across the error surface for the office rent prediction problem when learning rate decay is used ($\alpha_0 = 0.18$, $c = 10$);
(b) the changing sum of squared error values



(a) The journey across the error surface for the office rent prediction problem when learning rate decay is used ($\alpha_0 = 0.25$, $c = 100$);
(b) the changing sum of squared error values

CATEGORICAL FEATURES

The basic structure of the multivariable linear regression model allows for only continuous descriptive features, so we need a way to handle categorical descriptive features

- The most common approach to handling categorical features uses a transformation that converts a single categorical descriptive feature into a number of continuous descriptive feature values that can encode the levels of the categorical feature.
- The ENERGY RATING descriptive feature would be converted into three new continuous descriptive features, for its 3 distinct levels: ’A’, ’B’, ’C’.
  - In essence, we apply one-hot encoding to the categorical variable
  - Binary predictors in a regression model are acceptable – though often not as useful as continuous variables.

Replacing the categorical ENERGY RATING with three binary variables, ER_A, ER_B and ER_C…

ID	Size	Floor	Broadband Rate	*Energy Rating*	ER_A	ER_B	ER_C	Rental Price
1	500	4	8	*C*	0	0	1	320
2	550	7	50	*A*	1	0	0	380
3	620	9	7	*A*	1	0	0	400
4	630	5	24	*B*	0	1	0	390
5	665	8	100	*C*	0	0	1	385
6	700	4	8	*B*	0	1	0	410
7	770	10	7	*B*	0	1	0	480
8	880	12	50	*A*	1	0	0	600
9	920	14	8	*C*	0	0	1	570
10	1,000	9	24	*B*	0	1	0	620

```
ECE5984 SP20 Multivariate Linear Regression
Created on Thu Feb 20 17:41:33 2020

@author: crjones4
"""
from sklearn import preprocessing as preproc
from sklearn import linear_model as linmod
from sklearn import metrics
import pandas as pd
import numpy as np

pathName = "C:\\Data\\"
fileName = "DublinRental.xlsx"      # read from Excel file
targetName = "PRICE"
IDName = "ID"
catName = "ENERGY"
dataFrame = pd.read_excel(pathName + fileName, sheet_name='train2')     # with Energy binaries
trainX = dataFrame.drop([IDName, catName, targetName], axis=1).to_numpy()
trainY = dataFrame[targetName].to_numpy()
mlr = linmod.LinearRegression()     # creates the regressor object
mlr.fit(trainX, trainY)
print("R2 is %f" % mlr.score(trainX, trainY))
print("W = ", mlr.intercept_, mlr.coef_)
query = np.array([[600,6,1,0,0,20]])
print("prediction:", query, mlr.predict(query))
print(metrics.mean_squared_error(trainY, mlr.predict(trainX)))
```

```
R2 is 0.982786
('W = ', -4.469,
   array([ 0.643,  0.0167, -0.1325,  29.55, -17.00, -12.54]))
('prediction:', array([[600,   6,   1,   0,   0,  20]]),
array([130.49601854]))
172.35567222924
```

Did the inclusion of the one-hot encoded Energy variable improve the model?

With ER_A, ER_B and ER_C:

- R^2 = 0.982786
- W = [-4.469, 0.643, 0.0167, -0.1325, 29.55, -17.00, -12.54]
- M([600, 6, 20, 1, 0, 0]) = 408.423
- MSE = 172.35

Without ER_A, ER_B and ER_C:

- R^2 is 0.955209
- W = [19.562, 0.549, 4.964, -0.062]
- M([600, 6, 20]) = 377.345
- MSE = 448.456

Today's Objectives

Non-linear relationships

- Basis functions

More on Linear Regression

- Interpreting linear regression models
- Weight decay
- Categorical features