

ECE5984 – Applications of Machine Learning

Lecture 18 – Logistic Regression

Creed Jones, PhD

Course update

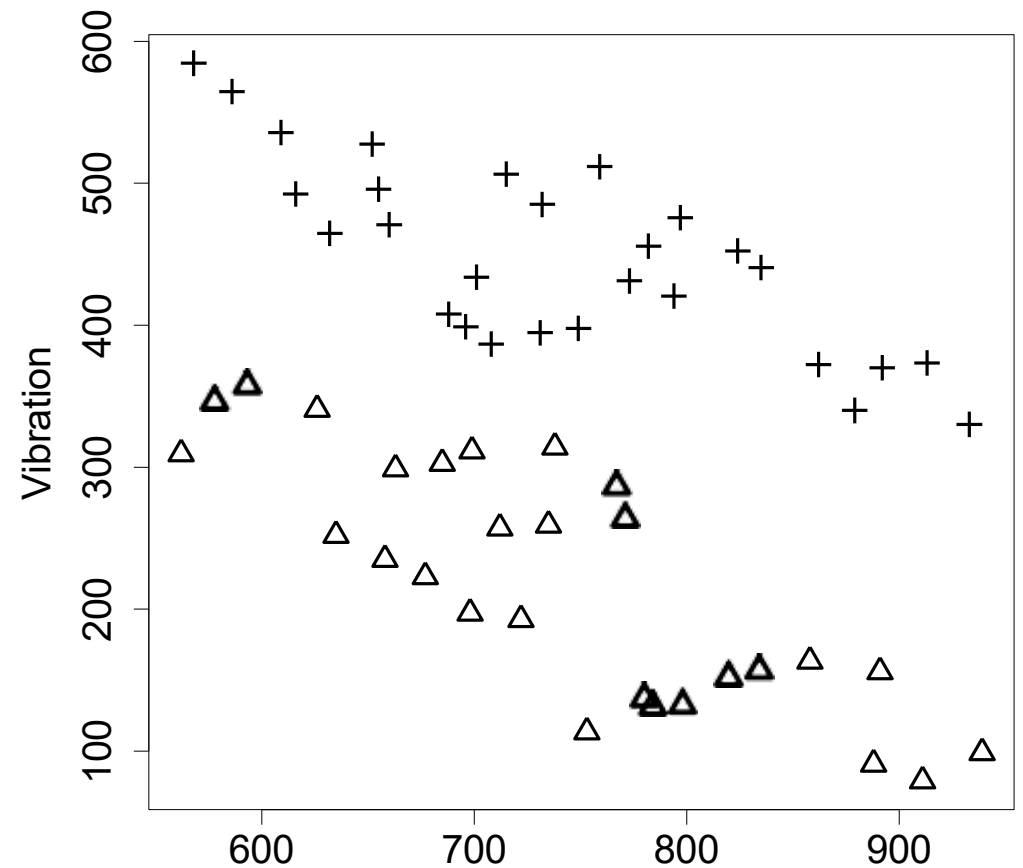
- HW 4 is posted
 - April 5
- Quiz TODAY
 - Lectures 14-17
- Project II Teams
 - If you want to make any changes, or have any concerns, please email me before next Monday, March 28

Today's Objectives

- Logistic Regression

LOGISTIC REGRESSION

ID	RPM	VIBR	STATUS	ID	RPM	VIBR	STATUS
1	568	585	good	29	562	309	faulty
2	586	565	good	30	578	346	faulty
3	609	536	good	31	593	357	faulty
4	616	492	good	32	626	341	faulty
5	632	465	good	33	635	252	faulty
6	652	528	good	34	658	235	faulty
7	655	496	good	35	663	299	faulty
8	660	471	good	36	677	223	faulty
9	688	408	good	37	685	303	faulty
10	696	399	good	38	698	197	faulty
11	708	387	good	39	699	311	faulty
12	701	434	good	40	712	257	faulty
13	715	506	good	41	722	193	faulty
14	732	485	good	42	735	259	faulty
15	731	395	good	43	738	314	faulty
16	749	398	good	44	753	113	faulty
17	759	512	good	45	767	286	faulty
18	773	431	good	46	771	264	faulty
19	782	456	good	47	780	137	faulty
20	797	476	good	48	784	131	faulty
21	794	421	good	49	798	132	faulty
22	824	452	good	50	820	152	faulty
23	835	441	good	51	834	157	faulty
24	862	372	good	52	858	163	faulty
25	879	340	good	53	888	91	faulty
26	892	370	good	54	891	156	faulty
27	913	373	good	55	911	79	faulty
28	933	330	good	56	939	99	faulty

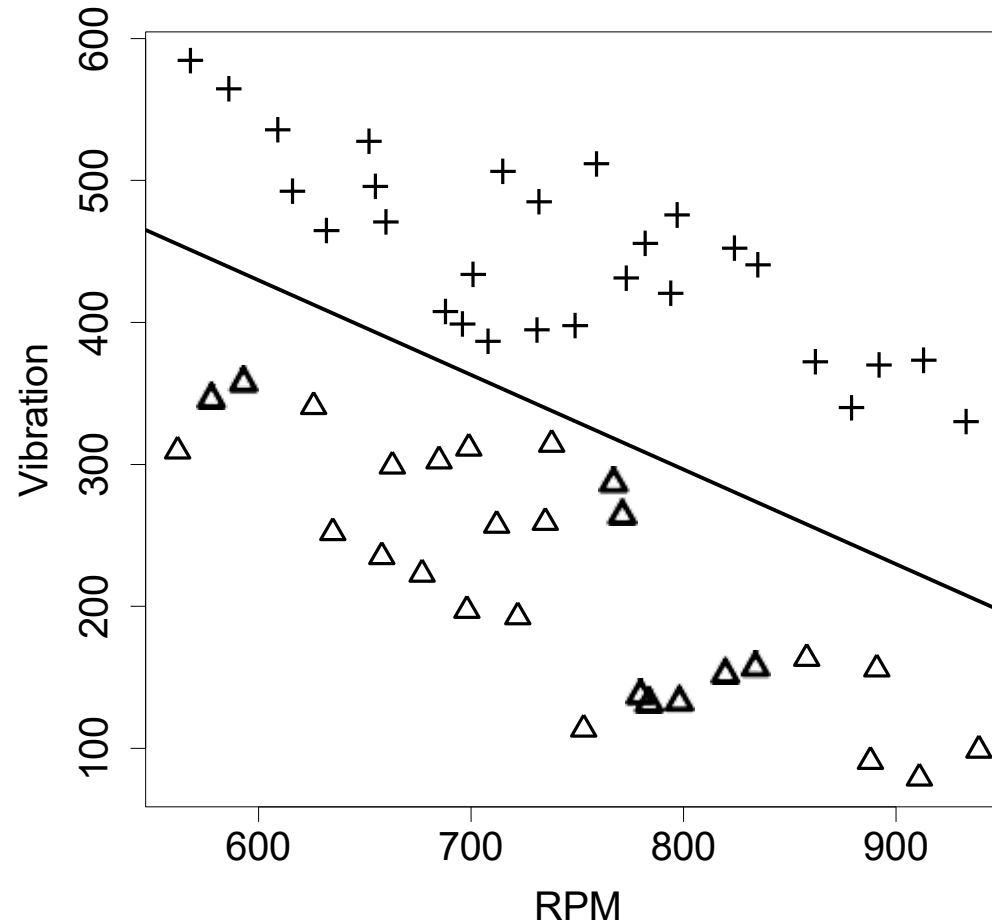


A scatter plot of the RPM and VIBRATION descriptive features from the generators dataset shown in the Table where 'good' generators are shown as crosses and 'faulty' generators are shown as triangles.

The line shown is:

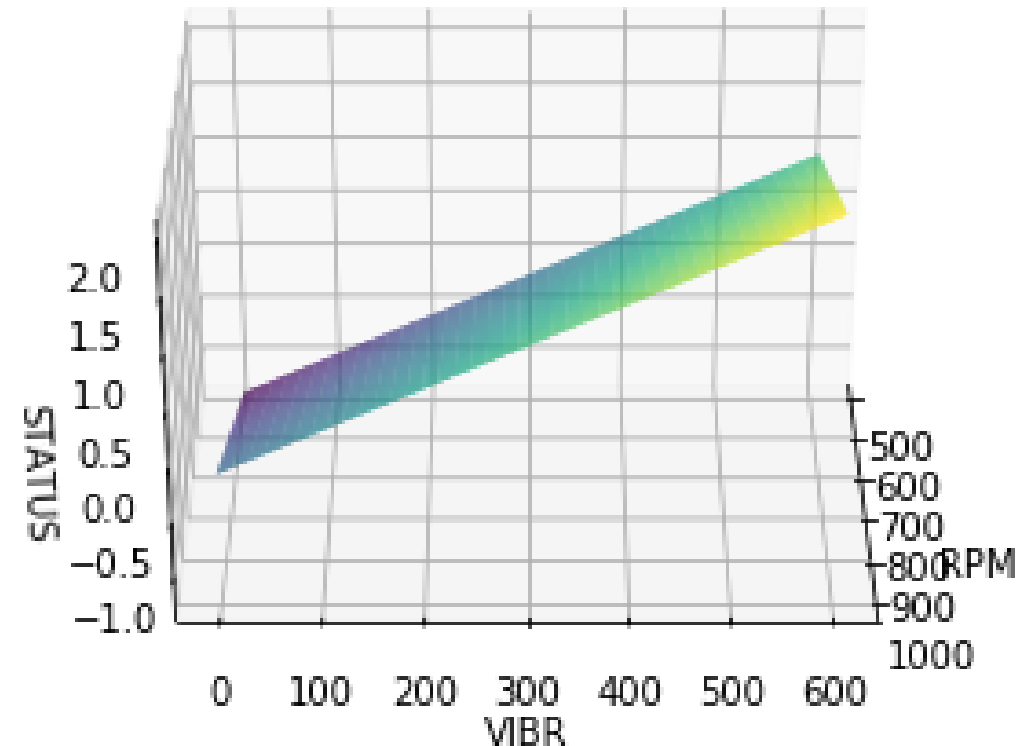
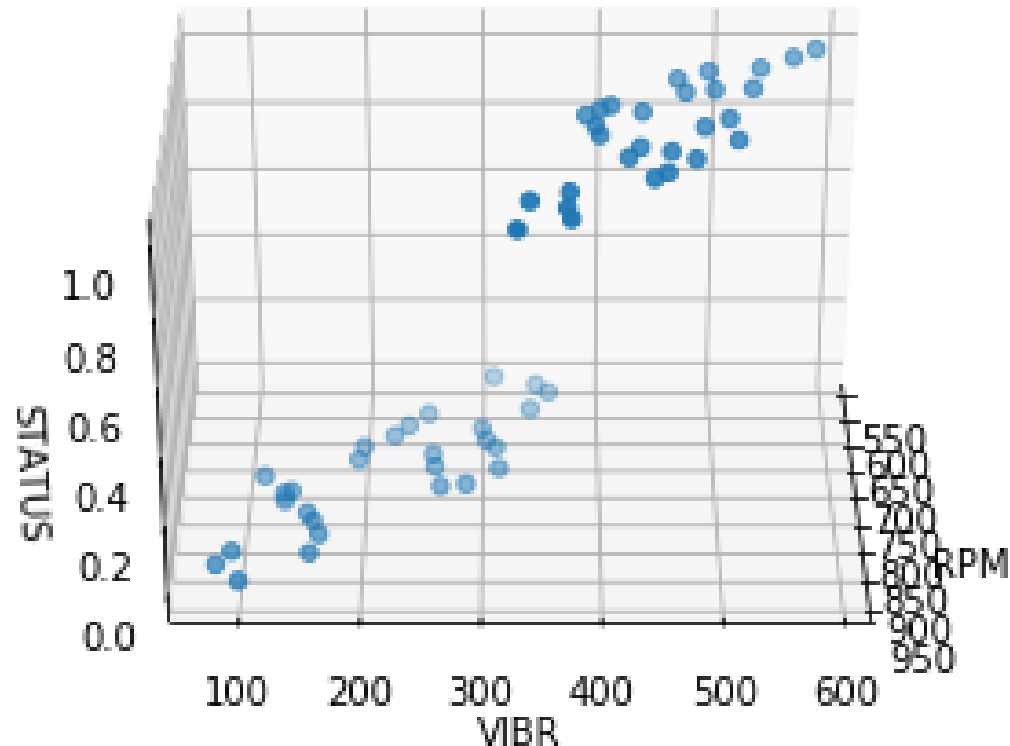
$$VIBR = -0.667 \cdot RPM + 830$$

Good generators are above the line; bad ones are below it (in this plot).



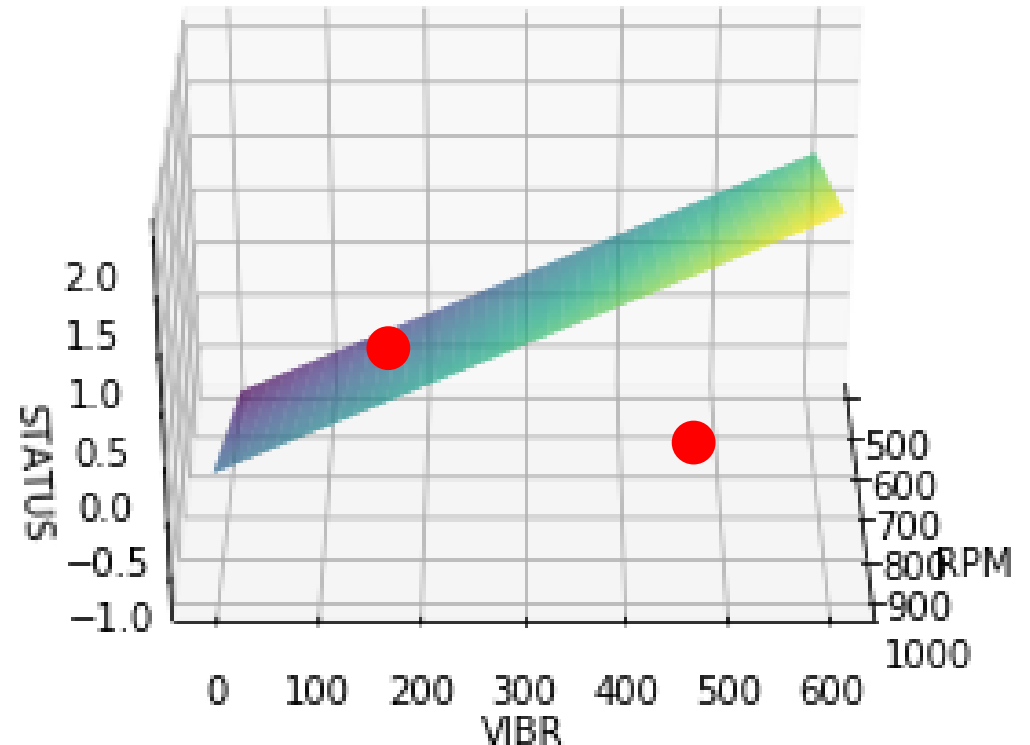
A scatter plot of the RPM and VIBRATION descriptive features from the generators dataset shown in Table 4^[18]. A decision boundary separating 'good' generators (crosses) from 'faulty' generators (triangles) is also shown.

Since the generator dataset has only two features, we can plot the data and the linear decision boundary; the plane is defined by $830 - 0.667 \cdot RPM - VIBR = 0$



Using the decision boundary equation, we can classify query inputs: $cl(r, v) = 830 - 0.667 \cdot r - v$

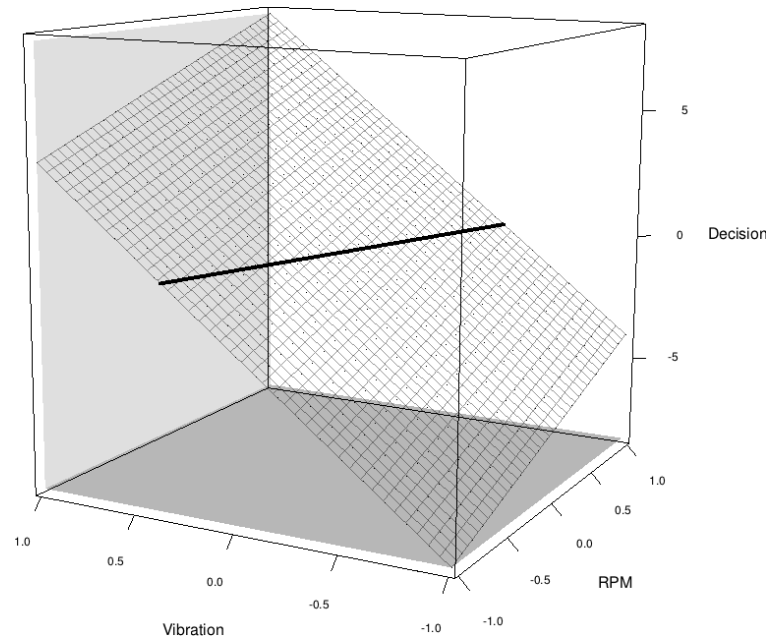
- Consider the instance RPM = 810, VIBRATION = 495 (which is “above” the decision boundary):
- $cl(810, 495) = 830 - 0.667 \cdot 810 - 495 = -205.27$
- This positive result is interpreted as CL=1
- Alternatively, if RPM = 650 and VIBRATION = 240 (“below” the decision boundary), we get:
- $cl(650, 240) = 830 - 0.667 \cdot 650 - 240 = 156.45$
- This positive result is interpreted as CL=0



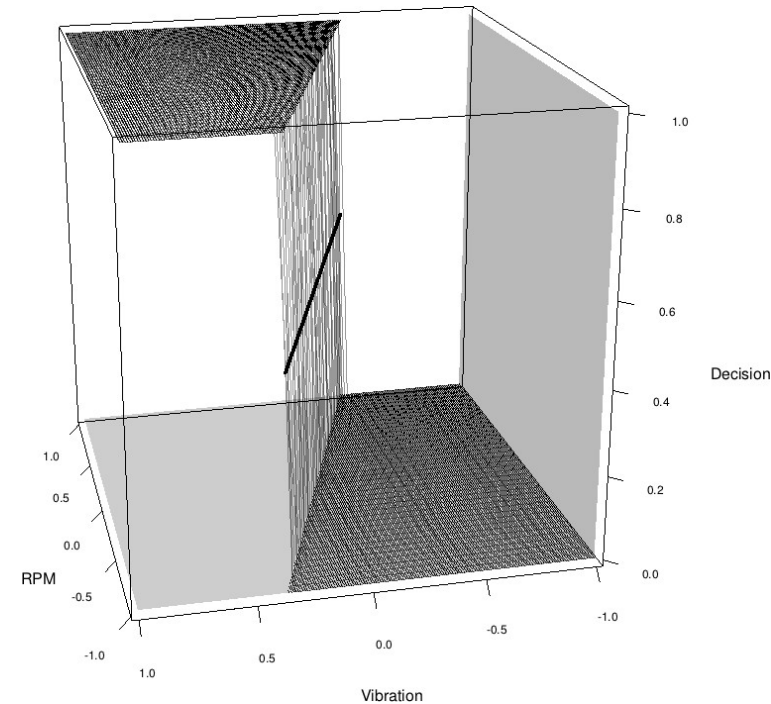
Linear regression creates a planar *decision surface*; in high dimensions, the surface is a hyperplane

- $cl(810, 495) = 830 - 0.667 \cdot 810 - 495 = -205.27$
 - This positive result is interpreted as CL=1
- $cl(650, 240) = 830 - 0.667 \cdot 650 - 240 = 156.45$
 - This positive result is interpreted as CL=0
- All the data points above the decision boundary result in a negative value when plugged into the decision boundary equation, while all data points below the decision boundary result in a positive value.
- The model is then defined by a *decision surface*:
$$\mathbb{M}_w(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{d} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The model $\mathbb{M}_w(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{d} \geq 0 \\ 0 & \text{otherwise} \end{cases}$ is the planar decision surface with a hard threshold applied



(a)



(b)

- (a) The decision surface for $cl(r, v) = 830 - 0.667 \cdot r - v$.
- (b) The same surface linearly thresholded at zero to operate as a predictor.

Applying a simple threshold to the dot product is not mathematically convenient; we would like an alternative

- The hard decision boundary given by thresholding the plane is discontinuous and not differentiable; the gradient of the error surface can't be calculated.
- Furthermore, and more importantly, the output is always 1 or 0; this doesn't give any information about how close a query is to the decision boundary.
- We address these issues by using a more sophisticated threshold function that is continuous, and therefore differentiable, and that allows for the subtlety desired - the logistic function:

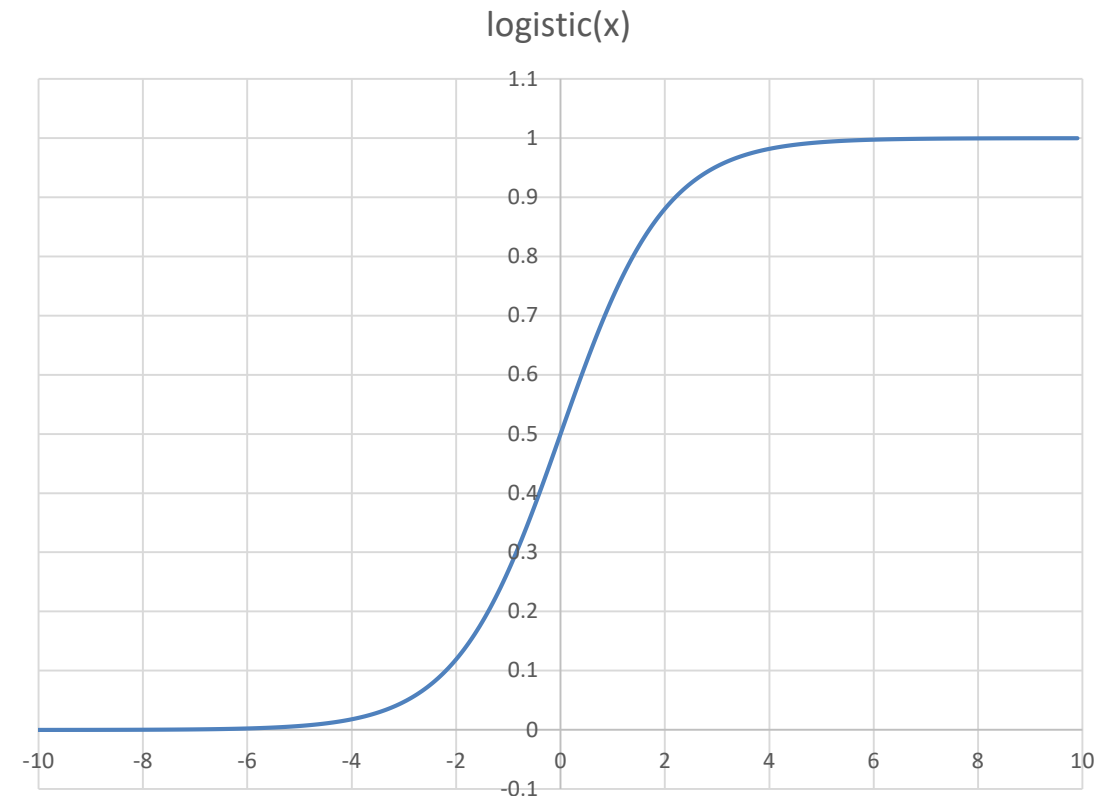
$$\text{Logistic}(x) = \frac{1}{1 + e^{-x}}$$

The logistic function $Logistic(x) = \frac{1}{1+e^{-x}}$ has domain $[-\infty, \infty]$ and range $[0,1]$

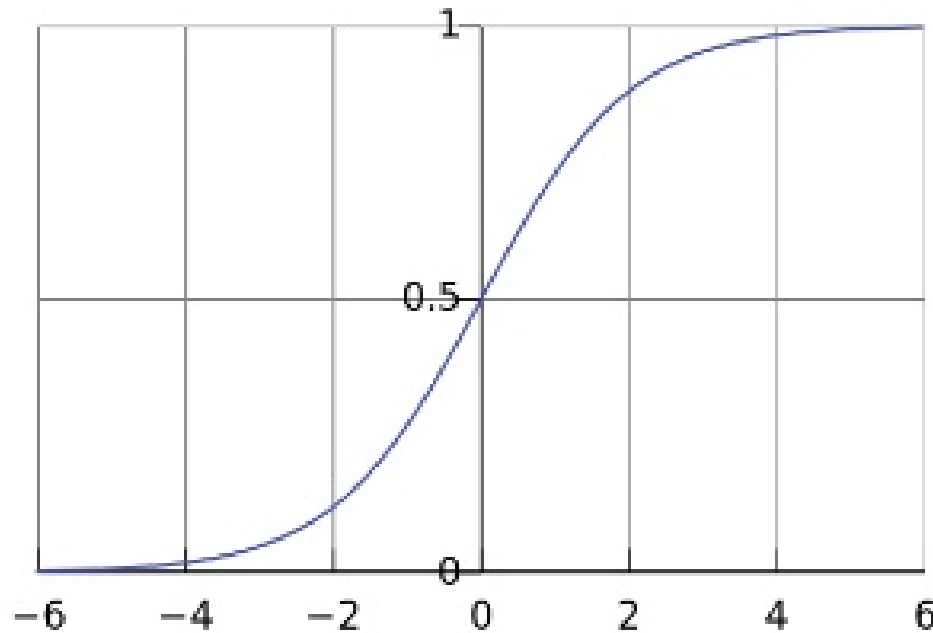
- To build a logistic regression model, we simply pass the output of the basic linear regression model through the logistic function

$$\mathbb{M}_w(\mathbf{d}) = \frac{1}{1 + e^{-w \cdot \mathbf{d}}}$$

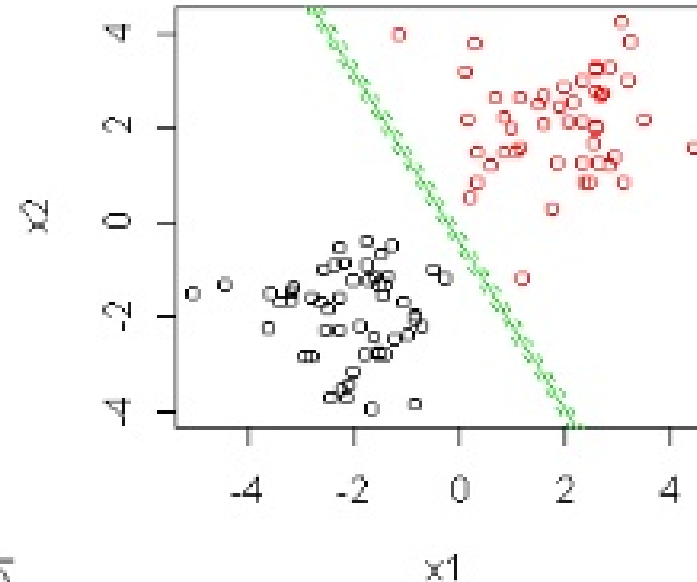
- Before we train a logistic regression model we map the binary target feature levels to 0 or 1



Binary Logistic Regression



$$d = \frac{ax_1 + bx_2 + cx_0}{\sqrt{a^2 + b^2}} \quad \text{where } x_0 = 1$$



$$P(y=1|\vec{x}, \vec{w}) = \frac{\exp(d)}{1 + \exp(d)} = \frac{\exp(\vec{x} \vec{w})}{1 + \exp(\vec{x} \vec{w})}$$

$$P(y=0|\vec{x}, \vec{w}) = \frac{1}{1 + \exp(\vec{x} \vec{w})}$$

$$\log \frac{P(y=1|\vec{x}, \vec{w})}{P(y=0|\vec{x}, \vec{w})} = \vec{x} \vec{w}$$

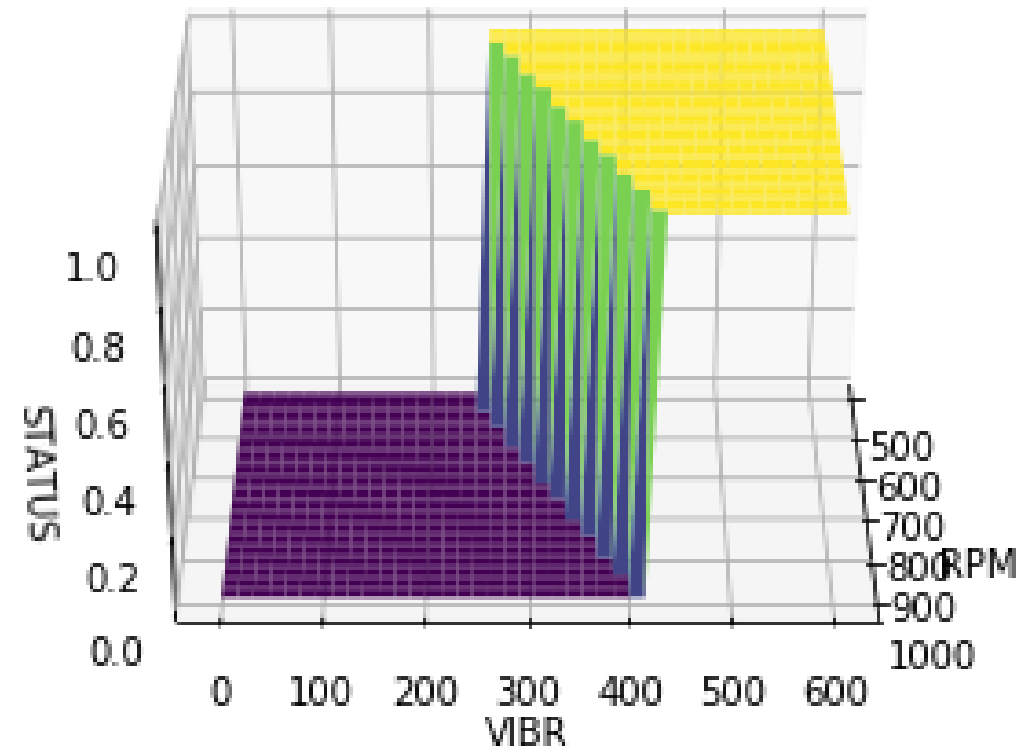
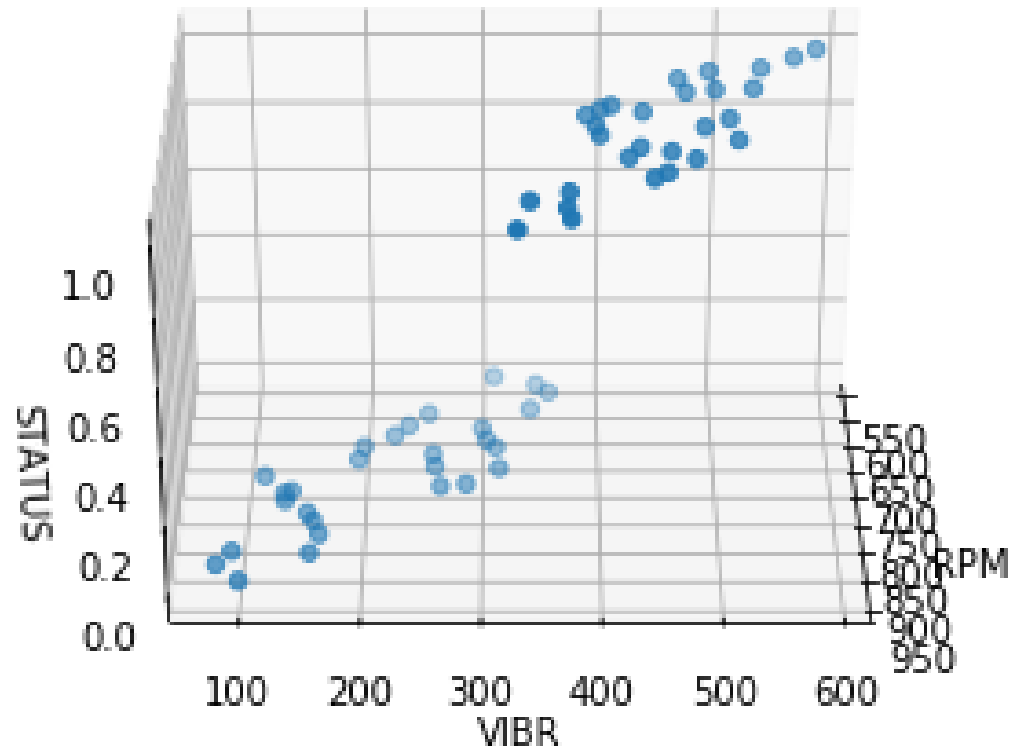
$$w_0 = \frac{c}{\sqrt{a^2 + b^2}} \quad \text{where } w_0 \text{ is called as intercept}$$

$$w_1 = \frac{a}{\sqrt{a^2 + b^2}}$$

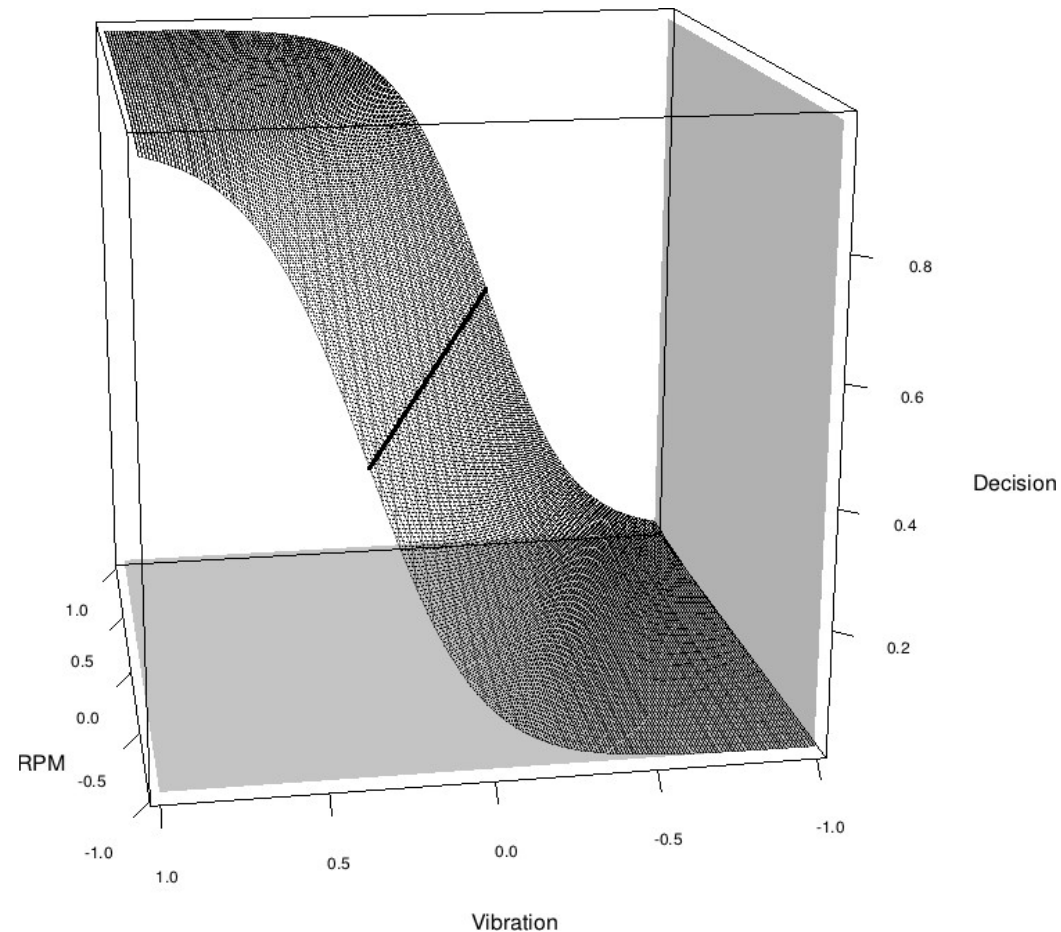
$$w_2 = \frac{b}{\sqrt{a^2 + b^2}}$$

The logistic model for the generators is

$$M_w(d) = \frac{1}{1 + e^{-(-0.4077 + 4.1697 \cdot RPM + 6.0460 \cdot VIBR)}}$$



Here is a better look at the decision surface for the logistic model $M_w(d) = \frac{1}{1+e^{-(-0.4077+4.1697 \cdot RPM+6.0460 \cdot VIBR)}}$



The gradient descent algorithm is still used for training a logistic regression classifier – with a modification to the update rule

- See pg. 358 in book for details of how to derive the new weight update rule
- The algorithm is now:
 1. Set $\mathbf{w} \leftarrow \text{random starting point}$
 2. repeat
 1. for each $\mathbf{w}[j]$ in \mathbf{w} do
 1. Set $\mathbf{w}[j] \leftarrow \mathbf{w}[j] + \alpha \cdot \sum_{i=1}^n \left((t - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)) \cdot \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i) \cdot (1 - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)) \cdot \mathbf{d}_i[j] \right)$
 2. end for
 3. until convergence occurs

We add some new samples to the generator dataset –
the classes are no longer separable!



The extended dataset is first normalized to $[-1, 1]$

- For logistic regression models descriptive feature values should always be normalized
- In this example, both descriptive features are normalized to the range $[-1, 1]$

ID	RPM	VIBR	GOOD	NORM RPM	NORM VIBR
1	498	604	0	-1.0000	1.0000
2	517	594	0	-0.9246	0.9586
3	541	574	0	-0.8294	0.8758
4	555	587	0	-0.7738	0.9296
5	572	537	0	-0.7063	0.7226
6	600	553	0	-0.5952	0.7888
7	621	482	0	-0.5119	0.4948
8	632	539	0	-0.4683	0.7308
...
62	878	168	1	0.5079	-0.8054
63	895	218	1	0.5754	-0.5983
64	916	221	1	0.6587	-0.5859
65	950	156	1	0.7937	-0.8551
66	956	174	1	0.8175	-0.7805
67	973	134	1	0.8849	-0.9462
68	1002	121	1	1.0000	-1.0000

sklearn.linear_model.LogisticReg

+

← → ↺

scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

☆ 🔍 ⚙️ 👤

Apps

VT gmail

VT cal

ECE internal

ECE

W W

OneCampus

Canvas

IEEE

IEEE

BBC

Ms

VT

CITI

CVL Wiki

Library Commi

Readin

scikit-learn

Install User Guide API Examples More ▾

Prev Up Next

scikit-learn 0.24.1

Other versions

Please [cite us](#) if you use the software.

sklearn.linear_model.LogisticRegression

Examples using sklearn.linear_model.LogisticRegression

sklearn.linear_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

[source]

Logistic Regression (aka logit, MaxEnt) classifier.


In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default.** It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).


The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

Read more in the [User Guide](#).

Parameters:	<p>penalty : {'l1', 'l2', 'elasticnet', 'none'}, default='l2' Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties. 'elasticnet' is only supported by the 'saga' solver. If 'none' (not supported by the liblinear solver), no regularization is applied.</p> <p><i>New in version 0.19:</i> l1 penalty with SAGA solver (allowing 'multinomial' + L1)</p> <p>dual : bool, default=False Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samples > n_features.</p>
--------------------	--



VIRGINIA TECH.™



BRADLEY DEPARTMENT
OF ELECTRICAL
& COMPUTER
ENGINEERING

ECE5984 SP22 18 - Logistic Regression

19

```

# 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

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()
trainY = dataFrame[targetName].to_numpy()
if (doScale):
    scalerX = preproc.MinMaxScaler(feature_range=(-1, 1))
    scalerX.fit(trainX)
    trainX = scalerX.transform(trainX)
mlr = linmod.LogisticRegression(tol=1e-6)      # creates the regressor object - note the lower tolerance!
mlr.fit(trainX, trainY)
Ypred = mlr.predict(trainX)
Ypredclass = 1*(Ypred > 0.5)
print("R2 = %f, MSE = %f, Classification Accuracy = %f" %
      (metrics.r2_score(testY, Ypred), metrics.mean_squared_error(testY, Ypred), metrics.accuracy_score(testY, Ypredclass)))
print("W: ", np.append(np.array(mlr.intercept_), mlr.coef_))

poly = preproc.PolynomialFeatures(2)  # object to generate polynomial basis functions
trainX = dataFrame.drop([IDName, targetName], axis=1).to_numpy()
bigTrainX = poly.fit_transform(trainX)
if (doScale):
    scalerX = preproc.MinMaxScaler(feature_range=(-1, 1))
    scalerX.fit(bigTrainX)
    bigTrainX = scalerX.transform(bigTrainX)
mlrf = linmod.LogisticRegression()      # creates the regressor object
mlrf.fit(bigTrainX, trainY)
Ypred = mlrf.predict(bigTrainX)
Ypredclass = 1*(Ypred > 0.5)
print("R2 = %f, MSE = %f, Classification Accuracy = %f" %
      (metrics.r2_score(testY, Ypred), metrics.mean_squared_error(testY, Ypred), metrics.accuracy_score(testY, Ypredclass)))
print("W: ", np.append(np.array(mlr.intercept_), mlr.coef_))

```

```
# 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
```

```
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()
trainY = dataFrame[targetName].to_numpy()
if (doScale):
    scalerX = preproc.MinMaxScaler(feature_range=(-1, 1))
    scalerX.fit(trainX)
    trainX = scalerX.transform(trainX)
mlr = linmod.LogisticRegression(tol=1e-6)    # creates the regressor object - note the lower tolerance!
mlr.fit(trainX, trainY)
Ypred = mlr.predict(trainX)
Ypredclass = 1*(Ypred > 0.5)
print("R2 = %f, MSE = %f, Classification Accuracy = %f" %
      (metrics.r2_score(testY, Ypred), metrics.mean_squared_error(testY, Ypred), metrics.accuracy_score(testY, Ypredclass)))
print("W: ", np.append(np.array(mlr.intercept_), mlr.coef_))
```

```
poly = preproc.PolynomialFeatures(2)    # object to generate polynomial basis functions
```

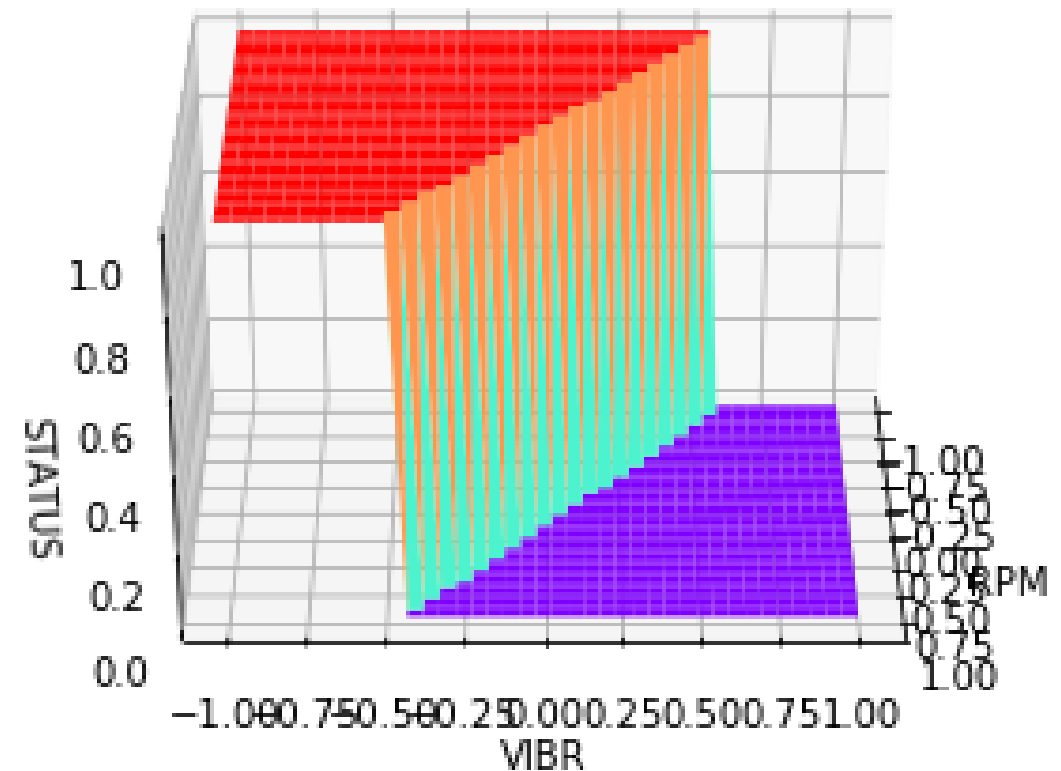
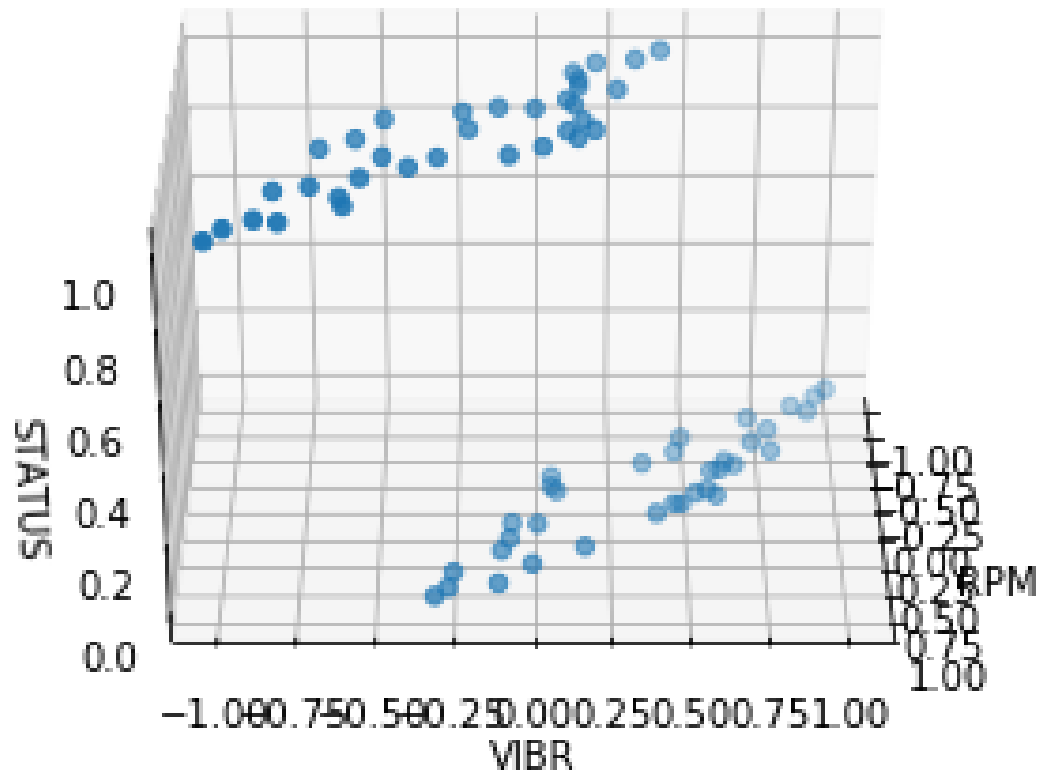
```
trainX = dataFrame.drop([IDName, targetName], axis=1).to_numpy()
bigTrainX = poly.fit_transform(trainX)
if (doScale):
    scalerX = preproc.MinMaxScaler(feature_range=(-1, 1))
    scalerX.fit(bigTrainX)
    bigTrainX = scalerX.transform(bigTrainX)
mlrf = linmod.LogisticRegression()    # creates the regressor object
mlrf.fit(bigTrainX, trainY)
Ypred = mlrf.predict(bigTrainX)
Ypredclass = 1*(Ypred > 0.5)
print("R2 = %f, MSE = %f, Classification Accuracy = %f" %
      (metrics.r2_score(testY, Ypred), metrics.mean_squared_error(testY, Ypred), metrics.accuracy_score(testY, Ypredclass)))
print("W: ", np.append(np.array(mlr.intercept_), mlr.coef_))
```

```
R2 = 0.809091, MSE = 0.047619, Classification Accuracy = 0.952381
W: [ 0.18708905 -1.2361221 -2.75014464]
R2 = 0.809091, MSE = 0.047619, Classification Accuracy = 0.952381
W: [-6.49174918e-03 -6.60528030e-06 -7.58941084e-01 -
1.18878150e+00
-8.24599317e-01 -2.02229111e+00 -1.30846715e+00]
```

The resulting model

$$M_w(d) = \frac{1}{1 + e^{-(0.1871 - 1.2361 \cdot RPM_{nm} - 2.7501 \cdot VIBR_{nm})}}$$

will not perfectly separate the classes (MSE = 0.0476)



In this case, using polynomial features doesn't give any measurable improvement (frequently happens with logistic regression)

Raw features only

- $R^2 = 0.809091$, $MSE = 0.047619$, Classification Accuracy = 0.952381
- W : [0.18708905 -1.2361221 -2.75014464]

$$\mathbb{M}_w(\mathbf{d}) = \frac{1}{1 + e^{-(0.1871 - 1.2361 \cdot RPM_{nm} - 2.7501 \cdot VIBR_{nm})}}$$

With polynomial, degree ≤ 2

- $R^2 = 0.809091$, $MSE = 0.047619$, Classification Accuracy = 0.952381
- W : [-6.49174918e-03 -6.60528030e-06 -7.58941084e-01 -1.18878150e+00
-8.24599317e-01 -2.02229111e+00 -1.30846715e+00]

$$\mathbb{M}_w(\mathbf{d}) = \frac{1}{1 + e^{-(-0.00649 - 0.75894 \cdot RPM_{nm} - 1.18878 \cdot VIBR_{nm} - 0.82460 \cdot RPM_{nm}^2 - 2.02229 \cdot RPM_{nm} \cdot VIBR_{nm} - 1.30847 \cdot VIBR_{nm}^2)}}$$

So what is all this for?

- A logistic regression model is often used for binary classification
 - The output is interpreted as the probability that the sample's class is a 1
- Since the output of the model is a continuous variable on $[0, 1]$, we apply a *decision threshold* to it to obtain either 0 or 1
- In the absence of any other information, 0.5 is often used
- We can choose other decision thresholds to give us optimal performance
 - As measured by the accuracy that is most important to us

Today's Objectives

- Logistic Regression