Elements Of Data Science - F2021

Week 5: Intro to Machine Learning Models

10/11/2021

TODOs

- Readings:
    - Recommended: https://scikit-learn.org/stable/supervised_learning.html
    - Reference: PML Chapter Chap 3

- Answer and submit Quiz 5
- HW1 Due Fri Oct 15th, 11:59pm ET
- HW2 out next week

Today

- Intro to Machine Learning Models
- Various types of ML
- Linear models

- Next week:
  - One Vs. Rest For Multiclass/Multilabel Classification
  - Distance Based: kNN
  - Tree Based: Decision Tree
  - Ensembles: Bagging, Boosting, Stacking

Questions?

Environment Setup

Environment Setup

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('darkgrid')

%matplotlib inline
```

Modeling and ML

- What is a Model?
    - Specification of a mathematical (or probabilistic) relationship between different variables.

- What is Machine Learning?
    - Creating and using models that are learned from data.

Questions for Models

Questions for Models

In [2]:
```
df_wine = pd.read_csv('../data/wine_dataset.csv',usecols=['alcohol','ash','proline','hue','class'])
df_wine.sample(4,random_state=1)
```

Out[2]:

	alcohol	ash	hue	proline	class
161	13.69	2.54	0.96	680.0	2
117	12.42	2.19	1.06	345.0	1
19	13.64	2.56	0.96	845.0	0
69	12.21	1.75	1.28	718.0	1

Questions for Models

In [2]:
```
df_wine = pd.read_csv('../data/wine_dataset.csv',usecols=['alcohol','ash','proline','hue','class'])
df_wine.sample(4,random_state=1)
```

Out[2]:

	alcohol	ash	hue	proline	class
161	13.69	2.54	0.96	680.0	2
117	12.42	2.19	1.06	345.0	1
19	13.64	2.56	0.96	845.0	0
69	12.21	1.75	1.28	718.0	1

- Can we predict target/label "class" from the other columns? **(Classification)**
- Can we predict target/label "hue" from the other columns? **(Regression)**
- What are the important features when predicting "hue"? **(Feature Selection)**
- Can a model tell us about how the features and target/label interact? **(Interpretation)**
- Do the observations group together in feature space? **(Clustering)**

Data Vocabulary for ML

Data Vocabulary for ML

In [3]:
```
df_wine.sample(5,random_state=1)
```

Out[3]:

	alcohol	ash	hue	proline	class
161	13.69	2.54	0.96	680.0	2
117	12.42	2.19	1.06	345.0	1
19	13.64	2.56	0.96	845.0	0
69	12.21	1.75	1.28	718.0	1
53	13.77	2.68	1.13	1375.0	0

Data Vocabulary for ML

In [3]:
```
df_wine.sample(5,random_state=1)
```

Out[3]:

	alcohol	ash	hue	proline	class
161	13.69	2.54	0.96	680.0	2
117	12.42	2.19	1.06	345.0	1
19	13.64	2.56	0.96	845.0	0
69	12.21	1.75	1.28	718.0	1
53	13.77	2.68	1.13	1375.0	0

- X, features, attributes, independent/exogenous/explanatory variables
  - Ex: alcohol, trip_distance, company_industry
- y, target, label, outcome, dependent/endogenous/response variables
  - Ex: class, hue, tip_amount, stock_price
- $f(X) \rightarrow y$, Model that maps features X to target y

Variations of ML Tasks

Variations of ML Tasks

- **Supervised** vs **Unsupervised**
    - is there a target/label?

Variations of ML Tasks

- **Supervised** vs **Unsupervised**
  - is there a target/label?

- **Regression** vs **Classification**
  - is the target numeric or categorical?

Variations of ML Tasks

- **Supervised** vs **Unsupervised**
  - is there a target/label?
- **Regression** vs **Classification**
  - is the target numeric or categorical?
- **Prediction** vs **Interpretation**
  - generate predictions or understand interactions?

Variations of ML Tasks

- **Supervised** vs **Unsupervised**
  - is there a target/label?
- **Regression** vs **Classification**
  - is the target numeric or categorical?
- **Prediction** vs **Interpretation**
  - generate predictions or understand interactions?
- Model Families
  - Linear, Tree, Distance, Probability, Neural Net, Ensemble, ...

Supervised vs Unsupervised vs Reinforcement Learning

Is there a target, y?

Other Learning Paradigms

Other Learning Paradigms

- Do we have a mix of labeled and unlabeled?
  - **Semi-Supervised Learning**
  - Can we use structure of unlabeled data along with labeled?

Other Learning Paradigms

- Do we have a mix of labeled and unlabeled?
    - **Semi-Supervised Learning**
    - Can we use structure of unlabeled data along with labeled?

- Will we continue getting new data?
    - **Online Learning**
    - Is there an oracle (ground truth) we can consult?
    - Can we select which points to make predictions on?

Supervised Learning: Regression vs Classification

- **Regression** -> predict a numeric value
    - Ex: tip_amount, stock_price, wine_hue

Prediction vs Interpretation

- Do we care more about: the accuracy when generating predictions?
  - Ex: For a given taxi trip, what will the tip size likely be?
  - Ex: For a given loan, will there be a default?

- Do we care more about: understanding how X relates to y?
  - Ex: What happens to tip size as taxi trip length increases?
  - Ex: What is the relationship between debt and loan default?

Model Families for Supervised Learning

- Linear
  - Simple/Multiple Linear Regression
  - Logistic Regression (for Classification)
  - Support Vector Machines
  - Perceptron
- Tree Based
  - Decision Tree
- Distance Based
  - K-Nearest Neighbor

Model Families for Supervised Learning Continued

- Probability
    - Naive Bayes
    - Bayes Net
- Ensemble
    - Random Forest
    - Gradient Boosted Trees
    - Stacking
- Network
    - Multi-layer Perceptron
    - Deep Neural-Networks
    - Convolutional Neural Nets
    - Recurrant Neural Nets

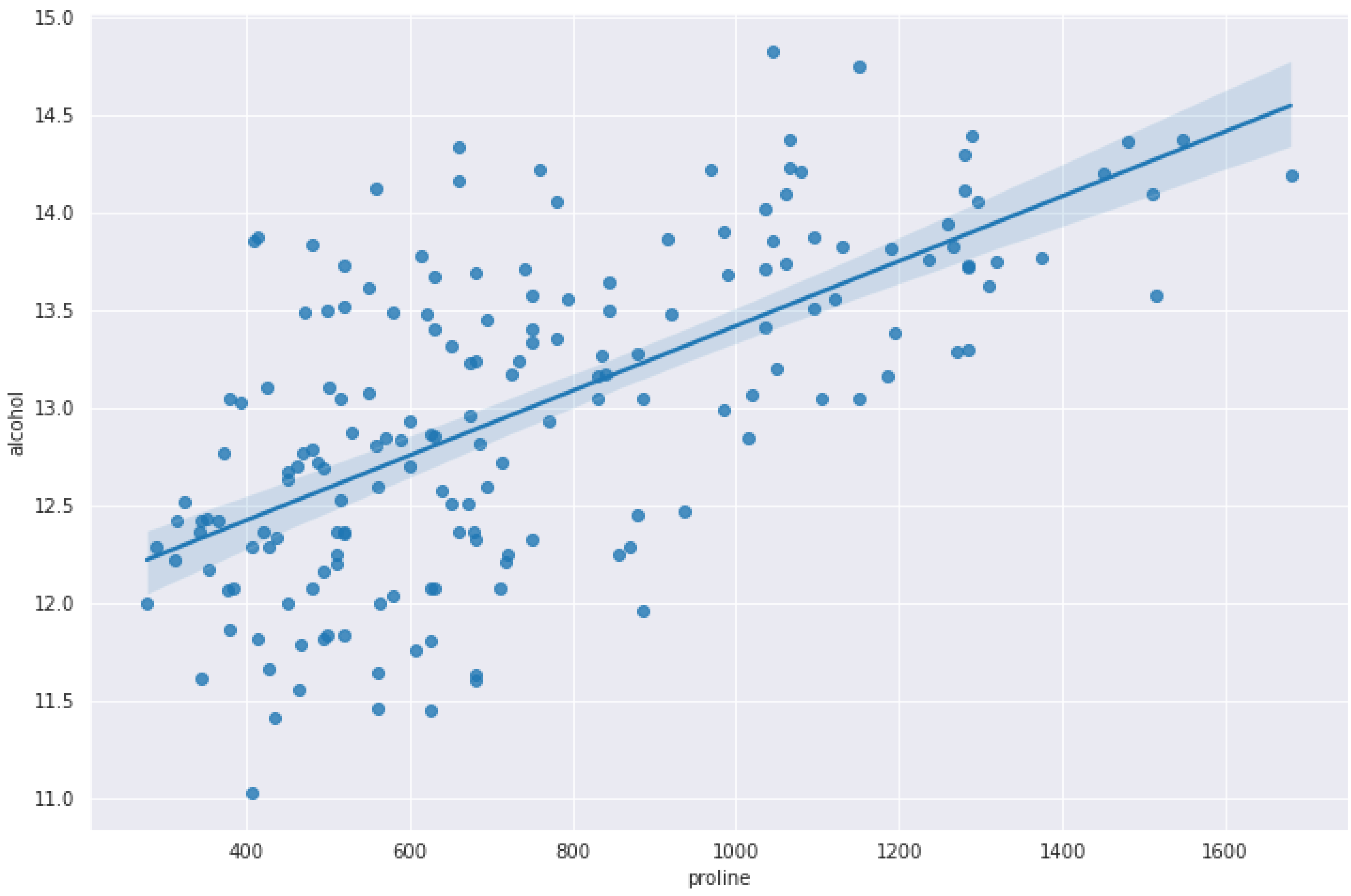
Example: Regression with a Linear Model

What is the relationship between 'proline' (an amino-acid) and 'alcohol' in wine?

Example: Regression with a Linear Model

What is the relationship between 'proline' (an amino-acid) and 'alcohol' in wine?

In [4]:
```
fig,ax = plt.subplots(1,1,figsize=(12,8))
sns.regplot(x='proline', y='alcohol', data=df_wine);
```



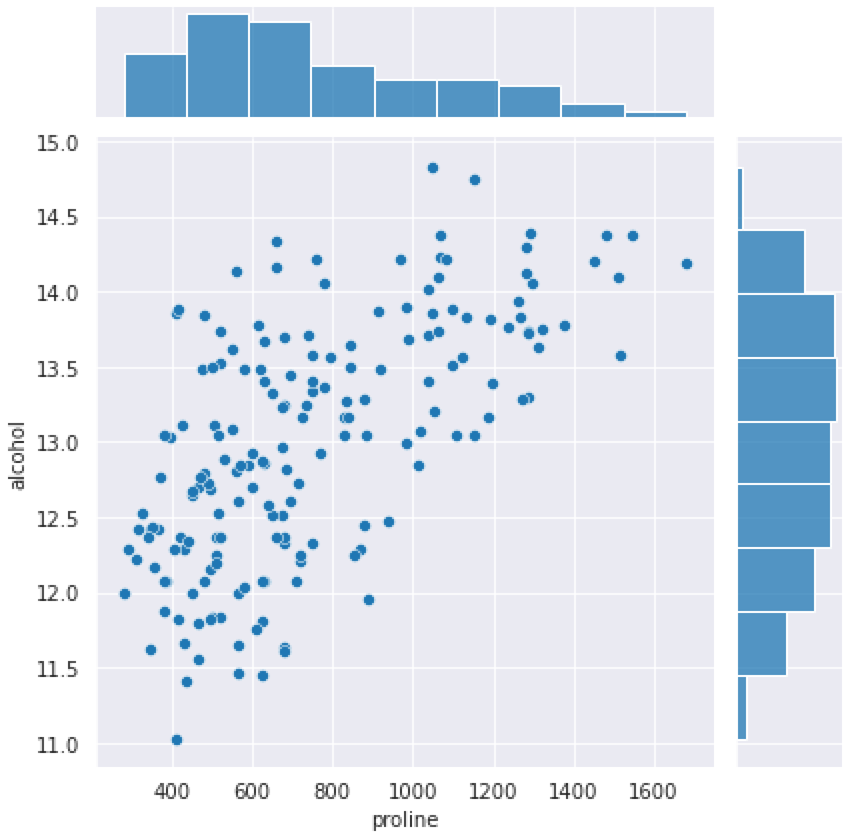
Aside: Correlation

Question: are total_bill and tips correlated?

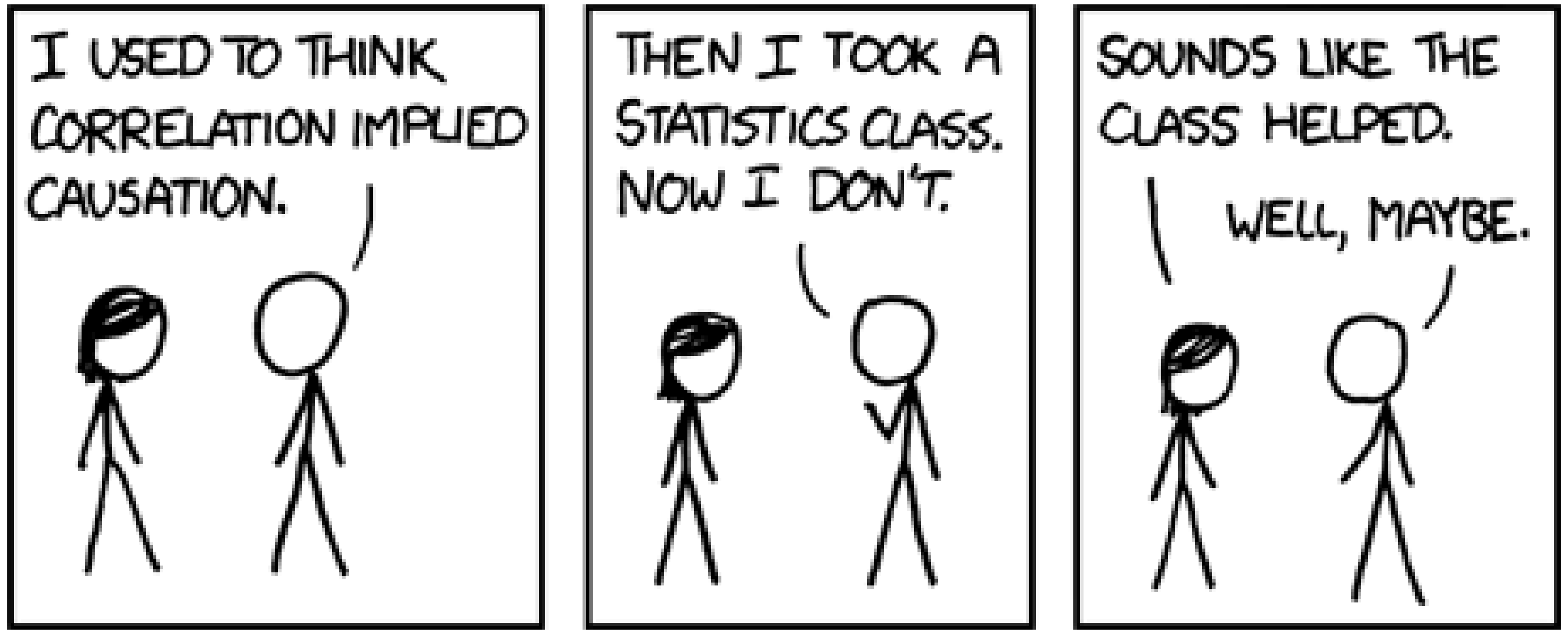
Aside: Correlation

Question: are total_bill and tips correlated?

In [5]:
```
sns.jointplot(x='proline',y='alcohol',data=df_wine);
```



Obligitory Correlation vs. Causation



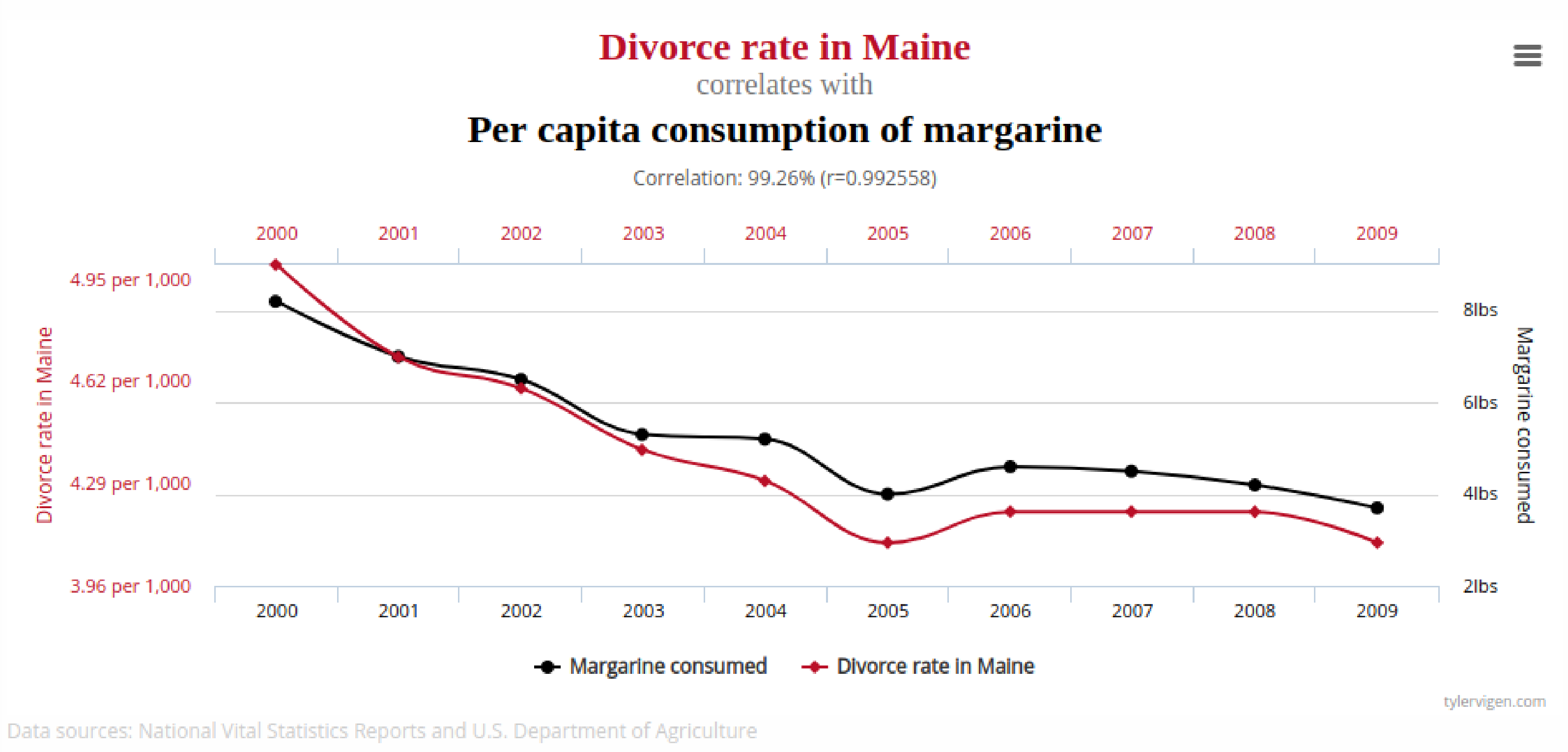
https://imgs.xkcd.com/comics/correlation.png

- Correlation does not mean causation!

- Causal Inference
  - controlled experiment
  - control for confounding variables

Spurious Correlation

- Also, look hard enough and you'll find correlation.
  - See spurious correlations for examples



Aside: Correlation

- Could calculate Pearson Correlation Coefficient
- Assumes normally distributed data! (which is not true here)
    - On the Effects of Non-Normality on the Distribution of the Sample Product-Moment Correlation Coefficient

Aside: Correlation

- Could calculate Pearson Correlation Coefficient
- Assumes normally distributed data! (which is not true here)
  - On the Effects of Non-Normality on the Distribution of the Sample Product-Moment Correlation Coefficient

In [6]:
```
from scipy.stats import pearsonr
r,p = pearsonr(df_wine.proline,df_wine.alcohol)
print(f'r: {r:.2f}, p: {p:.2f}')
```

```
r: 0.64, p: 0.00
```

Aside: Correlation

- Could calculate Pearson Correlation Coefficient
- Assumes normally distributed data! (which is not true here)
    - [On the Effects of Non-Normality on the Distribution of the Sample Product-Moment Correlation Coefficient]()

In [6]:
```
from scipy.stats import pearsonr
r,p = pearsonr(df_wine.proline,df_wine.alcohol)
print(f'r: {r:.2f}, p: {p:.2f}')
```

```
r: 0.64, p: 0.00
```

- We know that as proline goes up alcohol goes up, but by how much?

Python Modeling Libraries

Prediction - scikit-learn

Interpretation - scikit-learn and statsmodels

Additional Tools - mlxtend

Aside: MLxtend and conda-forge

- **MLxtend:** (machine learning extensions) is a Python library of useful tools for the day-to-day data science tasks.

- **Conda-Forge:** A community-led collection of recipes, build infrastructure and distributions for the conda package manager.

```
# NOTE: just an example, don't need to run, already installed
$ conda install --name eods-f21 --channel conda-forge mlxtend
```

Simple Linear Regression

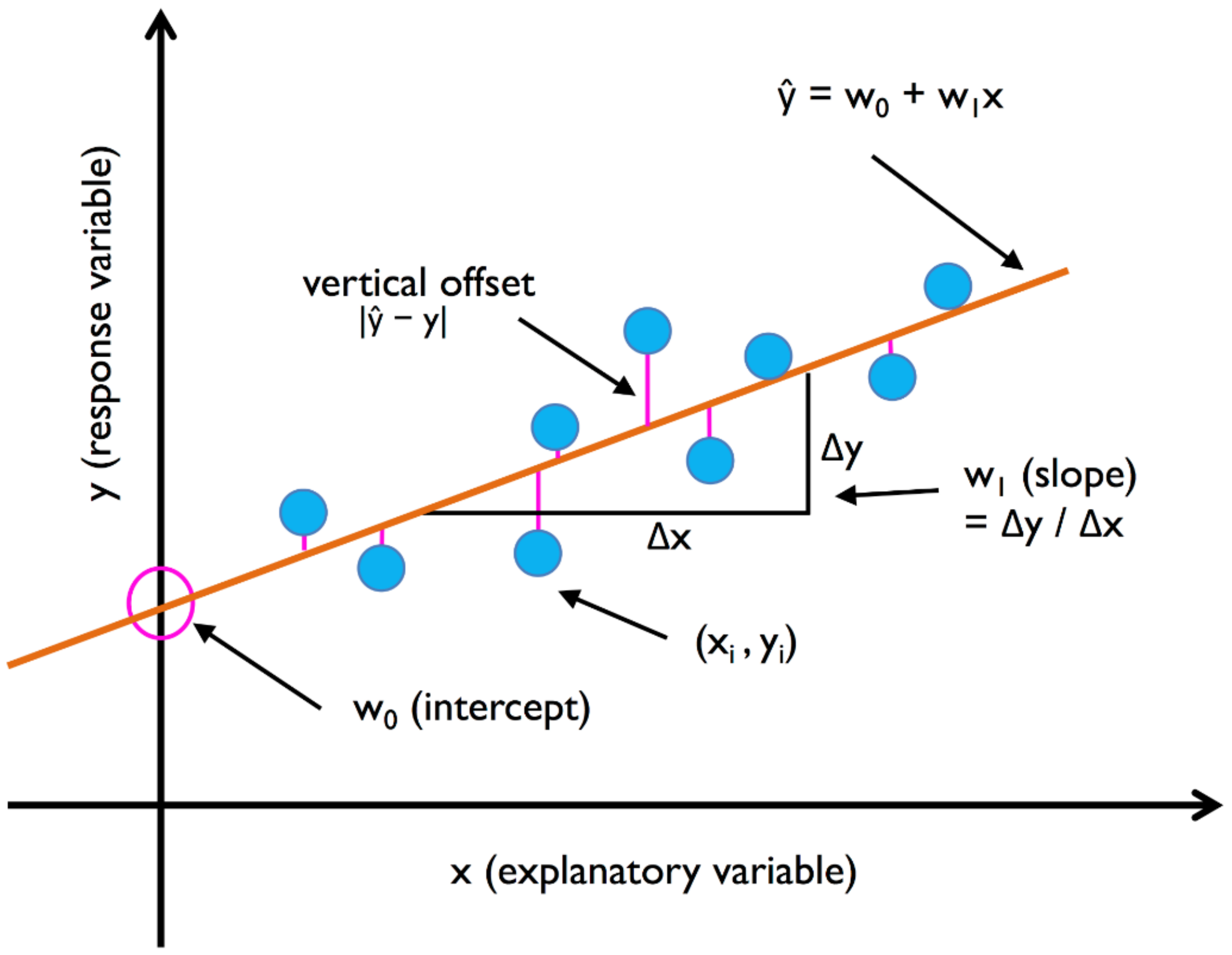
Simple Linear Regression

$$y = w_1 x + w_0 + \varepsilon_i$$

- y : dependent/endogenous response, target, label (Ex: `alcohol`)
- x_i : independent/exogenous/explanatory feature, attribute (Ex: `proline`)
- w_1 : coefficient, slope
- w_0 : bias term, intercept
- ε_i : error, hopefully small, often assumed $\mathcal{N}(0, 1)$

- Want to find values for w_1 and w_0 that best fit the data.
- Find a line as close to our observations as possible

Simple Linear Regression



from PML

Finding w_1 and w_0 with Ordinary Least Squares

- **prediction**: $\hat{y}_i = f(x_i) = w_1 x_i + w_0$
- **error**: $error(y_i, \hat{y}_i) = y_i - \hat{y}_i$
- **sum of squared errors**: $\sum_{i=1:n} \left(y_i - \hat{y}_i\right)^2$

- **least squares**: make the sum of squared errors as small as possible

- **gradient descent**: minimize error by following the gradient wrt w_1, w_0
    - can sometimes be optimized in closed form
    - often done iteratively

Aside: Gradient Descent

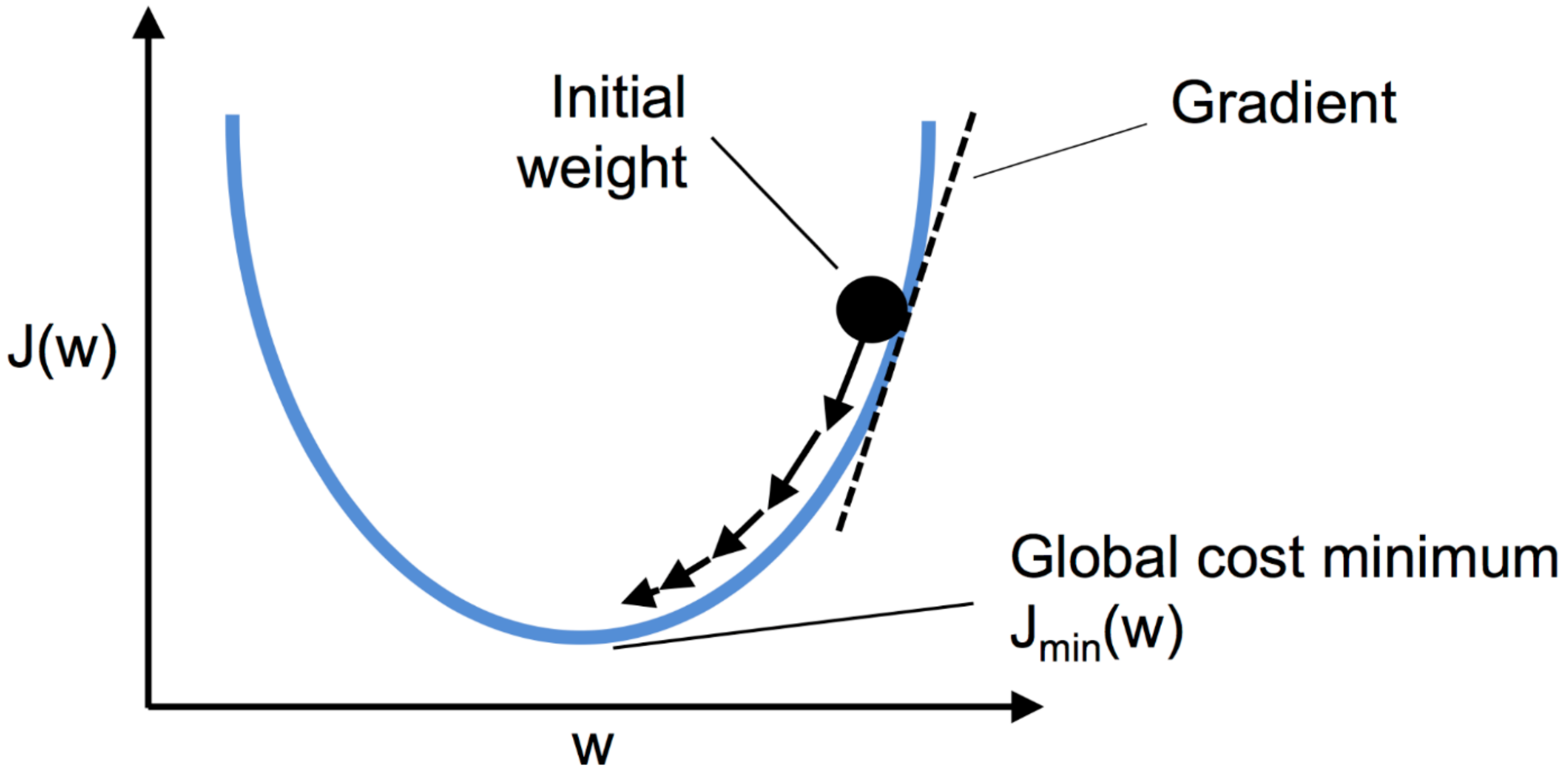
- Want to maximize or minimize something (Ex: squared error)

- **Gradient** : direction, vector of partial derivatives
  - can get complicated, often estimated

- **Gradient Descent** : take steps wrt the direction of the gradient
  - **maximize** : in the direction of the gradient
  - **minimize** : in the opposite direction of the gradient

- **Global Maximum/Minimum** : the single best solution
- **Local Maximum/Minimum** : the best solution in the neighborhood

Aside: Gradient Descent Cont.



Simple Regression Using scikit-learn

Simple Regression Using scikit-learn

In [7]:
```
# import the model from sklearn
from sklearn.linear_model import LinearRegression
```

Simple Regression Using scikit-learn

In [7]:
```
# import the model from sklearn
from sklearn.linear_model import LinearRegression
```

In [8]:
```
 # instantiate the model and set hyperparameters
lr = LinearRegression(fit_intercept=True, # by default
                      normalize=False)    # by default
```

Simple Regression Using scikit-learn

In [7]:
```
# import the model from sklearn
from sklearn.linear_model import LinearRegression
```

In [8]:
```
 # instantiate the model and set hyperparameters
lr = LinearRegression(fit_intercept=True, # by default
                      normalize=False)    # by default
```

In [9]:
```
# fit the model
lr.fit(X=df_wine.proline.values.reshape(-1, 1), y=df_wine.alcohol);
```

Simple Regression Using scikit-learn

In [7]:
```
# import the model from sklearn
from sklearn.linear_model import LinearRegression
```

In [8]:
```
 # instantiate the model and set hyperparameters
lr = LinearRegression(fit_intercept=True, # by default
                      normalize=False)    # by default
```

In [9]:
```
# fit the model
lr.fit(X=df_wine.proline.values.reshape(-1, 1), y=df_wine.alcohol);
```

In [10]:
```
# display learned coefficients (trailing underscore indicates learned values)
print(lr.coef_)
print(lr.intercept_)
```

```
[0.0016595]
11.761148483143145
```

Simple Regression Using scikit-learn

In [7]:
```
# import the model from sklearn
from sklearn.linear_model import LinearRegression
```

In [8]:
```
 # instantiate the model and set hyperparameters
lr = LinearRegression(fit_intercept=True, # by default
                      normalize=False)    # by default
```

In [9]:
```
# fit the model
lr.fit(X=df_wine.proline.values.reshape(-1, 1), y=df_wine.alcohol);
```

In [10]:
```
# display learned coefficients (trailing underscore indicates learned values)
print(lr.coef_)
print(lr.intercept_)
```

```
[0.0016595]
11.761148483143145
```

In [11]:
```
# predict given new values for proline
X = np.array([1000,2000]).reshape(-1,1)
lr.predict(X)
```

Out[11]:
```
array([13.42064866, 15.08014884])
```

Why .reshape(-1,1)?

scikit-learn models expect the input features to be 2 dimensional

Why .reshape(-1,1)?

scikit-learn models expect the input features to be 2 dimensional

In [12]:
```
df_wine.proline.values[:5]
```

Out[12]:
```
array([1065., 1050., 1185., 1480.,  735.])
```

Why .reshape(-1,1)?

scikit-learn models expect the input features to be 2 dimensional

In [12]:
```
df_wine.proline.values[:5]
```
Out[12]:
```
array([1065., 1050., 1185., 1480.,  735.])
```
In [13]:
```
df_wine.proline.values.shape
```
Out[13]:
```
(178,)
```

Why .reshape(-1,1)?

scikit-learn models expect the input features to be 2 dimensional

In [12]:
```
df_wine.proline.values[:5]
```
Out[12]:
```
array([1065., 1050., 1185., 1480.,  735.])
```
In [13]:
```
df_wine.proline.values.shape
```
Out[13]:
```
(178,)
```
In [14]:
```
df_wine.proline.values.reshape(-1,1).shape # -1 means "infer from the data"
```
Out[14]:
```
(178, 1)
```

Why .reshape(-1,1)?

scikit-learn models expect the input features to be 2 dimensional

In [12]:
```
df_wine.proline.values[:5]
```

Out[12]:
```
array([1065., 1050., 1185., 1480.,  735.])
```

In [13]:
```
df_wine.proline.values.shape
```

Out[13]:
```
(178,)
```

In [14]:
```
df_wine.proline.values.reshape(-1,1).shape # -1 means "infer from the data"
```

Out[14]:
```
(178, 1)
```

Alternatives:

Why .reshape(-1,1)?

scikit-learn models expect the input features to be 2 dimensional

In [12]:
```
df_wine.proline.values[:5]
```

Out[12]:
```
array([1065., 1050., 1185., 1480.,  735.])
```

In [13]:
```
df_wine.proline.values.shape
```

Out[13]:
```
(178,)
```

In [14]:
```
df_wine.proline.values.reshape(-1,1).shape # -1 means "infer from the data"
```

Out[14]:
```
(178, 1)
```

Alternatives:

In [15]:
```
df_wine.loc[:,['proline']].shape
```

Out[15]:
```
(178, 1)
```

Why .reshape(-1,1)?

scikit-learn models expect the input features to be 2 dimensional

In [12]:
```
df_wine.proline.values[:5]
```
Out[12]:
```
array([1065., 1050., 1185., 1480.,  735.])
```
In [13]:
```
df_wine.proline.values.shape
```
Out[13]:
```
(178,)
```
In [14]:
```
df_wine.proline.values.reshape(-1,1).shape # -1 means "infer from the data"
```
Out[14]:
```
(178, 1)
```

Alternatives:

In [15]:
```
df_wine.loc[:,['proline']].shape
```
Out[15]:
```
(178, 1)
```
In [16]:
```
df_wine[['proline']].shape
```
Out[16]:
```
(178, 1)
```

Interpreting Coefficients

Interpreting Coefficients

In [17]:
```
print(f'w_1 = {lr.coef_[0]:0.3f}, w_0 = {lr.intercept_:0.3f}')
```

```
w_1 = 0.002, w_0 = 11.761
```

Interpreting Coefficients

In [17]:
```
print(f'w_1 = {lr.coef_[0]:0.3f}, w_0 = {lr.intercept_:0.3f}')
```

```
w_1 = 0.002, w_0 = 11.761
```

In [18]:
```
print(f'alchohol = {lr.coef_[0]:0.3f}*proline + {lr.intercept_:0.3f}')
```

```
alchohol = 0.002*proline + 11.761
```

Interpreting Coefficients

In [17]:
```
print(f'w_1 = {lr.coef_[0]:0.3f}, w_0 = {lr.intercept_:0.3f}')
```
```
w_1 = 0.002, w_0 = 11.761
```

In [18]:
```
print(f'alchohol = {lr.coef_[0]:0.3f}*proline + {lr.intercept_:0.3f}')
```
```
alchohol = 0.002*proline + 11.761
```

- When proline goes up by 1, alcohol goes up by .002
- When proline is 0, alcohol is 11.761

Plotting The Model

Plotting The Model

In [19]:
```
x_predict = [df_wine.proline.min(),df_wine.proline.max()]
y_hat = lr.predict(np.array(x_predict).reshape(-1,1))

fig,ax = plt.subplots(1,1,figsize=(12,8))
ax = sns.scatterplot(x=df_wine.proline,y=df_wine.alcohol);
ax.plot(x_predict,y_hat);
```

Multiple Linear Regression

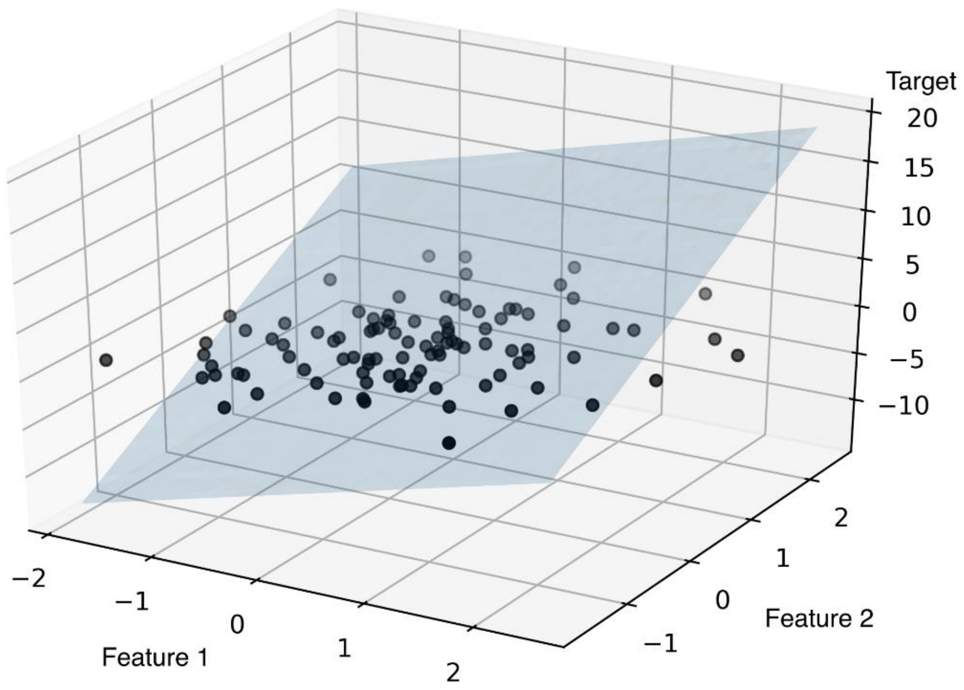
- Including multiple independent variables

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \ldots + w_m x_{im} + \varepsilon_i$$

Ex:

alcohol = w_0 + w_1*proline + w_2*hue

Objective: Find a *plane* that falls as close to our points as possible



Multiple Linear Regression in scikit-learn

Multiple Linear Regression in scikit-learn

In [20]:
```
mlr = LinearRegression()
mlr.fit(df_wine[['proline','hue']],y=df_wine.alcohol);

for (name,coef) in zip(['proline','hue'],mlr.coef_):
    print(f'{name:10s} : {coef: 0.3f}')
print(f'{"intercept":10s} : {mlr.intercept_:0.3f}')
```

```
proline    :  0.002
hue        : -0.842
intercept  : 12.459
```

Multiple Linear Regression in scikit-learn

```
In [20]: mlr = LinearRegression()
         mlr.fit(df_wine[['proline','hue']],y=df_wine.alcohol);

         for (name,coef) in zip(['proline','hue'],mlr.coef_):
             print(f'{name:10s} : {coef: 0.3f}')
         print(f'{"intercept":10s} : {mlr.intercept_:0.3f}')
```

```
proline    :  0.002
hue        : -0.842
intercept  : 12.459
```

- If we hold everything else constant, what effect does each variable have?
  - If hue is held constant, a rise of 1 proline -> rise of .002 in alcohol
  - If proline is held constant, a rise of 1 hue -> decrease of .842 in alcohol
- Can add interaction terms to allow both to move
  - Ex: hue * proline
  - more complicated to interpret

Multiple Linear Regression in statsmodels

Multiple Linear Regression in statsmodels

In [21]:
```
import statsmodels.api as sm

X = df_wine[['proline','hue']].copy()
X['const'] = 1 # or use sm.add_constant(X)
y = df_wine.alcohol
sm_mlr = sm.OLS(y,X).fit() # Note: X,y passed as parameters to object, not fit
sm_mlr.summary()
```

Out[21]:

OLS Regression Results

Dep. Variable:	alcohol	**R-squared:**	0.467
Model:	OLS	**Adj. R-squared:**	0.461
Method:	Least Squares	**F-statistic:**	76.79
Date:	Mon, 11 Oct 2021	**Prob (F-statistic):**	1.15e-24
Time:	14:48:42	**Log-Likelihood:**	-158.89
No. Observations:	178	**AIC:**	323.8
Df Residuals:	175	**BIC:**	333.3
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
proline	0.0018	0.000	12.325	0.000	0.002	0.002
hue	-0.8418	0.202	-4.175	0.000	-1.240	-0.444
const	12.4593	0.203	61.347	0.000	12.058	12.860

Omnibus:	0.751	**Durbin-Watson:**	1.734
Prob(Omnibus):	0.687	**Jarque-Bera (JB):**	0.606
Skew:	0.142	**Prob(JB):**	0.739
Kurtosis:	3.028	**Cond. No.**	4.96e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Dealing With the Intercept/Bias

- Two ways of keeping track of the bias term

Dealing With the Intercept/Bias

- Two ways of keeping track of the bias term

1. Keep it as a separate parameter:

    - $y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$
    - $y = w_0 + \sum_{i=1}^{m} w_i x_i$

Dealing With the Intercept/Bias

- Two ways of keeping track of the bias term

1. Keep it as a separate parameter:

    - $y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$
    - $y = w_0 + \sum_{i=1}^{m} w_i x_i$

2. Append a constant of $x_0 = 1$ so x and w are the same length

    - $y = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$
    - $y = \sum_{i=0}^{m} w_i x_i$

Standardizing/Normalizing Features for Interpretation

Standardizing/Normalizing Features for Interpretation

In [22]:
```
for (name,coef) in zip(['proline','hue'],mlr.coef_):
    print(f'{name:10s} : {coef: 0.3f}')
```

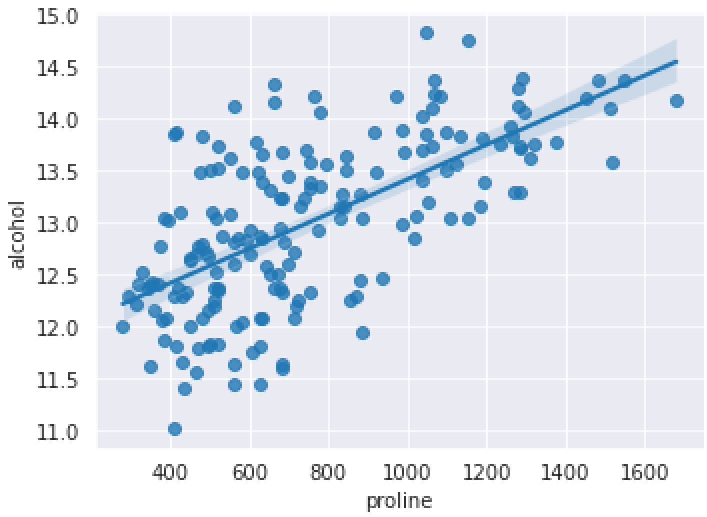
```
proline    :  0.002
hue        : -0.842
```

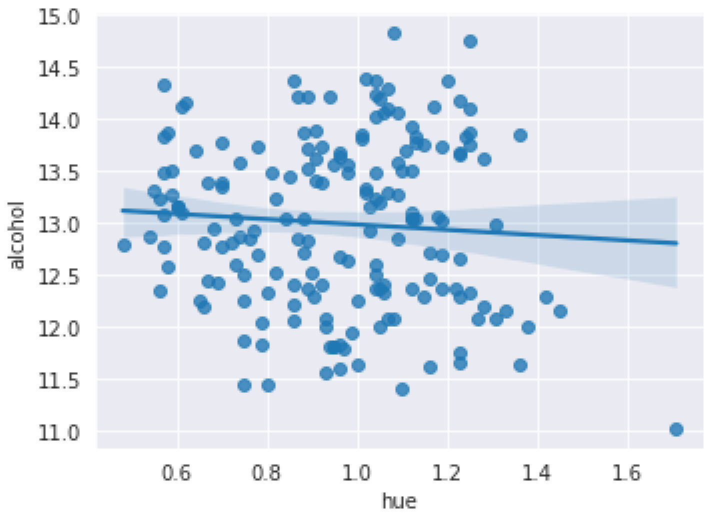
Standardizing/Normalizing Features for Interpretation

In [22]:
```
for (name,coef) in zip(['proline','hue'],mlr.coef_):
    print(f'{name:10s} : {coef: 0.3f}')
```

```
proline    :  0.002
hue        : -0.842
```

In [23]:
```
fig,ax = plt.subplots(1,2,figsize=(12,4))
sns.regplot(x='proline',y='alcohol',data=df_wine,ax=ax[0])
sns.regplot(x='hue',y='alcohol',data=df_wine,ax=ax[1]);
```



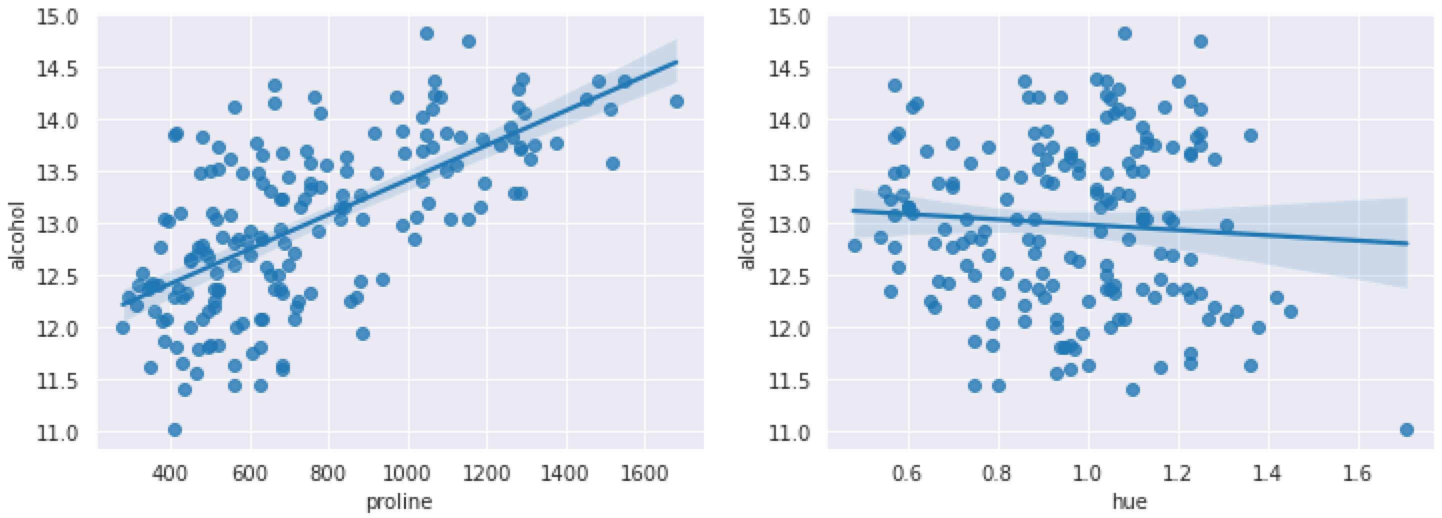


Standardizing/Normalizing Features for Interpretation

In [22]:
```
for (name,coef) in zip(['proline','hue'],mlr.coef_):
    print(f'{name:10s} : {coef: 0.3f}')
```

```
proline    :  0.002
hue        : -0.842
```

In [23]:
```
fig,ax = plt.subplots(1,2,figsize=(12,4))
sns.regplot(x='proline',y='alcohol',data=df_wine,ax=ax[0])
sns.regplot(x='hue',y='alcohol',data=df_wine,ax=ax[1]);
```



What would the coeffcents look like if the features were on the same scale?

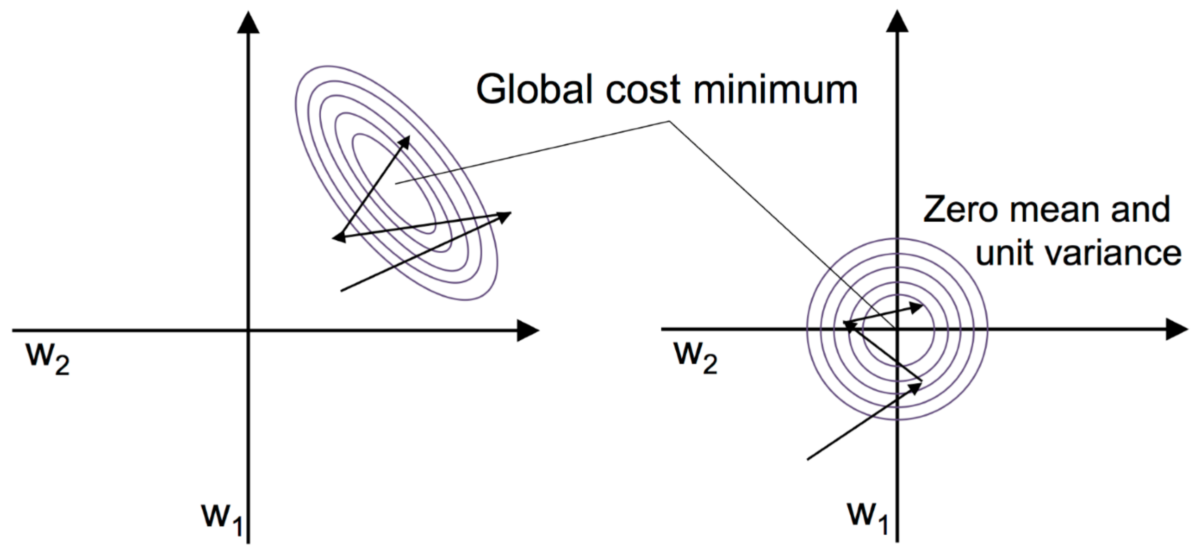
Standardizing/Normalizing Features for Gradient Descent

Standardizing/Normalizing Features for Gradient Descent

$$z = \frac{x - \bar{x}}{s}$$

Standardizing/Normalizing Features for Gradient Descent

$$z = \frac{x - \bar{x}}{s}$$



From PML

Multiple Linear Regression with Standardization/Normalization

Multiple Linear Regression with Standardization/Normalization

- `DataFrame.apply()`: apply a function across rows (axis=0) or columns (axis=1)

Multiple Linear Regression with Standardization/Normalization

- `DataFrame.apply()`: apply a function across rows (axis=0) or columns (axis=1)

```
In [24]: X_zscore = df_wine[['proline','hue']].apply(lambda x: (x-x.mean())/x.std(), axis=0)

         mlr_n = LinearRegression()
         mlr_n.fit(X_zscore, df_wine.alcohol)
         for (name,coef) in zip(X_zscore.columns,mlr_n.coef_):
             print(f'{name:10s} : {coef: 0.3f}')
```

```
proline    :  0.568
hue        : -0.192
```

Multiple Linear Regression with Standardization/Normalization

- `DataFrame.apply()`: apply a function across rows (axis=0) or columns (axis=1)

In [24]:
```
X_zscore = df_wine[['proline','hue']].apply(lambda x: (x-x.mean())/x.std(), axis=0)

mlr_n = LinearRegression()
mlr_n.fit(X_zscore, df_wine.alcohol)
for (name,coef) in zip(X_zscore.columns,mlr_n.coef_):
    print(f'{name:10s} : {coef: 0.3f}')
```

```
proline    :  0.568
hue        : -0.192
```

In [25]:
```
fig,ax = plt.subplots(1,2,figsize=(12,4))
sns.regplot(x=X_zscore.proline,y=df_wine.alcohol,ax=ax[0]);
sns.regplot(x=X_zscore.hue,y=df_wine.alcohol,ax=ax[1]);
```

Colinarity

- MLR assumes features are linearly independent
  - eg: Can't rewrite one column as a weighted sum of the others
  - Ex: at a restaurant number, of entrees ordered will likely be linearly related to table size
- Issue: Model won't know how to estimate w
  - If we add to one w_i and subtract from another, there will be no change in error
- Try to remove obvious colinearity
  - can use correlation and linear regression to detect
  - Important to consider when constructing categorical features (feature engineering)

Colinarity

- MLR assumes features are linearly independent
  - eg: Can't rewrite one column as a weighted sum of the others
  - Ex: at a restaurant number, of entrees ordered will likely be linearly related to table size
- Issue: Model won't know how to estimate w
  - If we add to one w_i and subtract from another, there will be no change in error
- Try to remove obvious colinearity
  - can use correlation and linear regression to detect
  - Important to consider when constructing categorical features (feature engineering)

In [26]:
```
df_wine.corr()
```

Out[26]:

	alcohol	ash	hue	proline	class
alcohol	1.000000	0.211545	-0.071747	0.643720	-0.328222
ash	0.211545	1.000000	-0.074667	0.223626	-0.049643
hue	-0.071747	-0.074667	1.000000	0.236183	-0.617369
proline	0.643720	0.223626	0.236183	1.000000	-0.633717
class	-0.328222	-0.049643	-0.617369	-0.633717	1.000000

Aside: Interpretation Vs. Prediction

- Interpretation: Explain how observed features relate to observed target
- Prediction: Given new features, can we generate a prediction

- Often asked to do one or the other, be clear which is most important
- In prediction, may not worry about interpreting the model!

- There is increased attention on interpretability

Questions re Regression with Linear Models?

Classification

- **Regression** -> predict a numeric value
- **Classification** -> predict a discrete class, category

- **Binary classification** : two categories
    - pos/neg, cat/dog, win/lose
- **Multiclass classification** : more than two categories/classes
    - red/green/blue, flower type, integer 0-10
- **Multilabel classification** : can assign more than one label to an instance
    - paper topics, entities in image

Wine as Binary Classification

Wine as Binary Classification

In [27]:
```
df_wine['class'].value_counts() # originally multiclass
```

Out[27]:
```
1    71
0    59
2    48
Name: class, dtype: int64
```

Wine as Binary Classification

In [27]:
```
df_wine['class'].value_counts() # originally multiclass
```

Out[27]:
```
1    71
0    59
2    48
Name: class, dtype: int64
```

In [28]:
```
# only keep classes 0 and 1
df_wine_2class = df_wine[df_wine['class'] < 2]

# rename 'class' as 'target', since class is a reserved python word
df_wine_2class = df_wine_2class.rename({'class':'target'},axis=1)

df_wine_2class.target.value_counts()
```

Out[28]:
```
1    71
0    59
Name: target, dtype: int64
```

Classifying Wine with a Linear Model

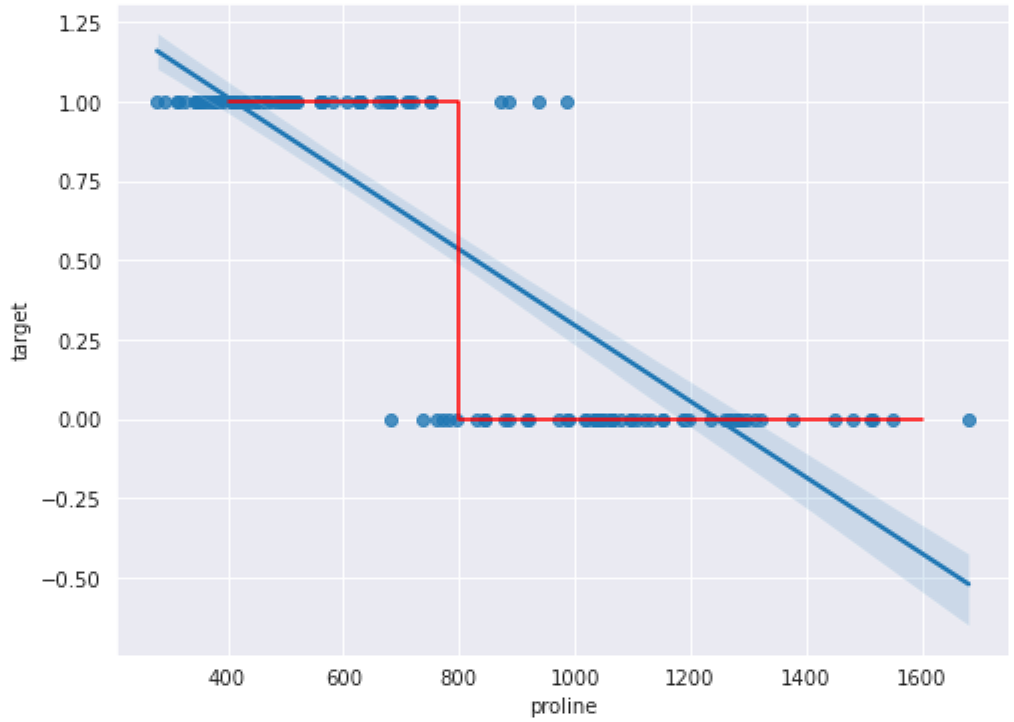
Classifying Wine with a Linear Model

- Can't use our linear regression model directly

Classifying Wine with a Linear Model

- Can't use our linear regression model directly

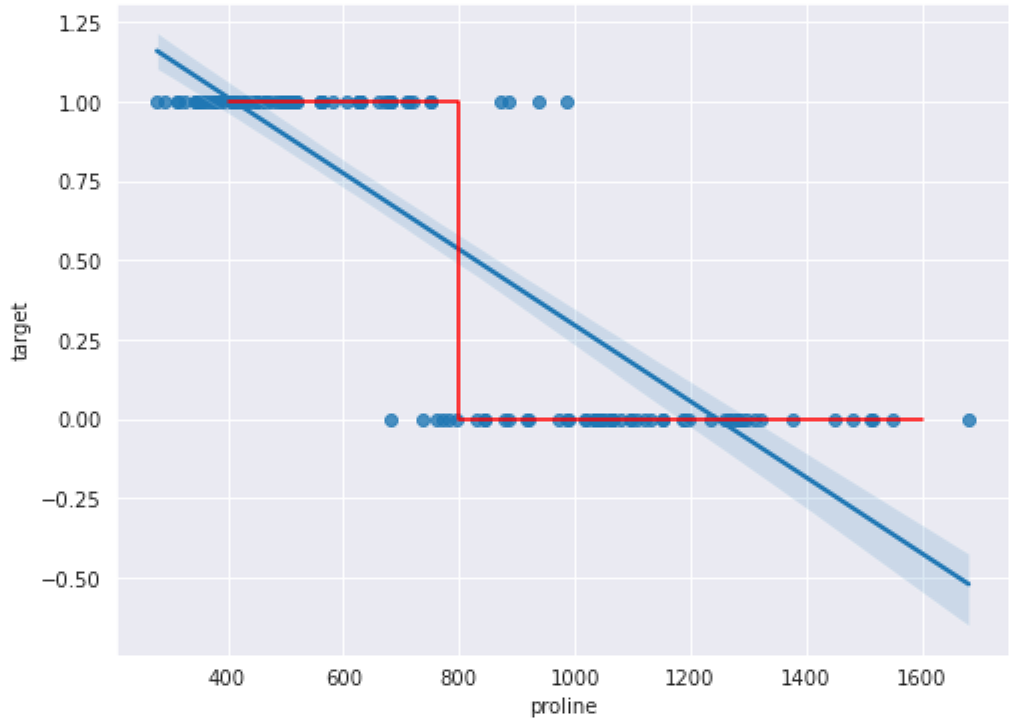
In [29]:
```
fig,ax = plt.subplots(1,1,figsize=(8,6))
sns.regplot(x=df_wine_2class.proline,y=df_wine_2class.target);
ax.plot([400,800,800,1600],[1,1,0,0],c='r');
```



Classifying Wine with a Linear Model

- Can't use our linear regression model directly

In [29]:
```
fig,ax = plt.subplots(1,1,figsize=(8,6))
sns.regplot(x=df_wine_2class.proline,y=df_wine_2class.target);
ax.plot([400,800,800,1600],[1,1,0,0],c='r');
```



- Want something with that looks like a threshold
- Would like a prediction between 0 and 1

Logistic Regression

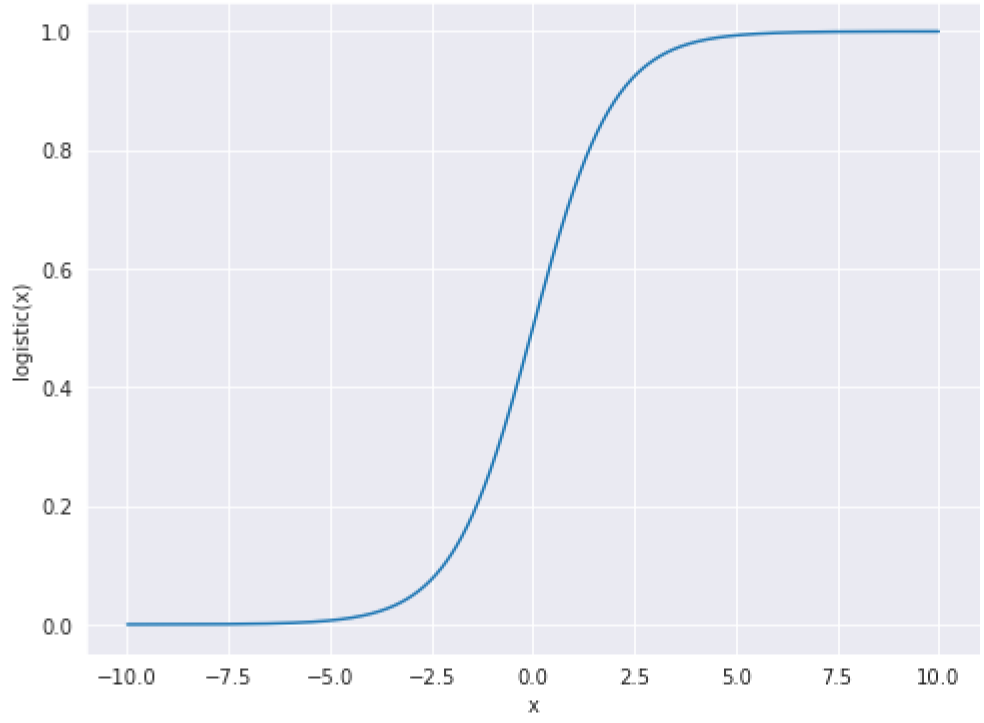
Logistic Regression

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

Logistic Regression

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

In [30]:
```
def logistic(x,w1=1,w0=0):
    return 1 / (1+np.exp(-(w0+w1*x)))

x = np.linspace(-10,10,1000) # generate 1000 numbers evenly spaced between -10 and 10
fig,ax = plt.subplots(1,1,figsize=(8,6))
ax.plot(x,logistic(x));
ax.set_xlabel('x');ax.set_ylabel('logistic(x)');
```



Logistic Regression with sklearn

Logistic Regression with sklearn

- Our problem becomes: $P(y_i = 1|x_i) = logistic(w_0 + w_1 x_i) + \varepsilon_i$

Logistic Regression with sklearn

- Our problem becomes: $P(y_i = 1|x_i) = logistic(w_0 + w_1 x_i) + \varepsilon_i$

In [31]:
```
from sklearn.linear_model import LogisticRegression

X = df_wine_2class.proline.values.reshape(-1,1)
y = df_wine_2class.target

logr = LogisticRegression(fit_intercept=True).fit(X,y)
print(f'w_0 = {logr.intercept_[0]:0.2f}')
print(f'w_1 = {logr.coef_[0][0]:0.2f}')
```

```
w_0 = 11.97
w_1 = -0.01
```

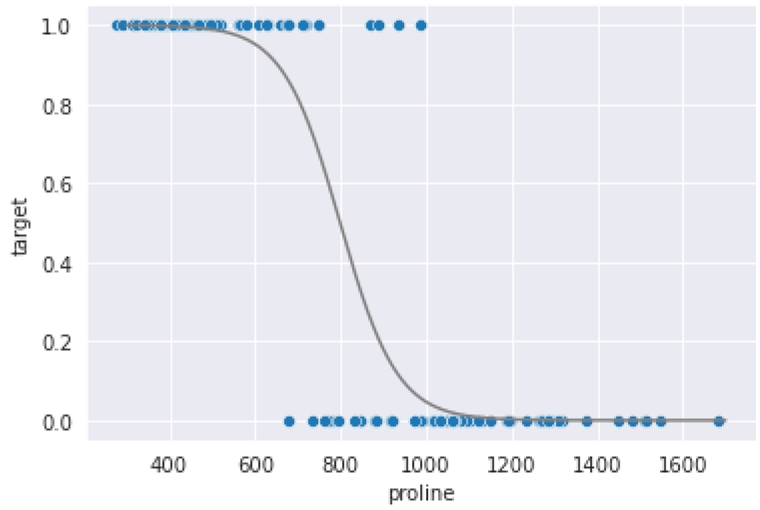
Logistic Regression with sklearn

- Our problem becomes: $P(y_i = 1|x_i) = logistic(w_0 + w_1 x_i) + \varepsilon_i$

In [31]:
```
from sklearn.linear_model import LogisticRegression

X = df_wine_2class.proline.values.reshape(-1,1)
y = df_wine_2class.target

logr = LogisticRegression(fit_intercept=True).fit(X,y)
print(f'w_0 = {logr.intercept_[0]:0.2f}')
print(f'w_1 = {logr.coef_[0][0]:0.2f}')
```

```
w_0 = 11.97
w_1 = -0.01
```

In [32]:
```
fig,ax = plt.subplots(1,1,figsize=(6,4))
x = np.linspace(300,1700,1000)
logistic_x = logistic(x,logr.coef_[0],logr.intercept_)
ax.plot(x,logistic_x,c='gray');
sns.scatterplot(x=df_wine_2class.proline,y=df_wine_2class.target, ax=ax);
```



Adding the Threshold

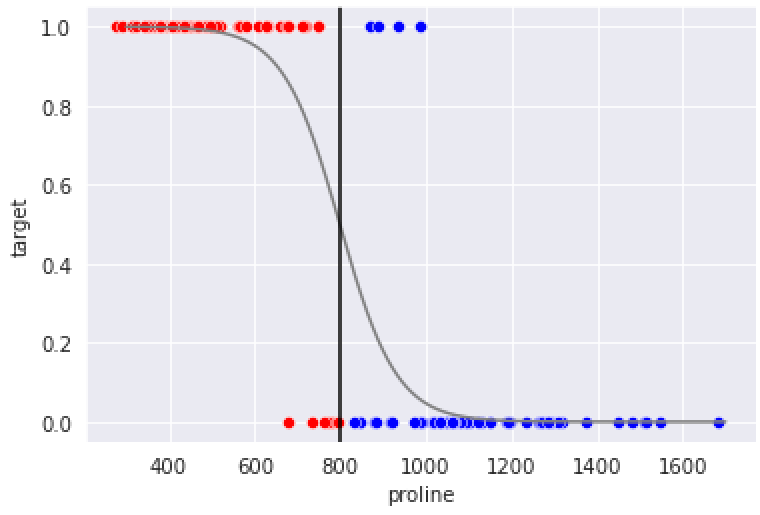
Adding the Threshold

- Can treat the output of the logistic function as $P(y = 1|x)$
- Threshold at .5 (50%) to get class prediction

Adding the Threshold

- Can treat the output of the logistic function as $P(y = 1|x)$
- Threshold at .5 (50%) to get class prediction

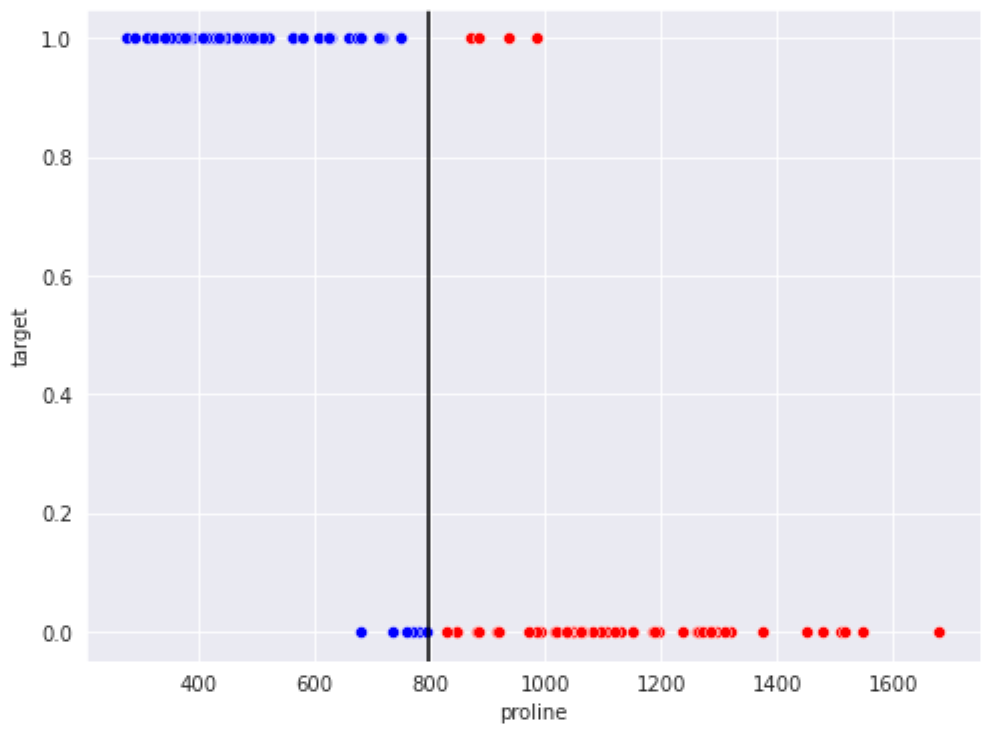
In [33]:
```
threshold = x[np.argmin(np.abs(logistic_x - .5))]

predicted_0 = df_wine_2class[df_wine_2class.proline <= threshold]
predicted_1 = df_wine_2class[df_wine_2class.proline > threshold]

fig,ax = plt.subplots(1,1,figsize=(6,4))
sns.scatterplot(x='proline',y='target', data=predicted_0, color='r',ax=ax);
sns.scatterplot(x='proline',y='target', data=predicted_1, color='b',ax=ax);
ax.plot(x,logistic_x,c='gray');
ax.axvline(threshold,c='k');
```



Getting Predictions from sklearn

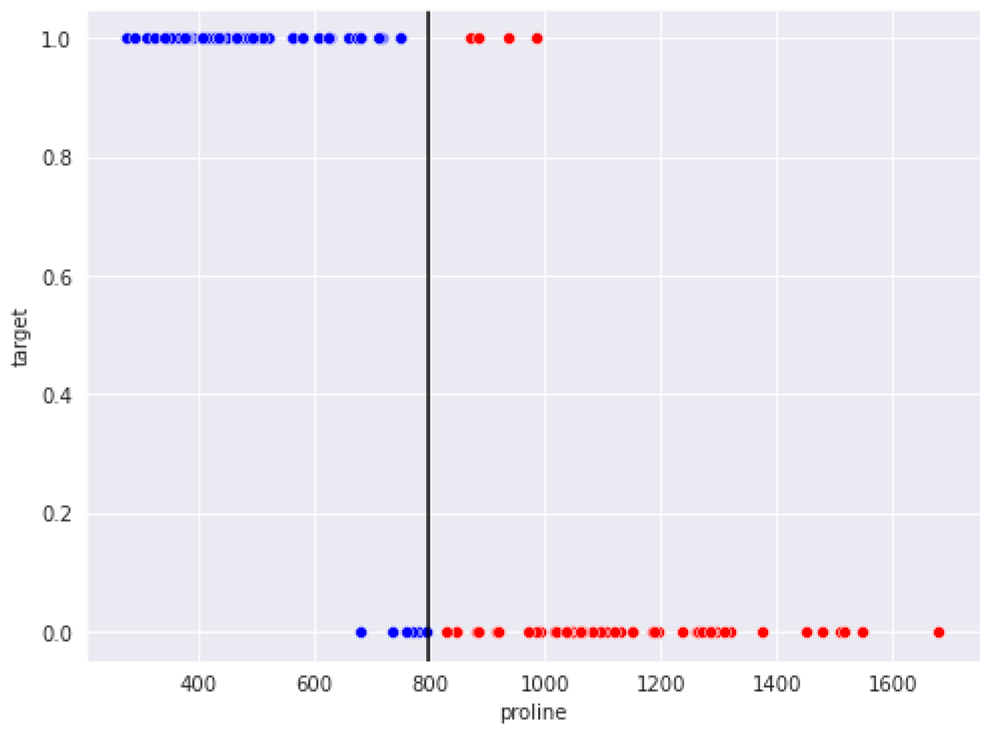
Getting Predictions from sklearn

In [34]:
```
yhat = logr.predict(X)

predicted_0 = df_wine_2class[yhat==0]
predicted_1 = df_wine_2class[yhat==1]

fig,ax = plt.subplots(1,1,figsize=(8,6))
sns.scatterplot(x='proline',y='target', data=predicted_0, color='r',ax=ax);
sns.scatterplot(x='proline',y='target', data=predicted_1, color='b',ax=ax);
ax.axvline(threshold,c='k');
```



Getting Predictions from sklearn

In [34]:
```
yhat = logr.predict(X)

predicted_0 = df_wine_2class[yhat==0]
predicted_1 = df_wine_2class[yhat==1]

fig,ax = plt.subplots(1,1,figsize=(8,6))
sns.scatterplot(x='proline',y='target', data=predicted_0, color='r',ax=ax);
sns.scatterplot(x='proline',y='target', data=predicted_1, color='b',ax=ax);
ax.axvline(threshold,c='k');
```



Note we have some errors!

Getting Probabilities from sklearn

Getting Probabilities from sklearn

- said we could use output of logistic as $P(y = 1|x)$

Getting Probabilities from sklearn

- said we could use output of logistic as $P(y = 1|x)$

In [35]:
```
p_y = logr.predict_proba(X)
p_y[:5] # p(y=0|x), p(y=1|x)
```

Out[35]:
```
array([[9.81833759e-01, 1.81662409e-02],
       [9.77356984e-01, 2.26430157e-02],
       [9.96947414e-01, 3.05258552e-03],
       [9.99963234e-01, 3.67664871e-05],
       [2.77482032e-01, 7.22517968e-01]])
```

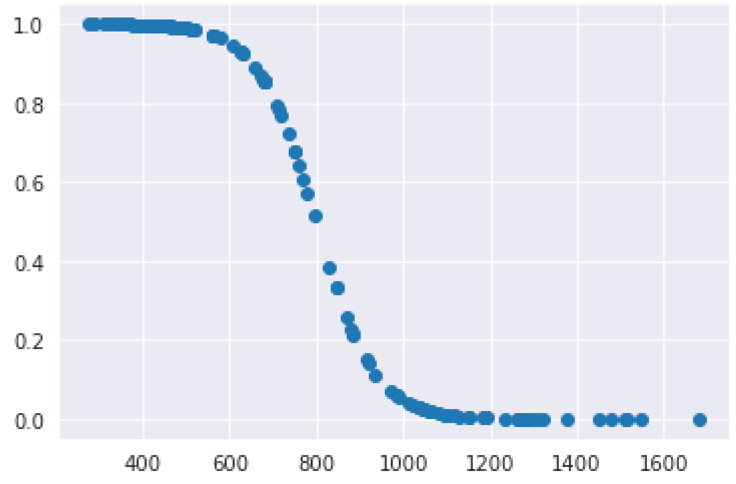
Getting Probabilities from sklearn

- said we could use output of logistic as $P(y = 1|x)$

In [35]:
```
p_y = logr.predict_proba(X)
p_y[:5] # p(y=0|x), p(y=1|x)
```

Out[35]:
```
array([[9.81833759e-01, 1.81662409e-02],
       [9.77356984e-01, 2.26430157e-02],
       [9.96947414e-01, 3.05258552e-03],
       [9.99963234e-01, 3.67664871e-05],
       [2.77482032e-01, 7.22517968e-01]])
```

In [36]:
```
plt.scatter(df_wine_2class.proline,p_y[:,1]);
```



Interpreting Logistic Regression Coefficients

- After some math

$$\log\left(\frac{y_i}{1-y_i}\right) = w_0 + w_1 x_{i1}$$

- this is the **log odds ratio** of p(y=1)/p(y=0)
- odds range from 0 to positive infinity
- odds(5) -> 5/1 -> 5 out of 6 times -> .83
- odds(.2) -> 1/5 -> 1 out of 6 times -> .16

See here for a good explanation

Logistic Regression with Multiple Features

Logistic Regression with Multiple Features

In [37]:
```
X = df_wine_2class[['proline','hue']]
X_zscore = X.apply(lambda x: (x-x.mean())/x.std())

logrm = LogisticRegression().fit(X_zscore,y)
for (name,coef) in zip(X.columns,logrm.coef_[0]):
    print(f'{name:10s} : {coef: 0.3f}')
```

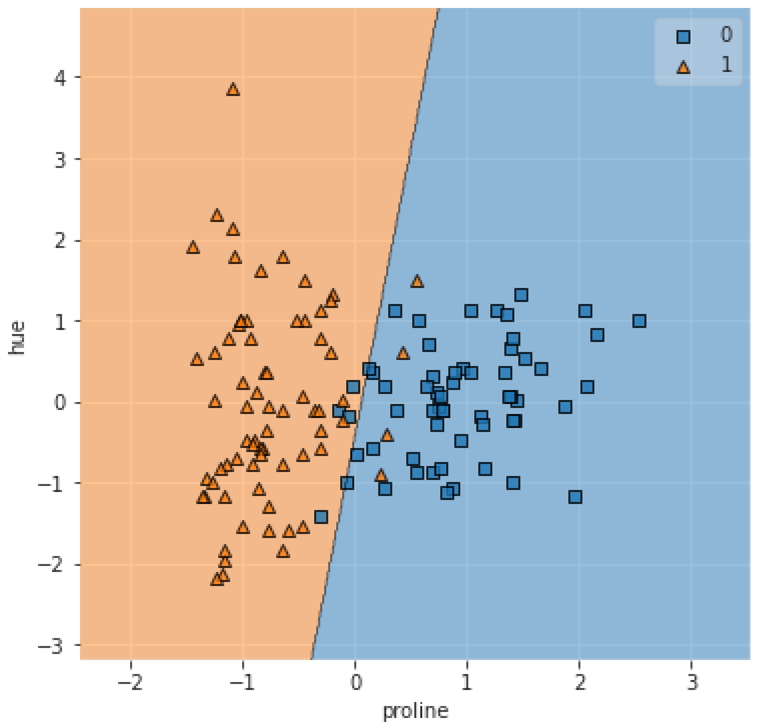
```
proline    : -3.464
hue        :  0.488
```

Logistic Regression with Multiple Features

In [37]:
```
X = df_wine_2class[['proline','hue']]
X_zscore = X.apply(lambda x: (x-x.mean())/x.std())

logrm = LogisticRegression().fit(X_zscore,y)
for (name,coef) in zip(X.columns,logrm.coef_[0]):
    print(f'{name:10s} : {coef: 0.3f}')
```

```
proline    : -3.464
hue        :  0.488
```

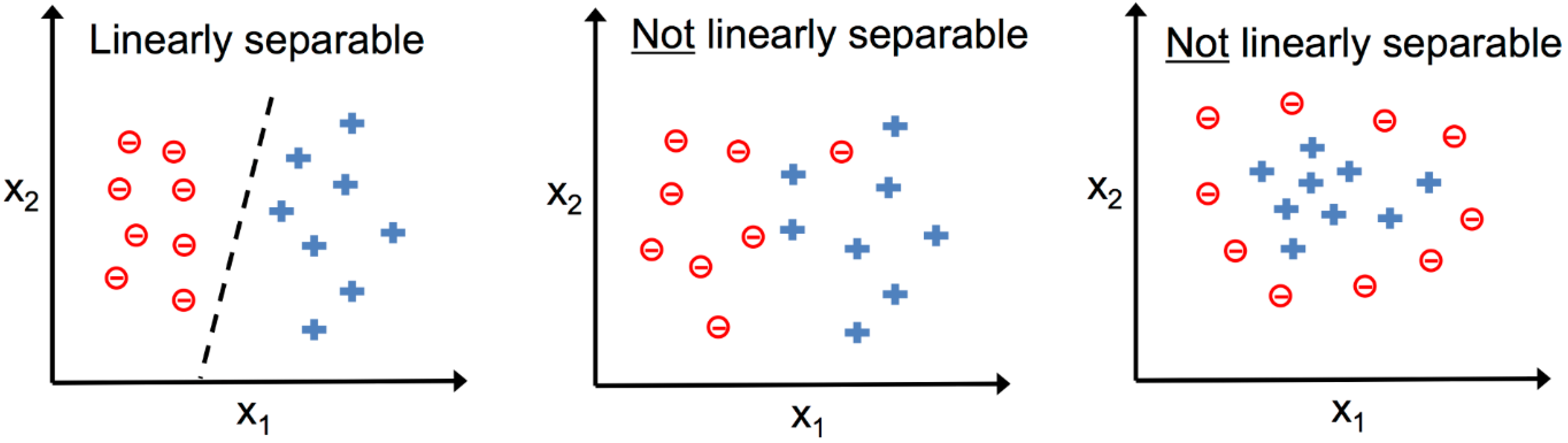
In [38]:
```
# need to have run: conda install -n eods-f20 -c conda-forge mlxtend
from mlxtend.plotting import plot_decision_regions

fig,ax = plt.subplots(1,1,figsize=(6,6))
plot_decision_regions(X_zscore.values, y.values, clf=logrm, ax=ax);
ax.set_xlabel(X.columns[0]); ax.set_ylabel(X.columns[1]);
```



Linearly Seperable Data

Linearly Seperable Data

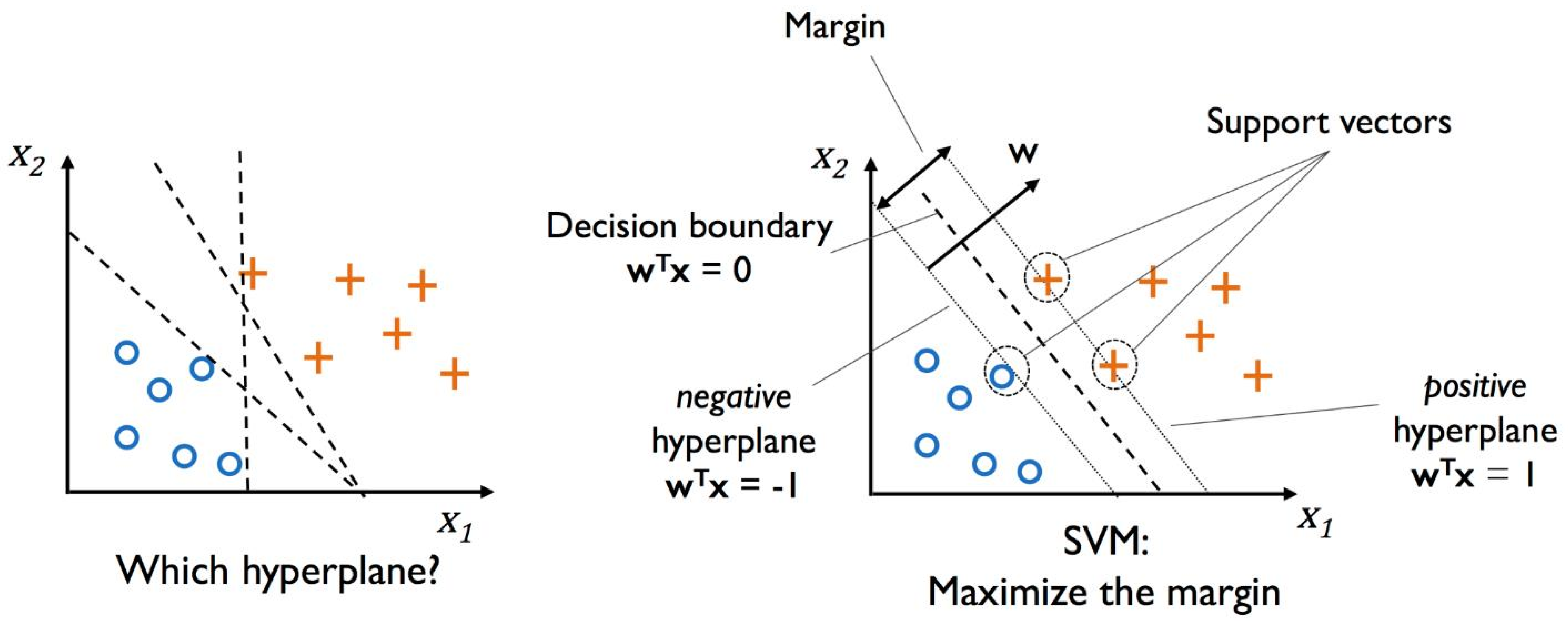
- Logistic Regression depends on data being linearly seperable



From PML

Which boundary to choose? Support Vector Machines (SVMs)

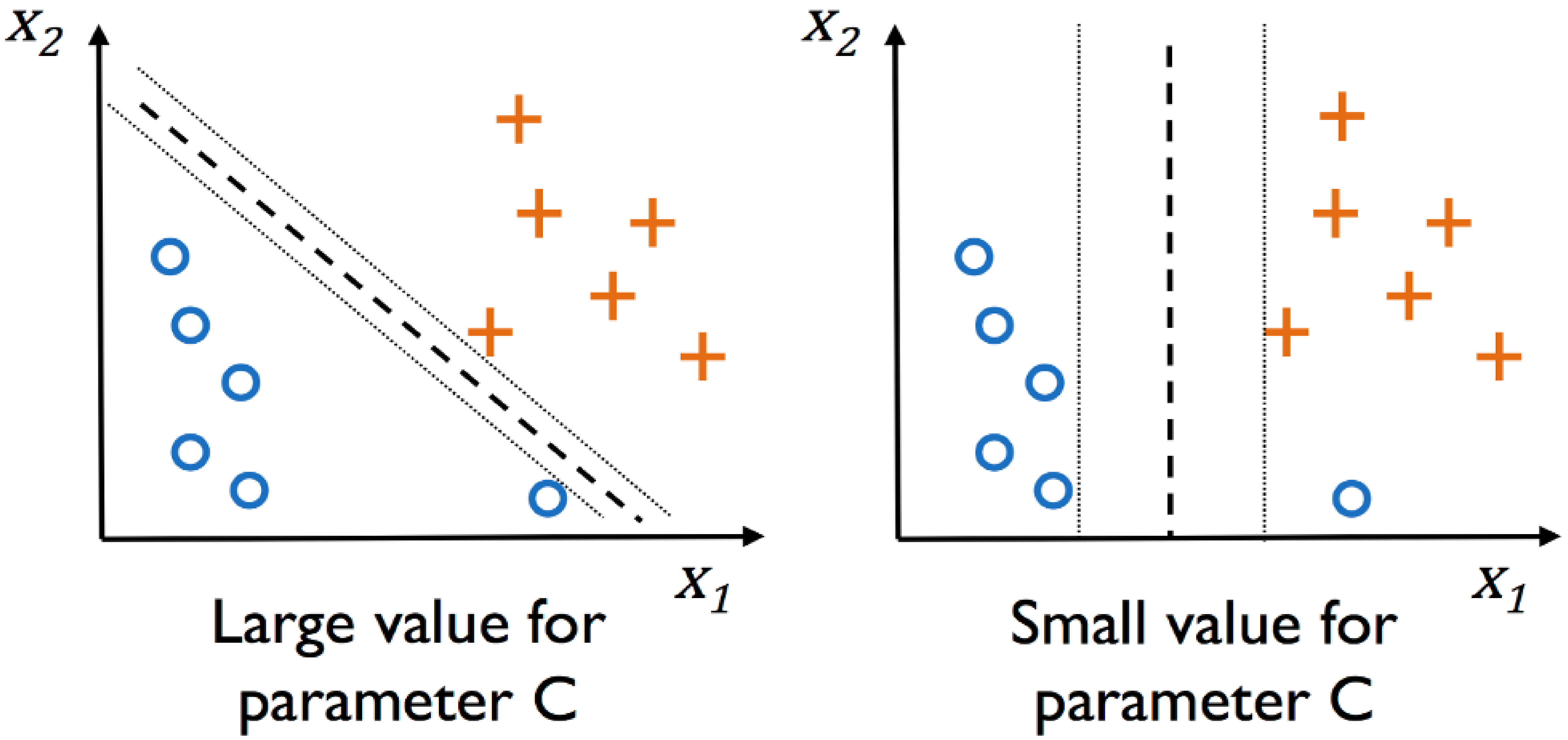
- For a linearly seperable dataset, where should we place the decision boundary?
- Support Vector Machine (SVM) tries to "maximize the margin" between classes



From PML

SVM Hyperparameter C

- **Hyperparameter**: Something we set



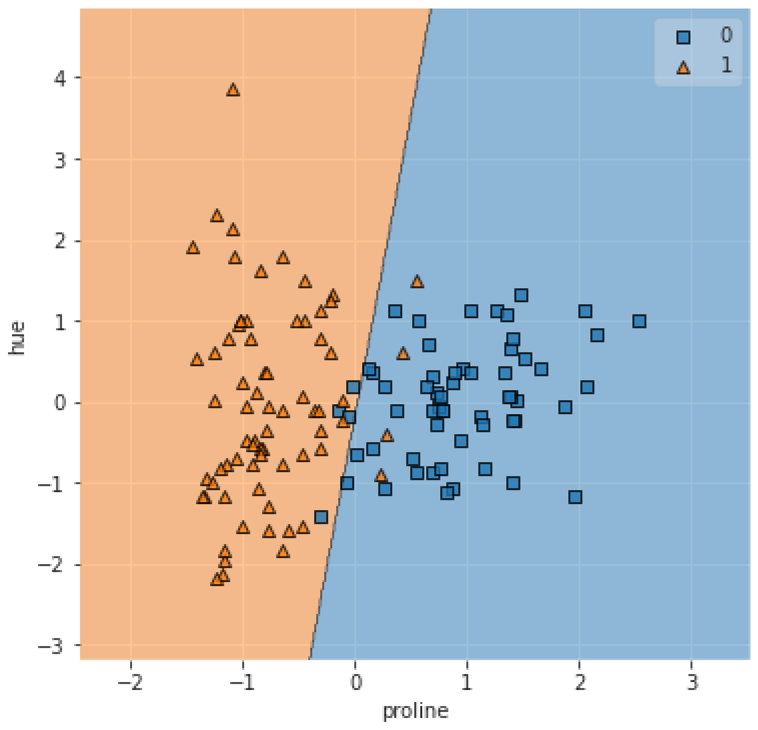
From PML

SVM with sklearn

SVM with sklearn

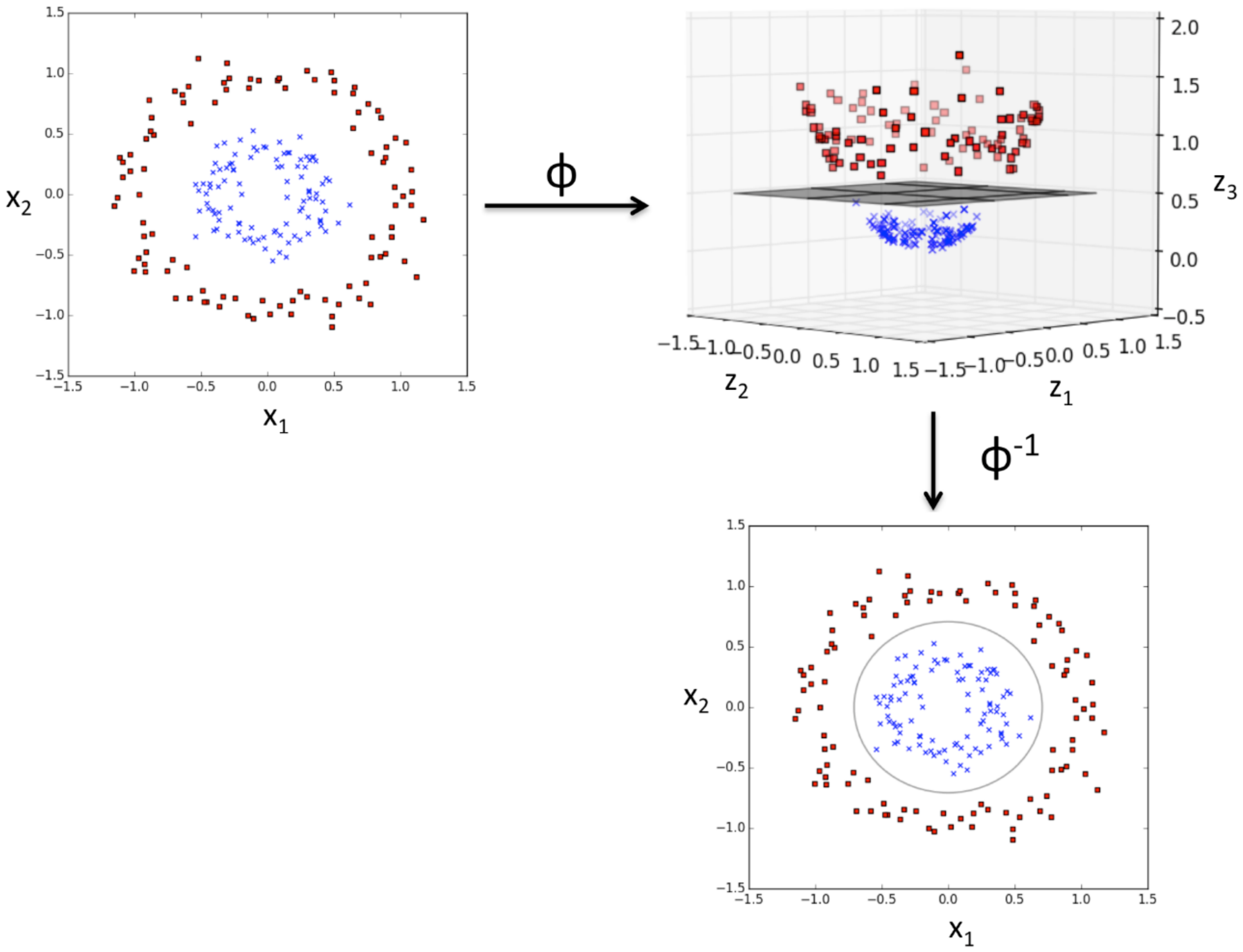
In [39]:

```
from sklearn.svm import SVC
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_zscore,y);

fig,ax = plt.subplots(1,1,figsize=(6,6))
plot_decision_regions(X_zscore.values, y.values, clf=svm_linear);
plt.xlabel(X.columns[0]); plt.ylabel(X.columns[1]);
```



Non-Linear Boundaries with SVMs Kernel Trick

- **Kernel Trick**: Map data to a higher dimensional space and find linear boundary there



From PML

SVM Kernel Trick with RBF Kernel

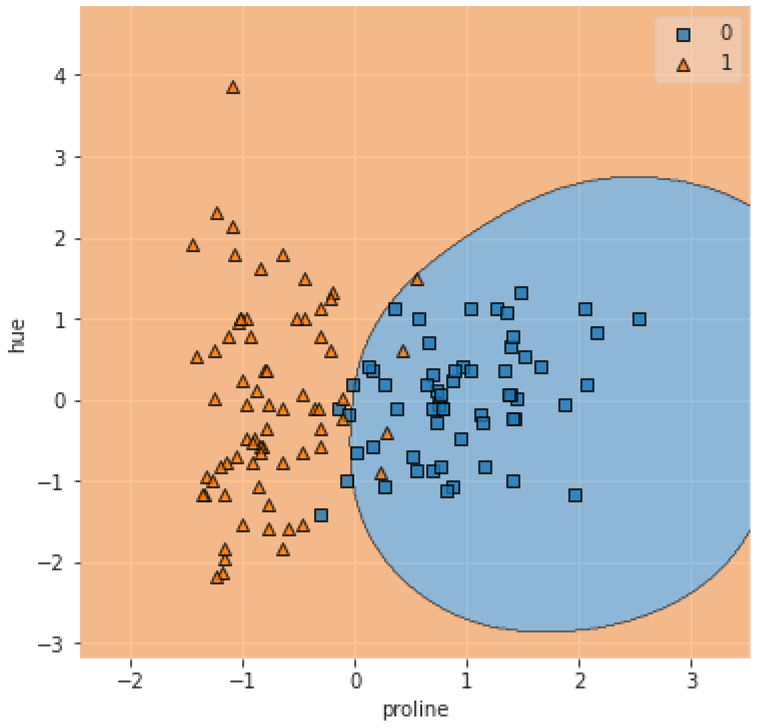
SVM Kernel Trick with RBF Kernel

- RBF (Radial-Basis Function) kernel

SVM Kernel Trick with RBF Kernel

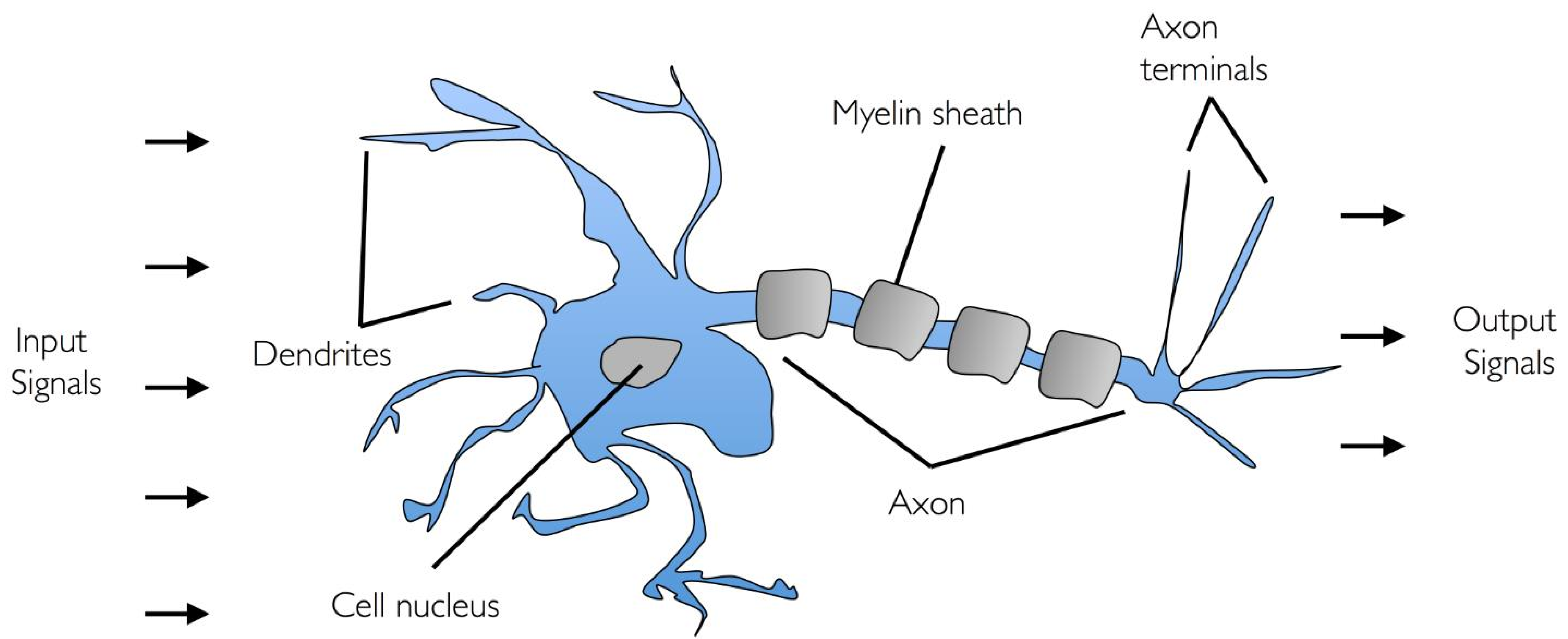
- RBF (Radial-Basis Function) kernel

In [40]:
```
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_zscore,y);

fig,ax = plt.subplots(1,1,figsize=(6,6))
plot_decision_regions(X_zscore.values, y.values, clf=svm_rbf);
plt.xlabel(X.columns[0]); plt.ylabel(X.columns[1]);
```



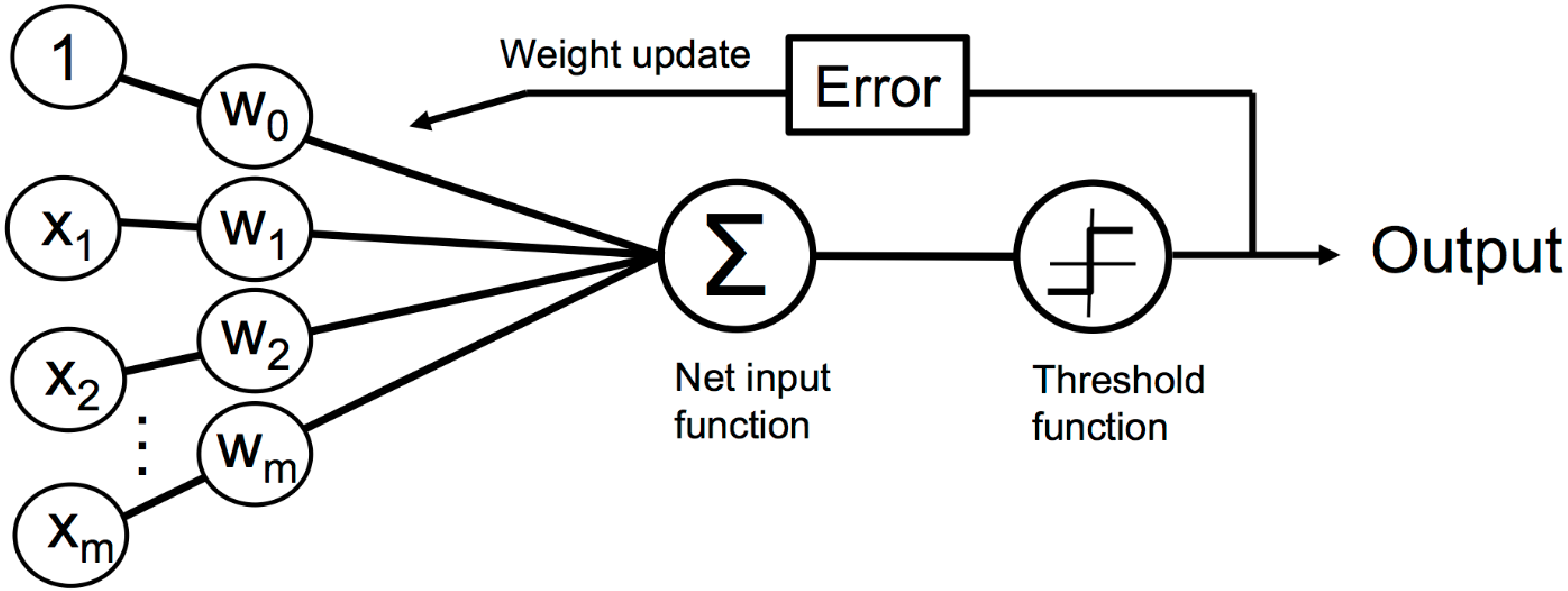
If we have time...

From Perceptron to Artificial Neural Network



From PML

Perceptron: Early Neuron Model

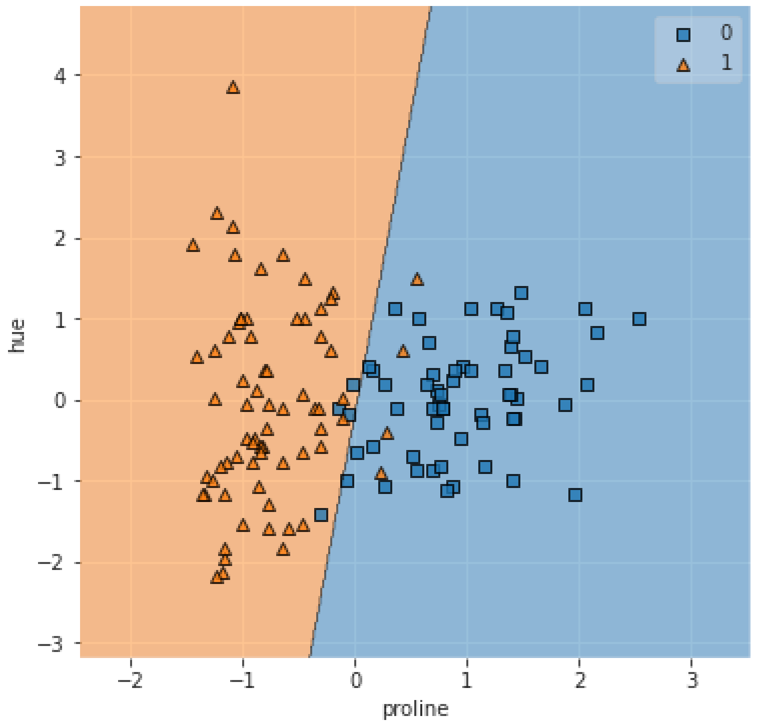


From PML

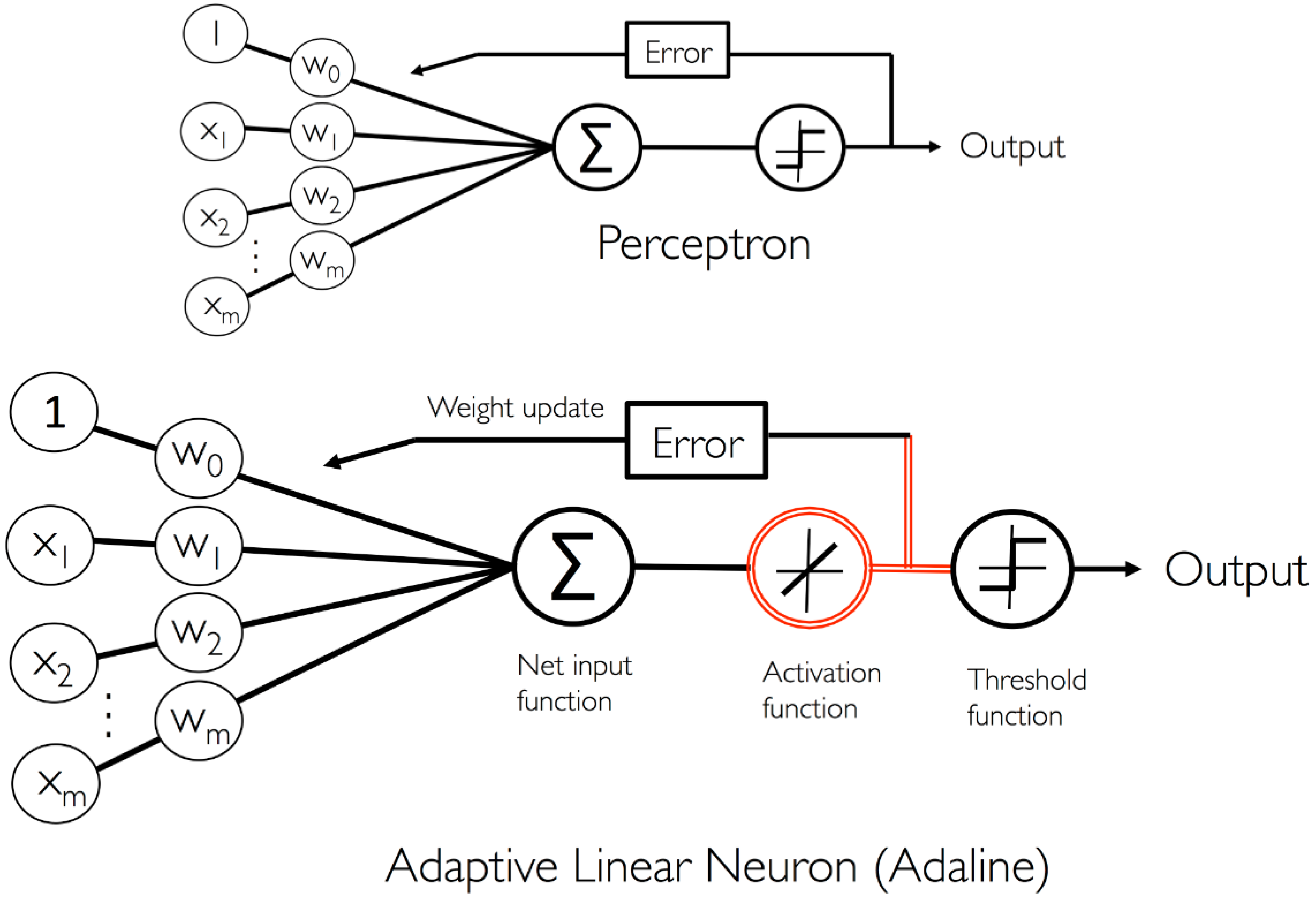
Perceptron in sklearn

Perceptron in sklearn

In [41]:
```
from sklearn.linear_model import Perceptron
perceptron = SVC(kernel='linear')
perceptron.fit(X_zscore,y);

fig,ax = plt.subplots(1,1,figsize=(6,6))
plot_decision_regions(X_zscore.values, y.values, clf=perceptron);
plt.xlabel(X.columns[0]); plt.ylabel(X.columns[1]);
```

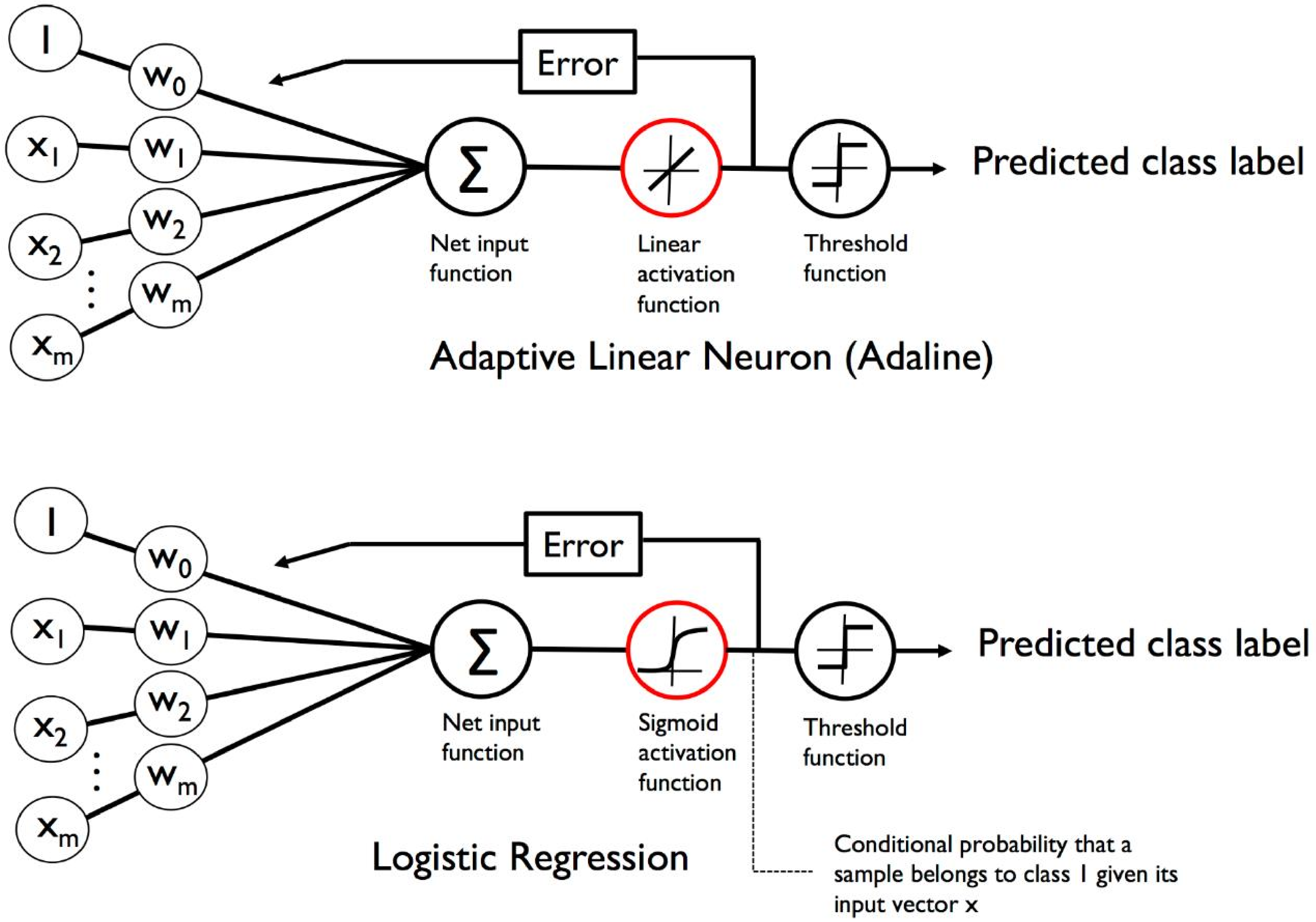


Perceptron to Adaline



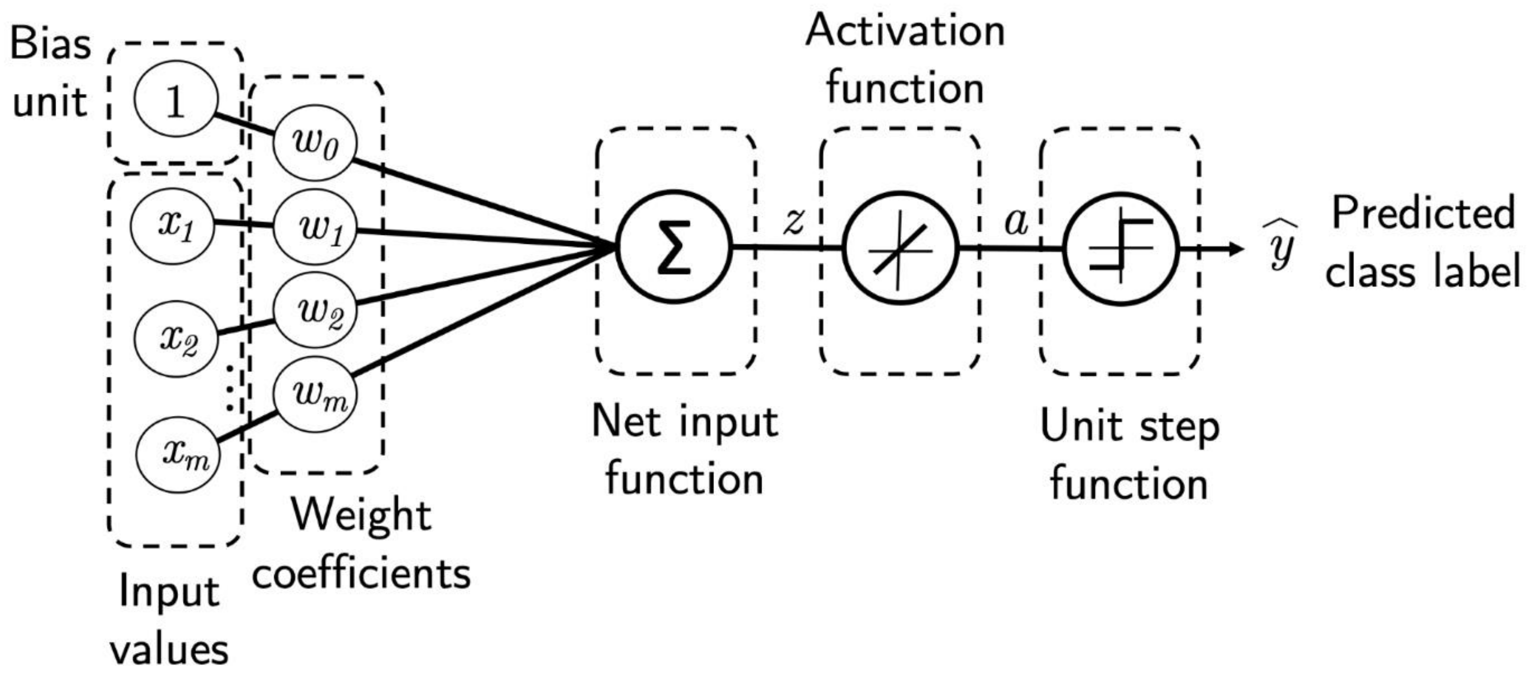
From PML

Adaline to Logistic Regression



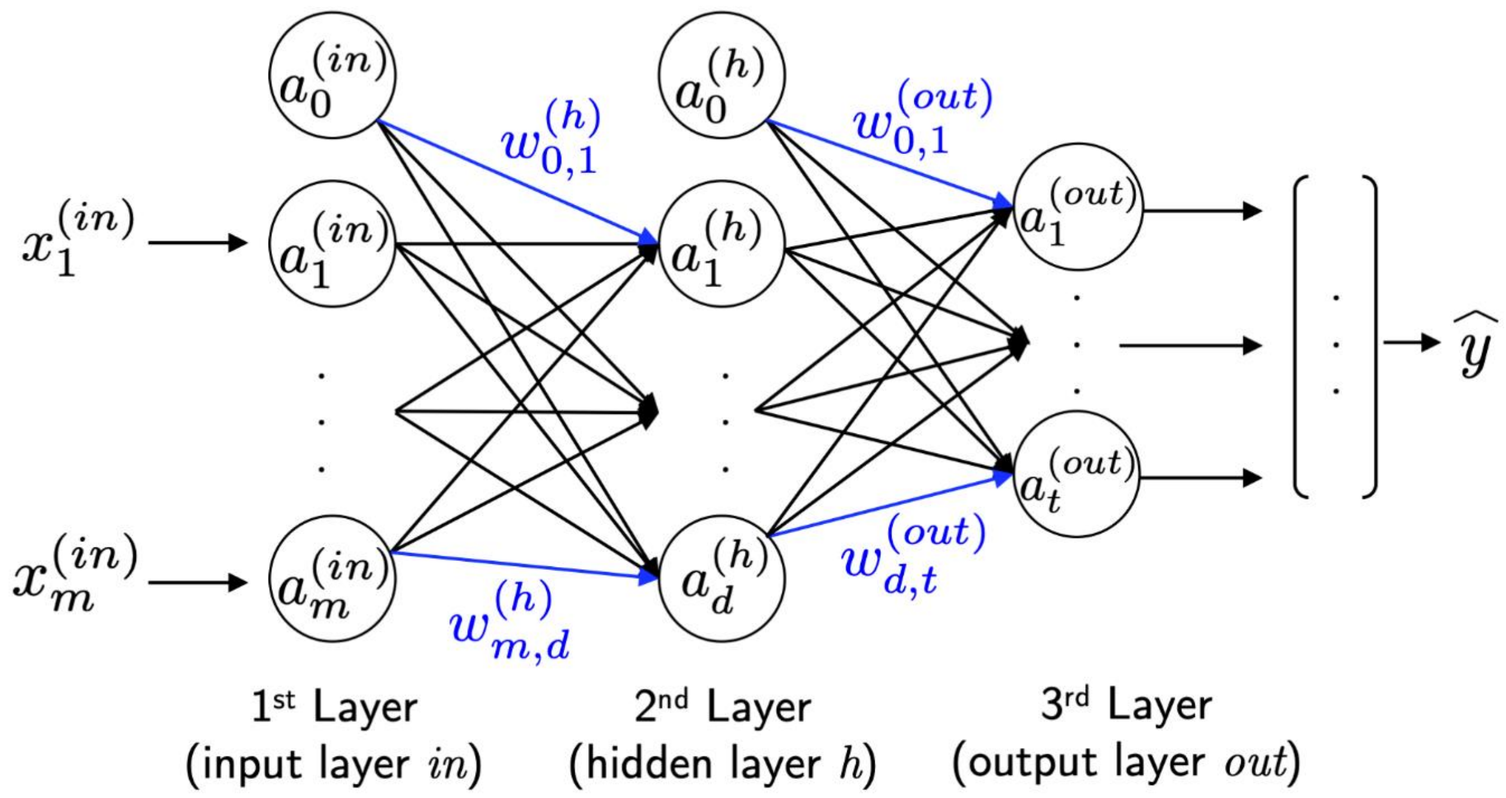
From PML

Components of Single Layer Neural Net



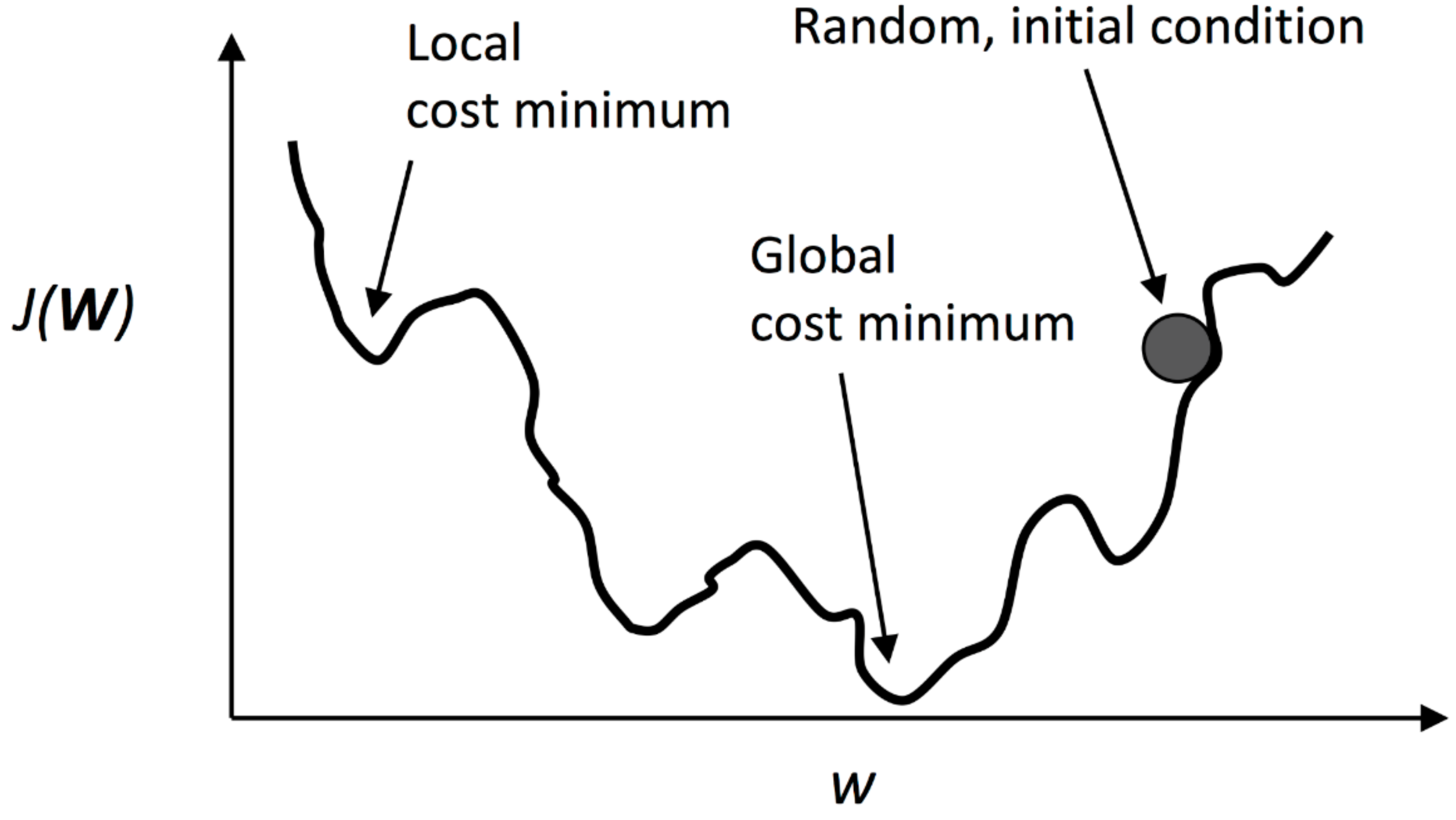
From PML

Multi-Layer Neural Network



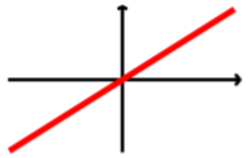
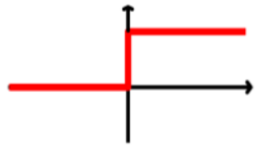
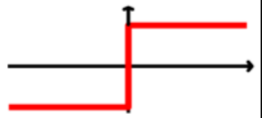
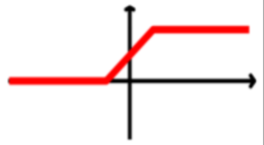
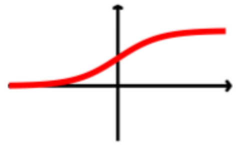
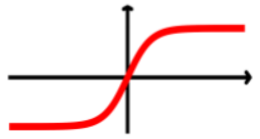
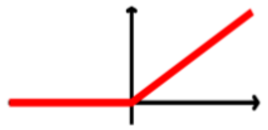
From PML

Complex Optimization Space



From PML

Activation Functions

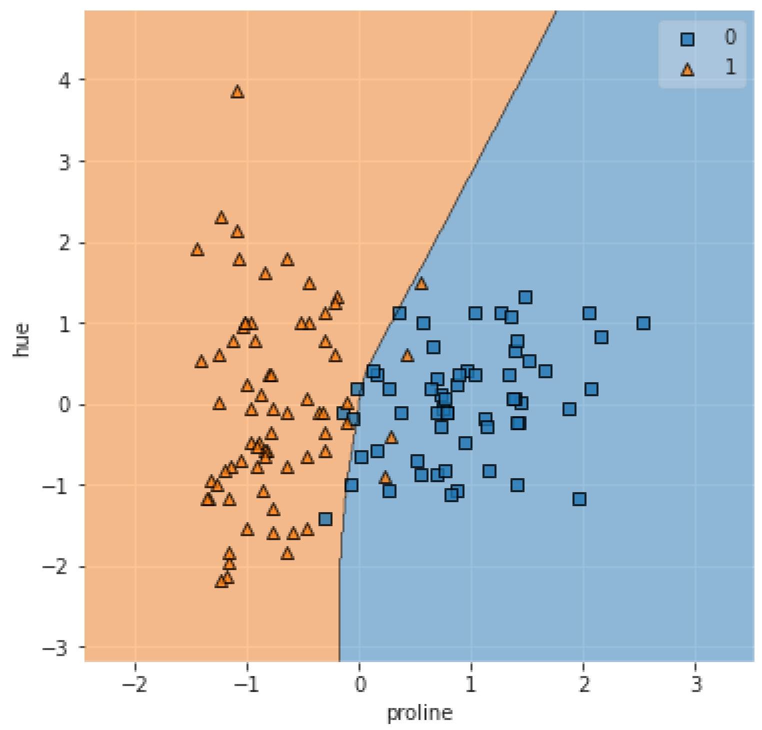
Activation function	Equation	Example	1D graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit step (Heaviside function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z)= \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise linear	$\phi(z)= \begin{cases} 0 & z \le -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \le z \le \frac{1}{2} \\ 1 & z \ge \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z)= \frac{1}{1 + e^{-z}}$	Logistic regression, multilayer NN	
Hyperbolic tangent (tanh)	$\phi(z)= \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z)= \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

From PML

Multi-Layer Perceptron with sklearn

Multi-Layer Perceptron with sklearn

In [42]:
```
from sklearn.neural_network import MLPClassifier, MLPRegressor
mlp = MLPClassifier(hidden_layer_sizes=(100,),
                    max_iter=1000)
mlp.fit(X_zscore,y);

fig,ax = plt.subplots(1,1,figsize=(6,6))
plot_decision_regions(X_zscore.values, y.values, clf=mlp);
plt.xlabel(X.columns[0]); plt.ylabel(X.columns[1]);
```



Questions re Classification with Linear Models?