Elements Of Data Science - S2022

Week 6: Intro to Machine Learning Models Continued

2/22/2022

TODOs

- Readings:
  - PDSH 05.03 **Hyperparameters and Model Validation**
  - Recommended: **https://scikit-learn.org/stable/model_selection.html**
  - Reference: **https://scikit-learn.org/stable/supervised_learning.html**
  - HOML: Chapter 2
  - HOML: Chapter 5
- Quiz 6, Due Mon Feb 28th, 11:59pm

Today

- Review Linear Models
- One Vs. Rest For Multiclass/Multilabel Classification
- Distance Based: kNN
- Tree Based: Decision Tree
- Ensembles: Bagging, Boosting, Stacking
- Model Review

Questions?

Environment Setup

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from mlxtend.plotting import plot_decision_regions

from sklearn.linear_model import LinearRegression

sns.set_style('darkgrid')
%matplotlib inline
```

In [2]:

```
def my_plot_decision_regions(X,y,model,figsize=(8,8)):
    '''Plot classifier decision regions, classification predictions and training data'''
    fig,ax = plt.subplots(1,1,figsize=figsize)
    # use mlxtend plot_decision_regions
    plot_decision_regions(X.values,y.values,model)
    ax.set_xlabel(X.columns[0]); ax.set_ylabel(X.columns[1]);

def my_plot_regression(X,y,model,label='yhat',figsize=(8,8)):
    '''Plot regression predictions and training data'''
    # generate test data and make predictions
    X_test = np.linspace(X.iloc[:,0].min(),X.iloc[:,0].max(),1000).reshape(-1,1)
    y_hat = model.predict(X_test)
    # plot
```

```
    fig,ax = plt.subplots(1,1,figsize=figsize)
    ax.scatter(X, y, s=20, edgecolor="black", c="darkorange", label="data")
    ax.plot(X_test, y_hat, color="cornflowerblue", label=label, linewidth=2)
    ax.set_xlabel(X.columns[0]); ax.set_ylabel(y.name); ax.legend();
```

Linear Models

- Simple/Multiple Linear Regression
- Logistic Regression
- SVM
- Perceptron, Multi-Layer Perceptron

Wine as Multi-Class Classification

In [3]:

```
df_wine = pd.read_csv('../data/wine_dataset.csv',usecols=['alcohol','ash','proline','hue','class'])

X = df_wine[['proline','hue']]
y_c = df_wine['class']

zscore = lambda x: (x-x.mean()) / x.std()

X_zscore = X.apply(zscore,axis=0)
alcohol_zscore = zscore(df_wine.alcohol)

y_c.value_counts().sort_index()
```

Out[3]:

```
0    59
1    71
2    48
Name: class, dtype: int64
```

One Vs. Rest (OvR) Classification For Multiclass, Multilabel

- Can use any binary classifier for Multiclass/Multilabel classification by training multiple models:
    - model 1 : class 1 vs (class 2 and class 3)
    - model 2 : class 2 vs (class 1 and class 3)
    - model 3 : class 3 vs (class 1 and class 2)

- For Multiclass
    - Predict $\hat{y}$ using the model with highest $P(y = \hat{y} \mid x)$, or distance from boundary, or ...

- For Multilabel

- Predict $\hat{y}$ for any model that predicts a value above some threshold

See **sklearn** for more info and other methods

OvR For Logistic Regression

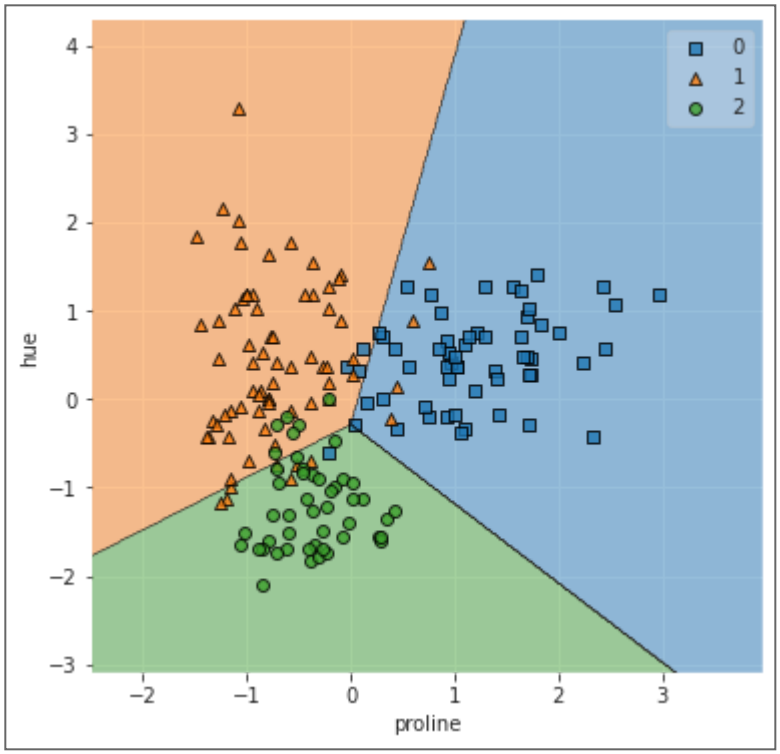
In [4]:

```
from sklearn.linear_model import LogisticRegression
logr = LogisticRegression(multi_class='ovr', # default
                          max_iter=1000)     # to avoid timeout errors
logr.fit(X_zscore,y_c)
y_hats = logr.predict(X_zscore.iloc[[15,82,166]]) # generate 3 predictions
y_prob = logr.predict_proba(X_zscore.iloc[[15,82,166]])

for y_hat,p in zip(y_hats,y_prob):
    print(y_hat,p.round(3))
```

```
0 [0.967 0.031 0.001]
1 [0.146 0.853 0.001]
2 [0.176 0.344 0.48 ]
```

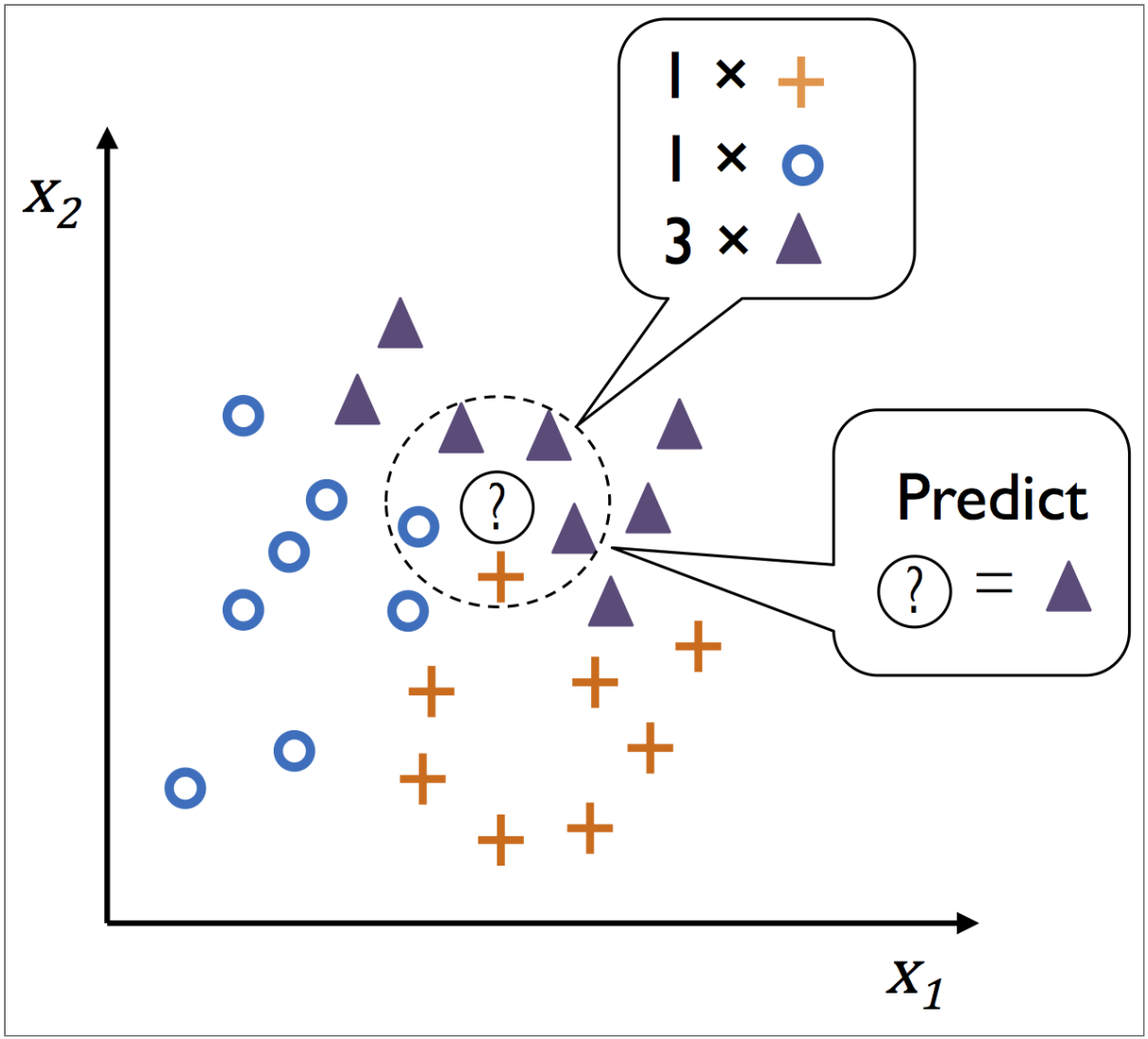
In [5]:

```
my_plot_decision_regions(X_zscore,y_c,logr,figsize=(6,6))
```


0
1
2
hue
proline
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

Distance Based: k-Nearest Neighbor (kNN)

- What category do most of the k nearest neighbors belong to?

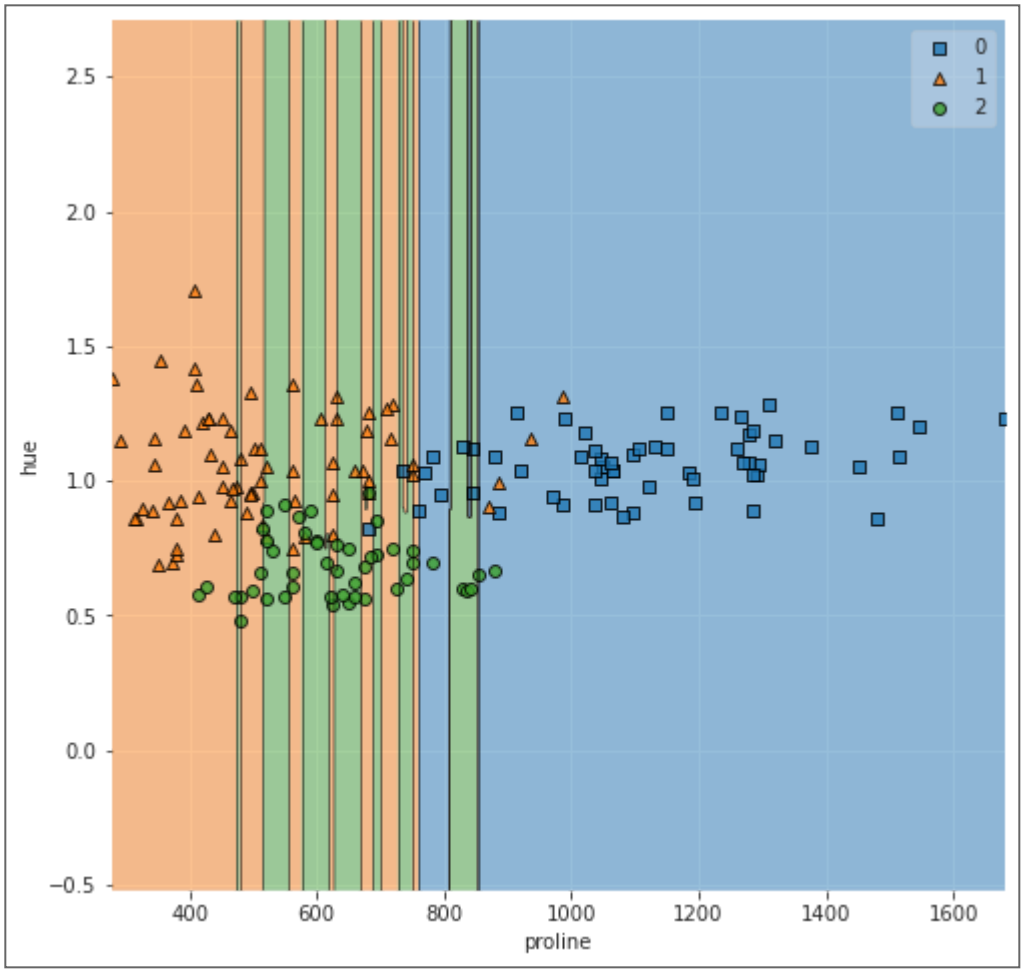


From PML

KNN in sklearn

In [6]:

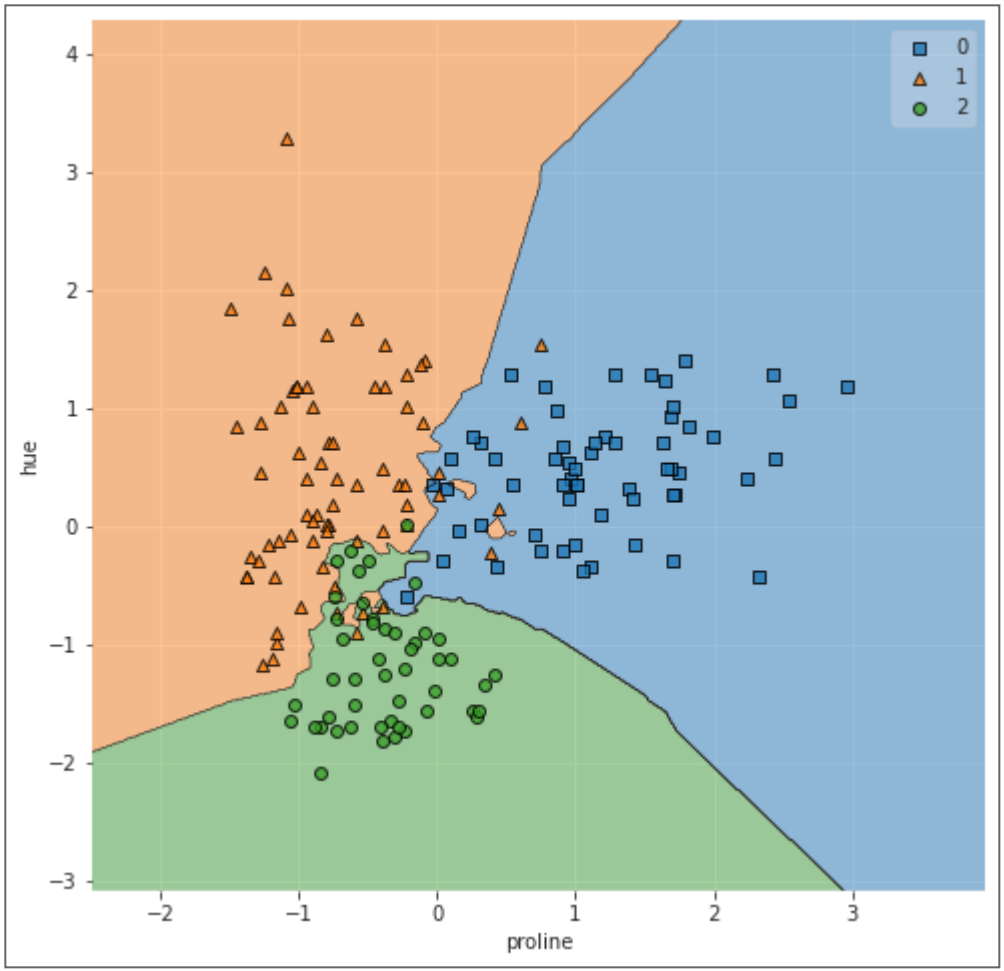
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X,y_c)

my_plot_decision_regions(X,y_c,knn)
```


0
1
2
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
hue
400
600
800
1000
1200
1400
1600
proline

Effects of Standardization on Distance Based Methods

In [7]:

```
knn_z = KNeighborsClassifier(n_neighbors=3)
knn_z.fit(X_zscore,y_c)

my_plot_decision_regions(X_zscore,y_c,knn_z)
```


0
1
2
hue
proline
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

Curse of Dimensionality

The more dimensions, the less likely points are "close" to each other.

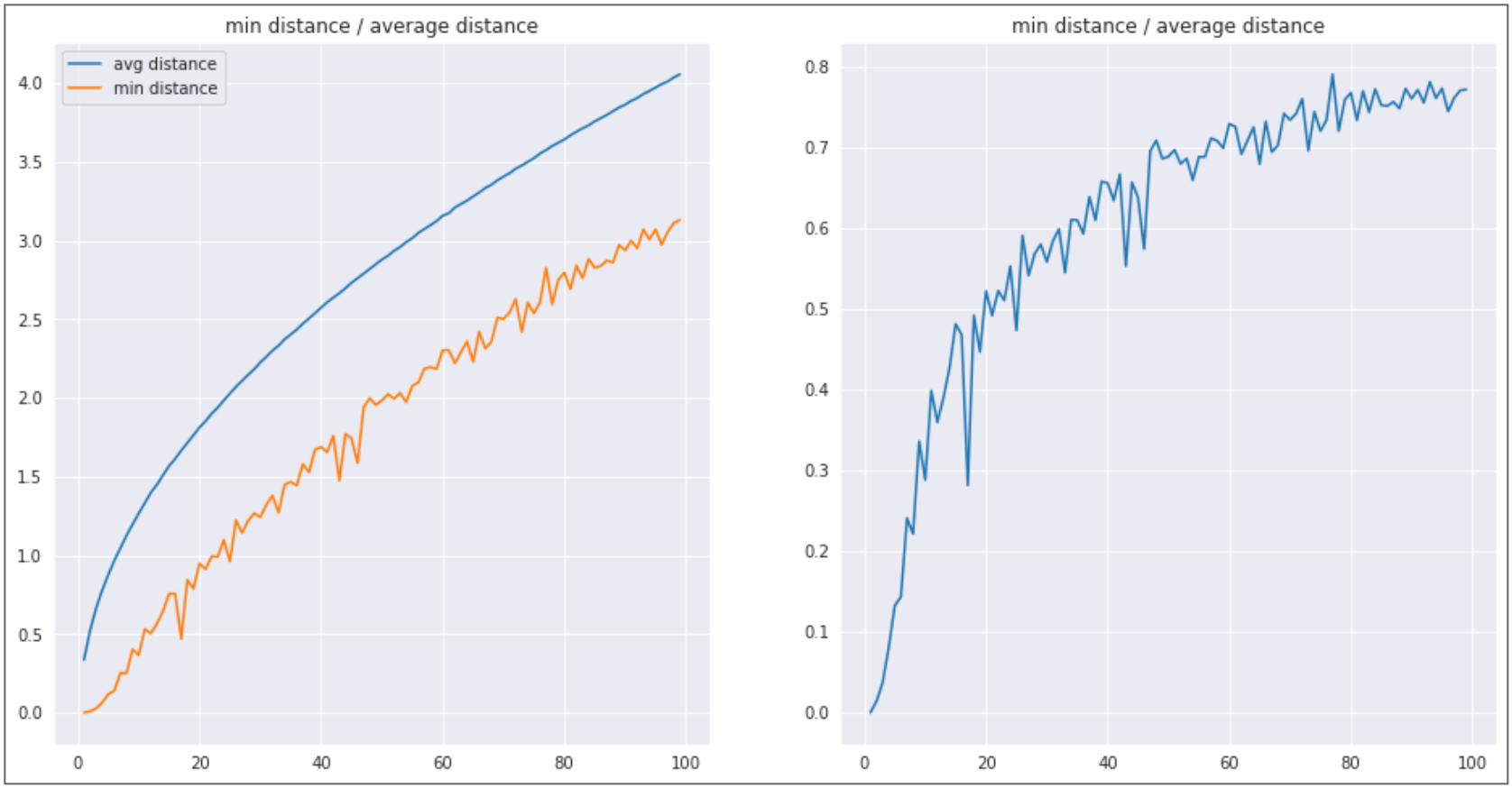
In [8]:

```
# From Data Science From Scratch by Joel Grus

def random_distances(dim,num_pairs=10_000):
    return np.sqrt(np.square(np.random.rand(num_pairs,dim) - np.random.rand(num_pairs,dim)).sum(axis=1))

# calculate average and minimum distance for 1 to 100 dimensions
dimensions = range(1,100)
avg_distances = []
min_distances = []
min_avg_ratio = []
np.random.seed(0)
for d in dimensions:
    distances = random_distances(d)
    avg_distances.append(distances.mean())
    min_distances.append(distances.min())
    min_avg_ratio.append(distances.min() / distances.mean())
```

Curse of Dimensionality Cont.

In [9]:

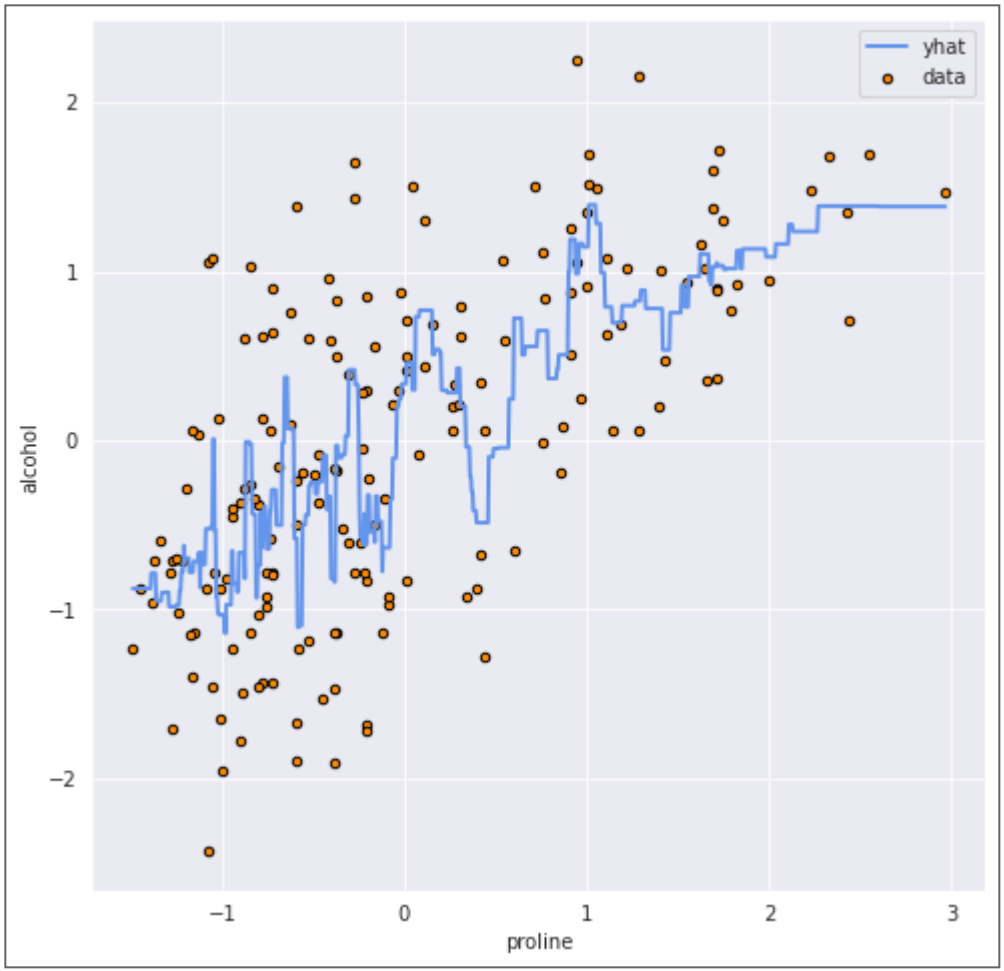
```
fig,ax = plt.subplots(1,2,figsize=(16,8))
ax[0].plot(dimensions,avg_distances,label='avg distance');
ax[0].plot(dimensions,min_distances,label='min distance');
ax[0].legend()
ax[0].set_title('min distance / average distance');
ax[1].plot(dimensions,min_avg_ratio)
ax[1].set_title('min distance / average distance');
```


min distance / average distance
avg distance
min distance
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100
min distance / average distance
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
20
40
60
80
100

Regression with kNN

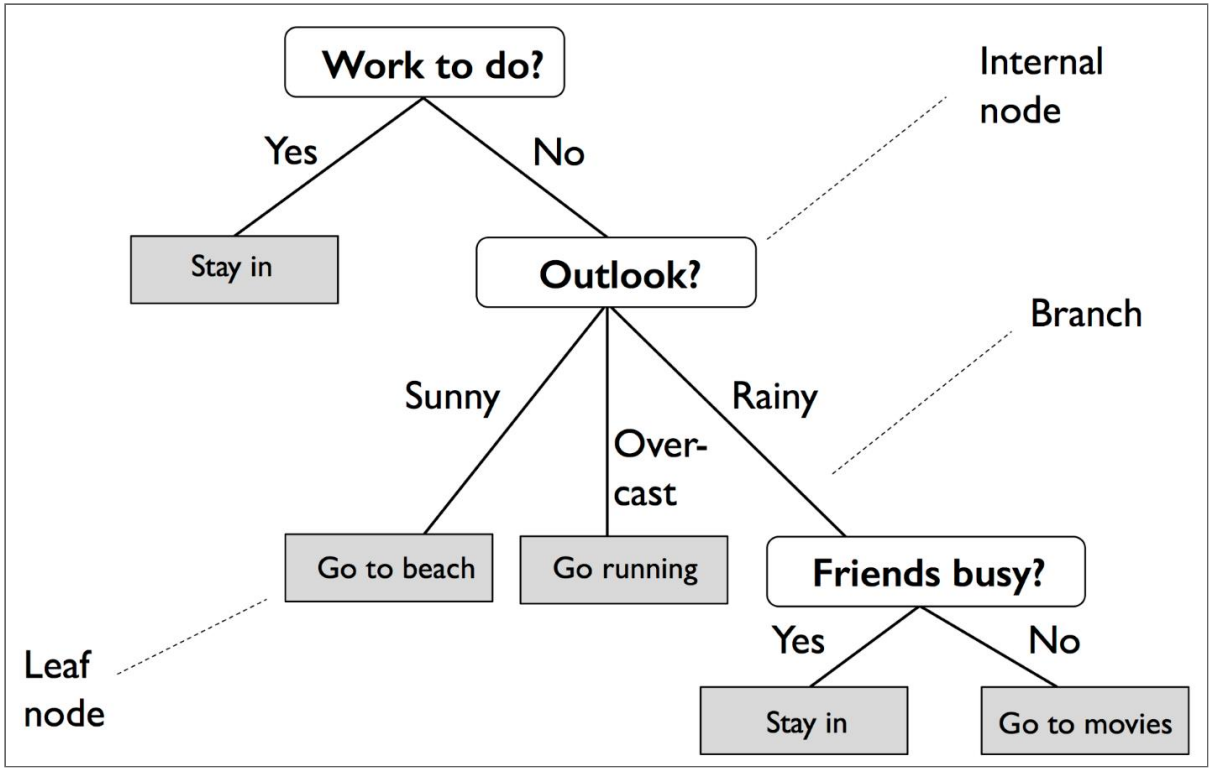
In [10]:

```
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier

knnr = KNeighborsRegressor(n_neighbors=5)
knnr.fit(X_zscore[['proline']],alcohol_zscore)

my_plot_regression(X_zscore[['proline']],alcohol_zscore,knnr)
```


yhat
data
alcohol
proline
2
1
0
−1
−2
−1
0
1
2
3

Decision Tree

- What answer does a series of yes/no questions lead us to?

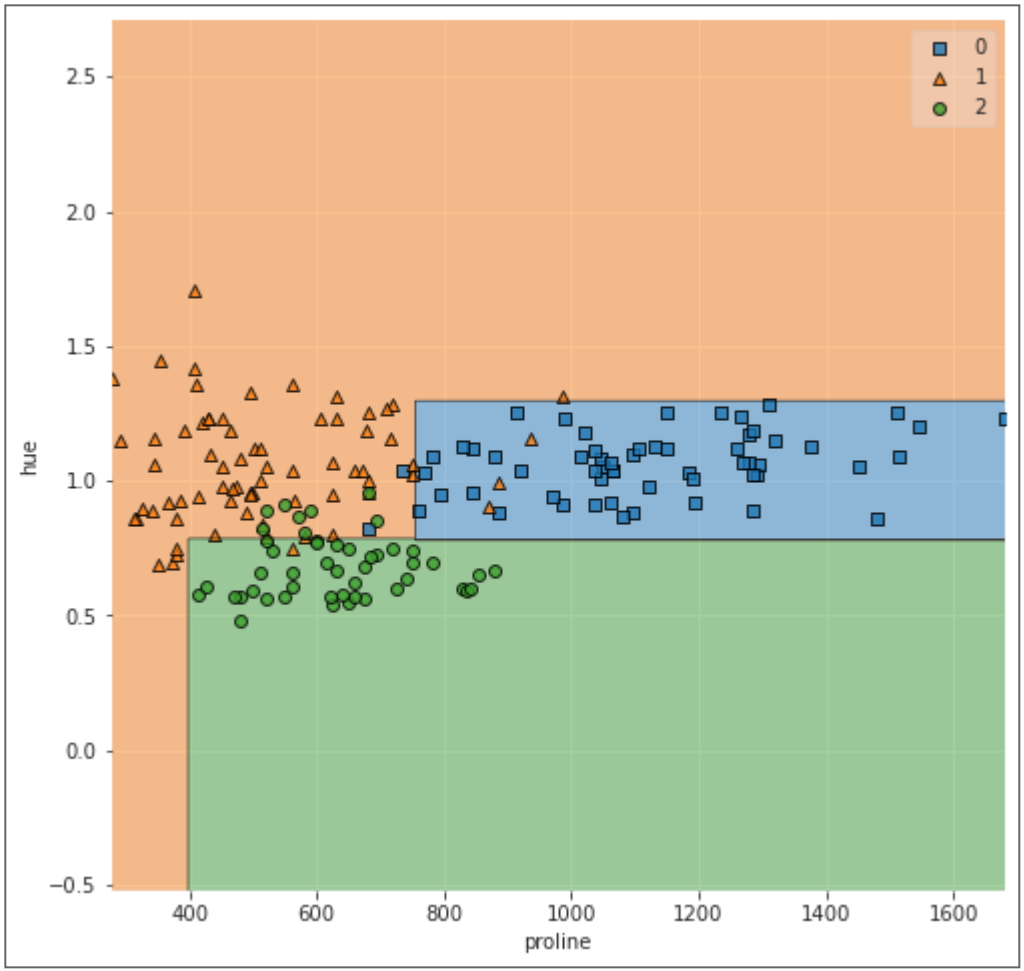


From PML

Decision Tree Classifier in sklearn

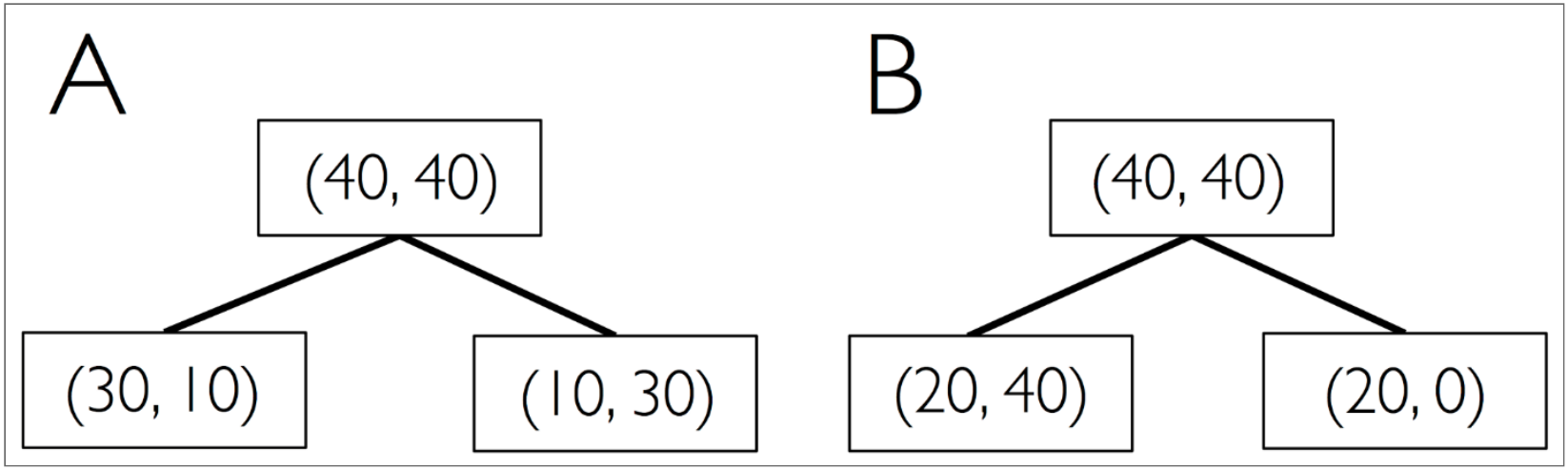
In [12]:

```
from sklearn.tree import DecisionTreeClassifier

dtc_md3 = DecisionTreeClassifier(max_depth=3) # max_depth: max number of questions
dtc_md3.fit(X,y_c)

my_plot_decision_regions(X,y_c,dtc_md3)
```


0
1
2
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
hue
400
600
800
1000
1200
1400
1600
proline

Building a Decision Tree

- How to decide which question to choose? **Reduce Impurity**

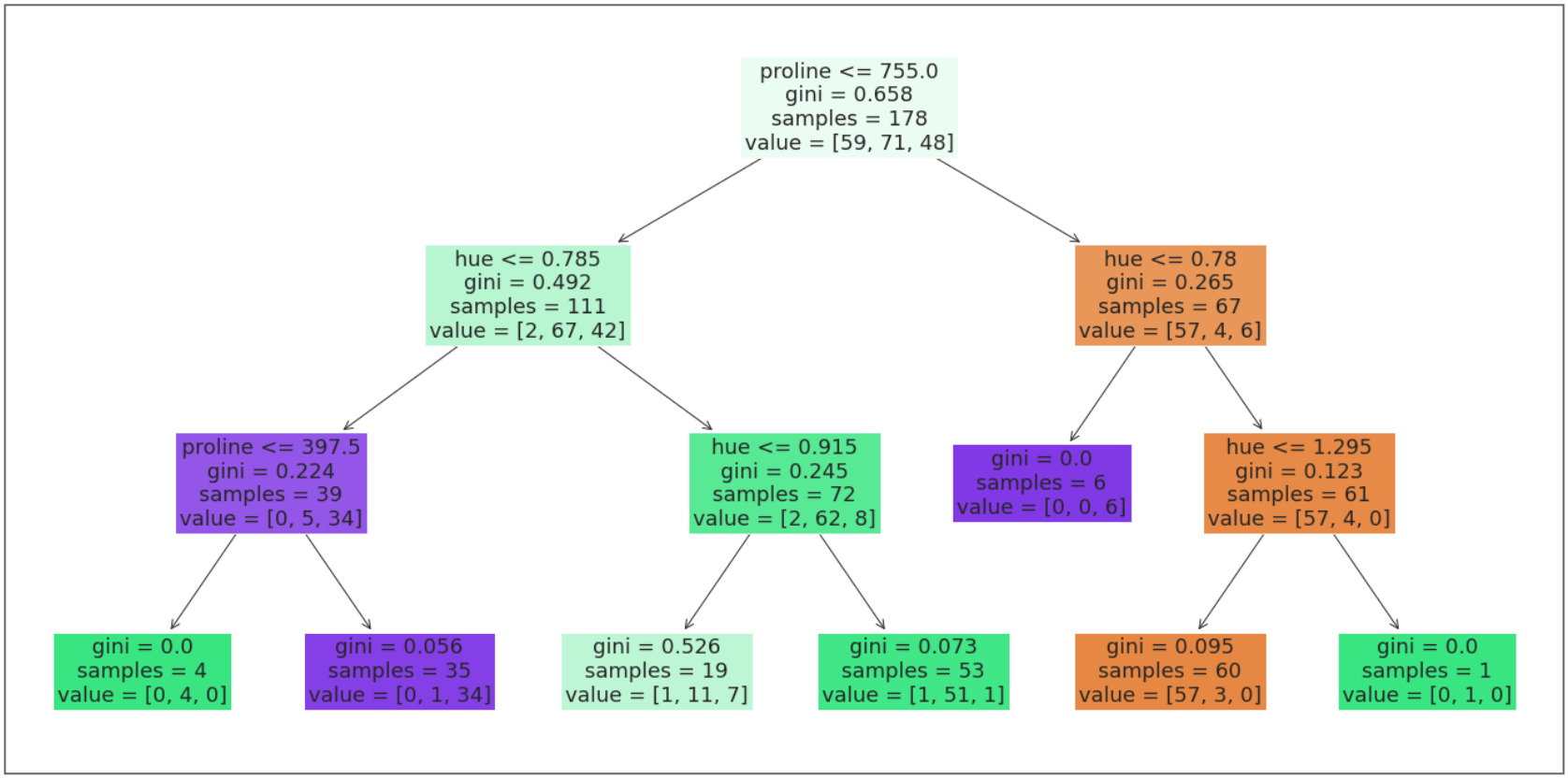


From PML

Plot Learned Decision Tree Using sklearn

In [14]:

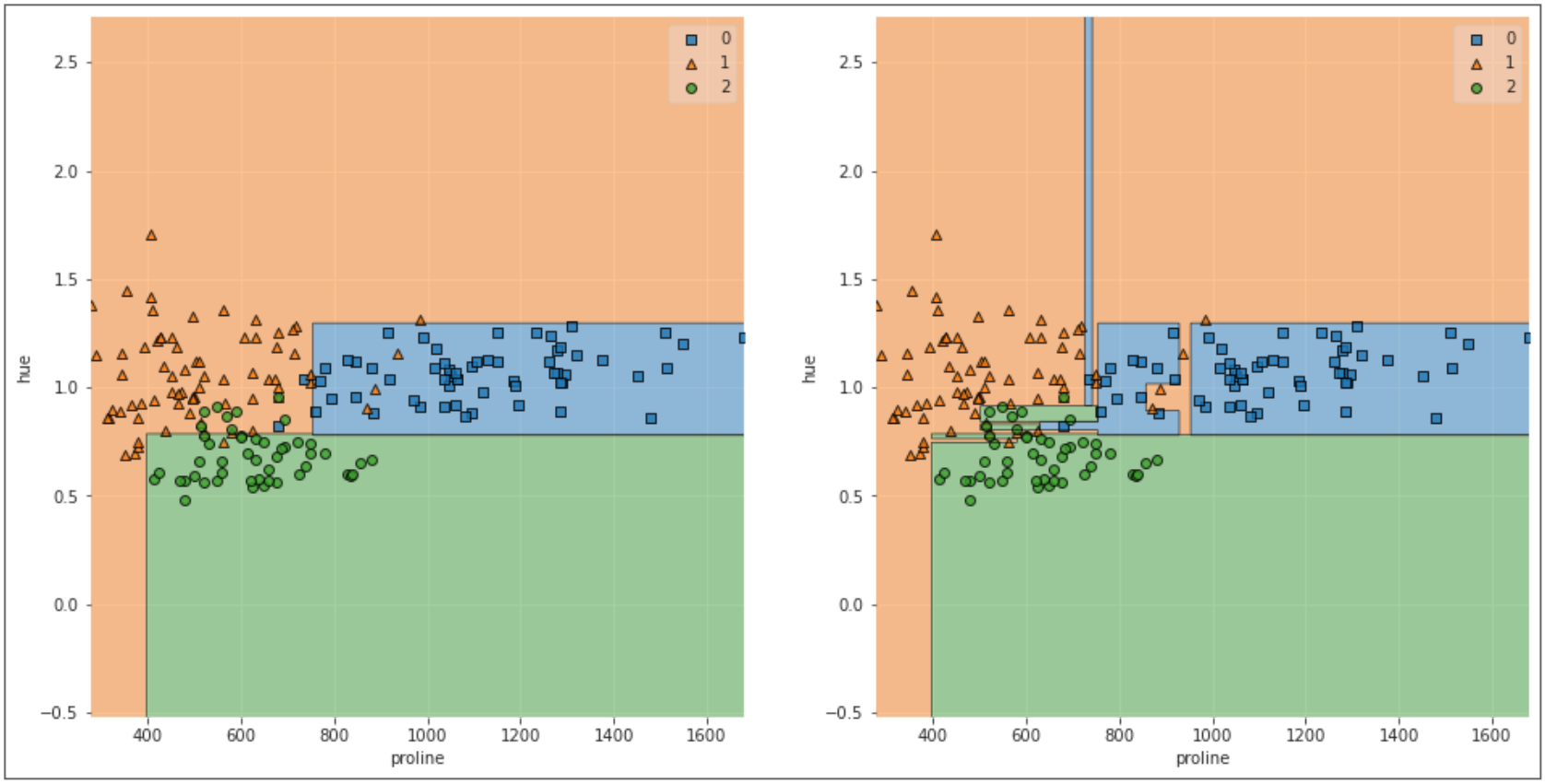
```
from sklearn.tree import plot_tree
fig,ax = plt.subplots(1,1,figsize=(24,12))
plot_tree(dtc_md3,ax=ax,fontsize=18,feature_names=X.columns,filled=True);
```



Decision Tree: Increase Maximum Depth

In [15]:

```
dtc_md10 = DecisionTreeClassifier(max_depth=10)
dtc_md10.fit(X,y_c)

fig,ax = plt.subplots(1,2,figsize=(16,8))
plot_decision_regions(X.values, y_c.values, clf=dtc_md3, ax=ax[0]);
ax[0].set_xlabel(X.columns[0]); ax[0].set_ylabel(X.columns[1]);

plot_decision_regions(X.values, y_c.values, clf=dtc_md10, ax=ax[1]);
ax[1].set_xlabel(X.columns[0]); ax[1].set_ylabel(X.columns[1]);
```

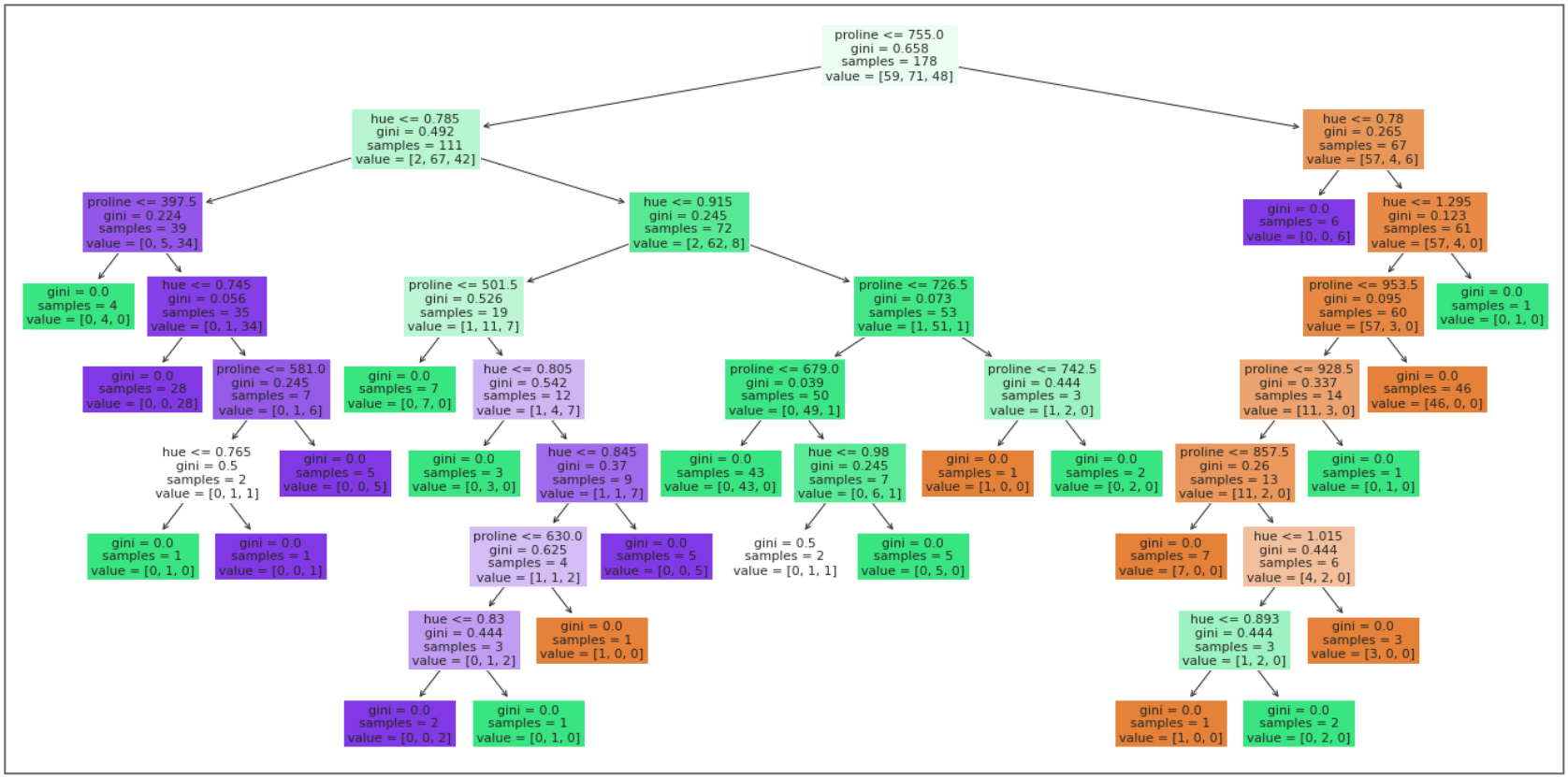

0
1
2
hue
proline
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
400
600
800
1000
1200
1400
1600

Plot Learned Decision Tree Using sklearn

- For tree with max_depth=10

In [16]:

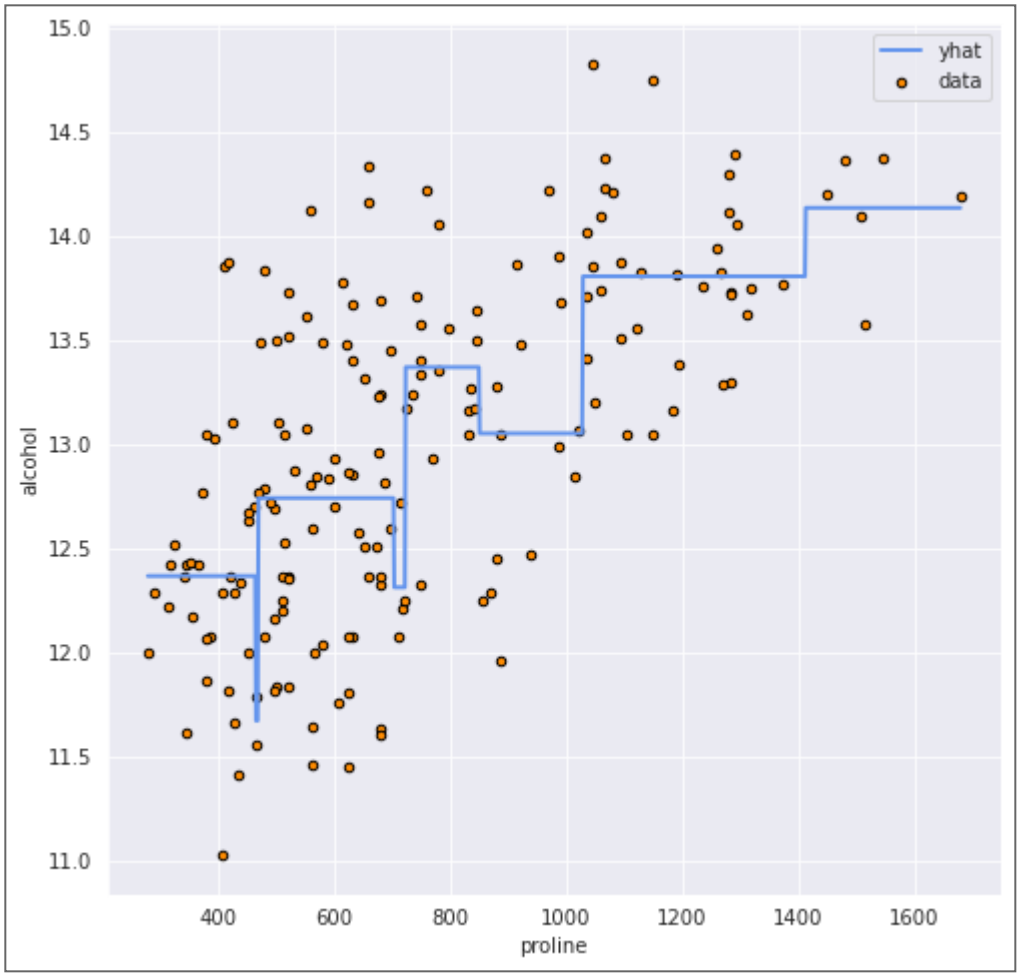
```
fig,ax = plt.subplots(1,1,figsize=(24,12))
plot_tree(dtc_md10,ax=ax,fontsize=11,feature_names=X.columns,filled=True);
```


proline <= 755.0
gini = 0.658
samples = 178
value = [59, 71, 48]
hue <= 0.785
gini = 0.492
samples = 111
value = [2, 67, 42]
hue <= 0.78
gini = 0.265
samples = 67
value = [57, 4, 6]
proline <= 397.5
gini = 0.224
samples = 39
value = [0, 5, 34]
hue <= 0.915
gini = 0.245
samples = 72
value = [2, 62, 8]
gini = 0.0
samples = 6
value = [0, 0, 6]
hue <= 1.295
gini = 0.123
samples = 61
value = [57, 4, 0]
gini = 0.0
samples = 4
value = [0, 4, 0]
hue <= 0.745
gini = 0.056
samples = 35
value = [0, 1, 34]
proline <= 501.5
gini = 0.526
samples = 19
value = [1, 11, 7]
proline <= 726.5
gini = 0.073
samples = 53
value = [1, 51, 1]
proline <= 953.5
gini = 0.095
samples = 60
value = [57, 3, 0]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 28
value = [0, 0, 28]
proline <= 581.0
gini = 0.245
samples = 7
value = [0, 1, 6]
gini = 0.0
samples = 7
value = [0, 7, 0]
hue <= 0.805
gini = 0.542
samples = 12
value = [1, 4, 7]
proline <= 679.0
gini = 0.039
samples = 50
value = [0, 49, 1]
proline <= 742.5
gini = 0.444
samples = 3
value = [1, 2, 0]
proline <= 928.5
gini = 0.337
samples = 14
value = [11, 3, 0]
gini = 0.0
samples = 46
value = [46, 0, 0]
hue <= 0.765
gini = 0.5
samples = 2
value = [0, 1, 1]
gini = 0.0
samples = 5
value = [0, 0, 5]
gini = 0.0
samples = 3
value = [0, 3, 0]
hue <= 0.845
gini = 0.37
samples = 9
value = [1, 1, 7]
gini = 0.0
samples = 43
value = [0, 43, 0]
hue <= 0.98
gini = 0.245
samples = 7
value = [0, 6, 1]
gini = 0.0
samples = 1
value = [1, 0, 0]
gini = 0.0
samples = 2
value = [0, 2, 0]
proline <= 857.5
gini = 0.26
samples = 13
value = [11, 2, 0]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
proline <= 630.0
gini = 0.625
samples = 4
value = [1, 1, 2]
gini = 0.0
samples = 5
value = [0, 0, 5]
gini = 0.5
samples = 2
value = [0, 1, 1]
gini = 0.0
samples = 5
value = [0, 5, 0]
gini = 0.0
samples = 7
value = [7, 0, 0]
hue <= 1.015
gini = 0.444
samples = 6
value = [4, 2, 0]
hue <= 0.83
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 1
value = [1, 0, 0]
hue <= 0.893
gini = 0.444
samples = 3
value = [1, 2, 0]
gini = 0.0
samples = 3
value = [3, 0, 0]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 1
value = [1, 0, 0]
gini = 0.0
samples = 2
value = [0, 2, 0]

Regression with Decision Trees

In [17]:

```
from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor(max_depth=3)
dtr.fit(X[['proline']],df_wine.alcohol)

my_plot_regression(X[['proline']],df_wine.alcohol,dtr)
```


yhat
data
alcohol
proline
15.0
14.5
14.0
13.5
13.0
12.5
12.0
11.5
11.0
400
600
800
1000
1200
1400
1600

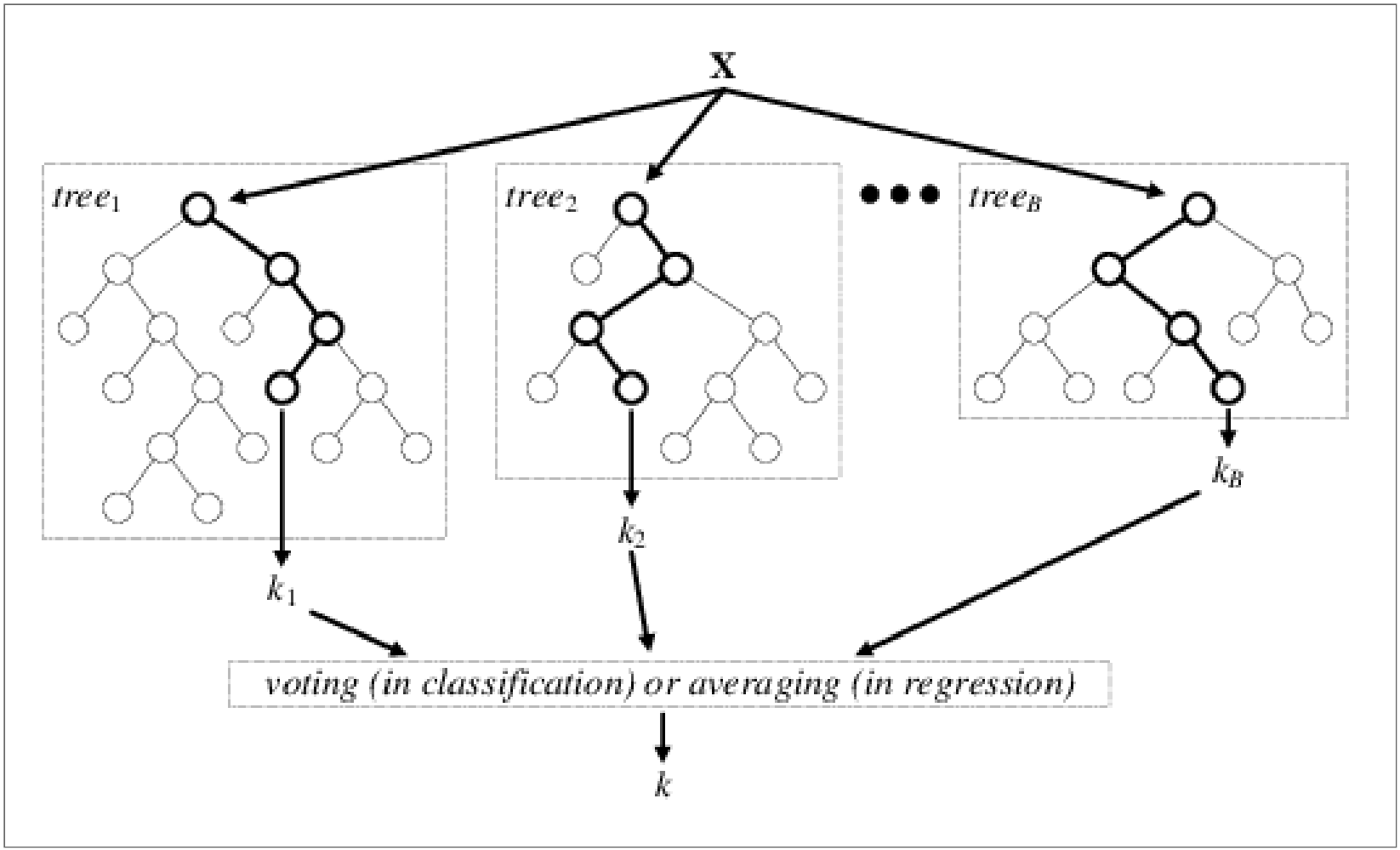
Ensemble Methods

- "Wisdom of the crowd"
- Can often achieve better performance with collection of learners
- Often use shallow trees as base learners

Common methods for generating ensembles:

- **Bagging** (Bootstrap Aggregation)
    - Random Forest
- **Boosting**
    - Gradient Boosting
- **Stacking**

Random Forest and Gradient Boosted Trees



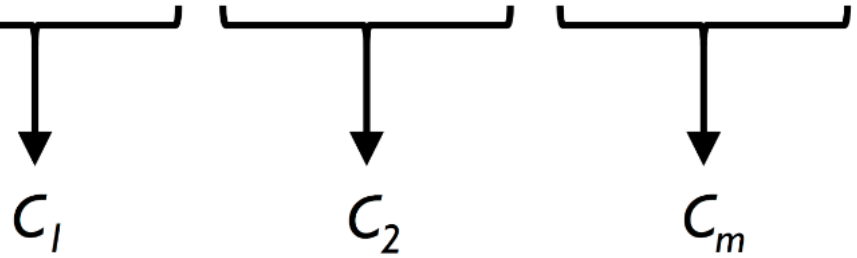
From

https://www.researchgate.net/publication/301638643_Electromyographic_Patterns_during_Golf_Swing_Act

Bagging with Random Forests

- Trees built with bootstrap samples and subsets of features
- Achieve variation with random selection of observations and features

Sample indices	Bagging round 1	Bagging round 2	…
1	2	7	…
2	2	3	…
3	1	2	…
4	3	1	…
5	7	1	…
6	2	7	…
7	4	7	…

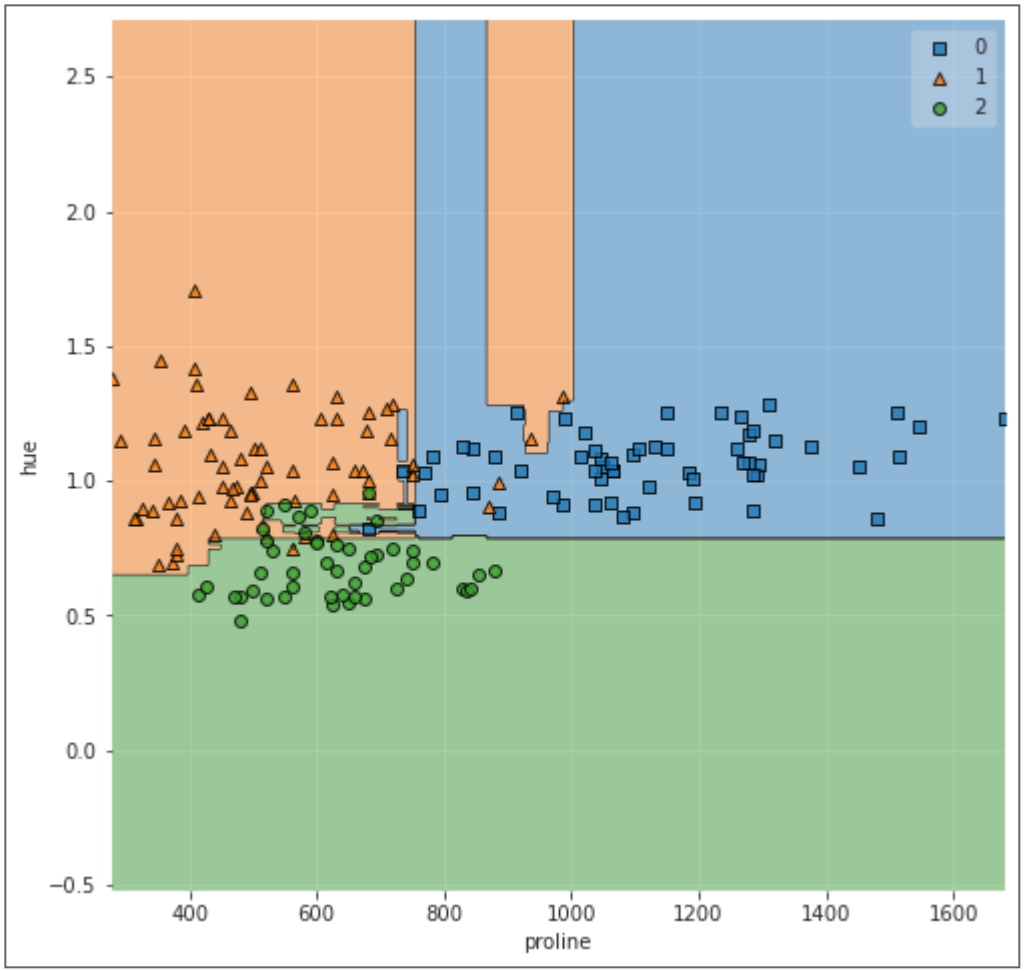


From PML

Random Forests with sklearn

In [18]:

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=10, # number of trees in ensemble
                             n_jobs=-1,       # parallelize using all available cores
                             random_state=0   # for demonstration only
                            )
rfc.fit(X,y_c)

my_plot_decision_regions(X,y_c,rfc)
```


0
1
2
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
hue
400
600
800
1000
1200
1400
1600
proline

Regression with RandomForests

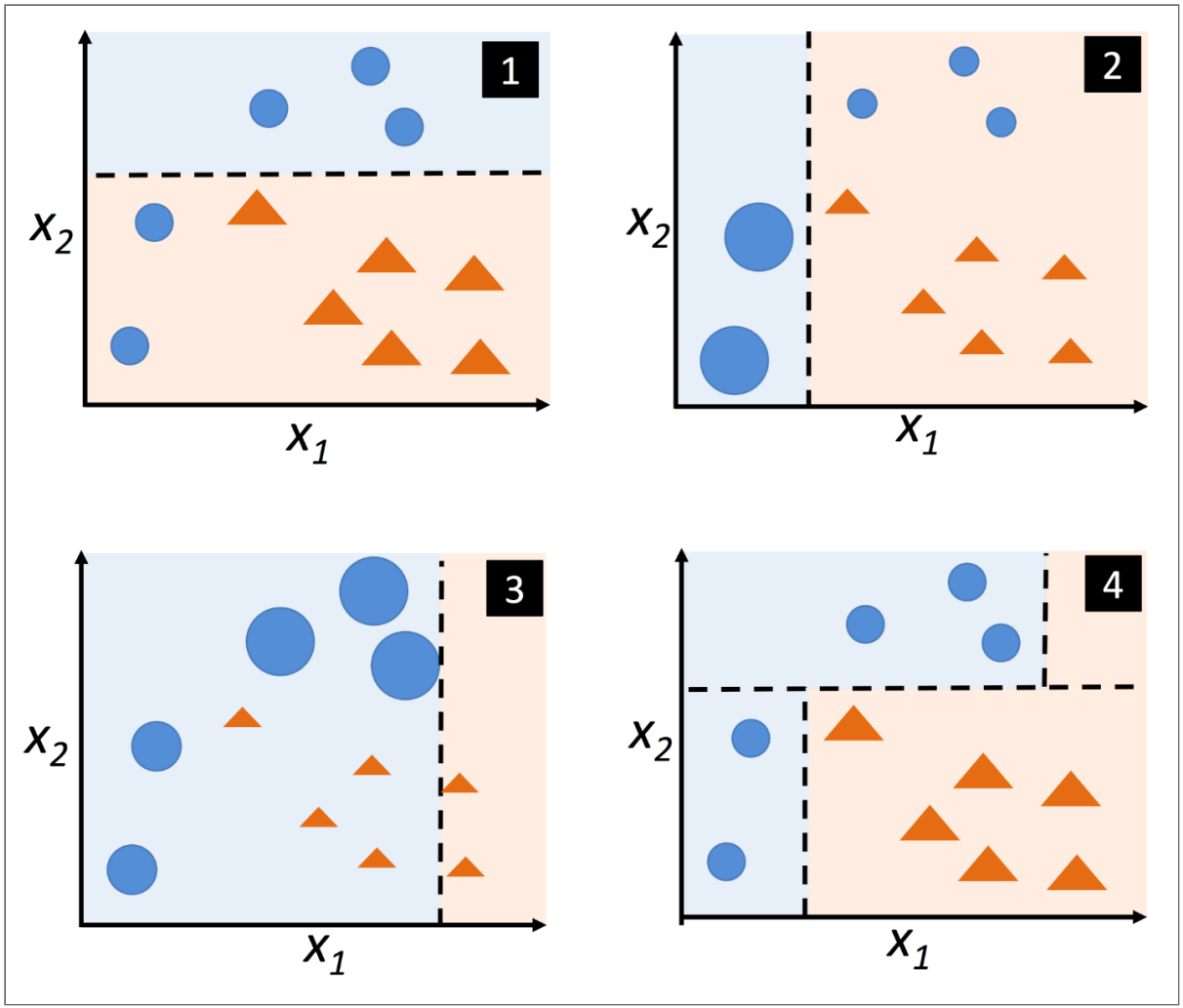
In [19]:

```
from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(n_estimators=3, n_jobs=-1)
rfr.fit(df_wine[['proline']],df_wine.alcohol)

my_plot_regression(df_wine[['proline']],df_wine.alcohol,rfr)
```


yhat
data
alcohol
proline
15.0
14.5
14.0
13.5
13.0
12.5
12.0
11.5
11.0
400
600
800
1000
1200
1400
1600

Gradient Boosted Trees

- Trees built by adding weight to mis-classification
- Achieve variation due to changes in weights on observations

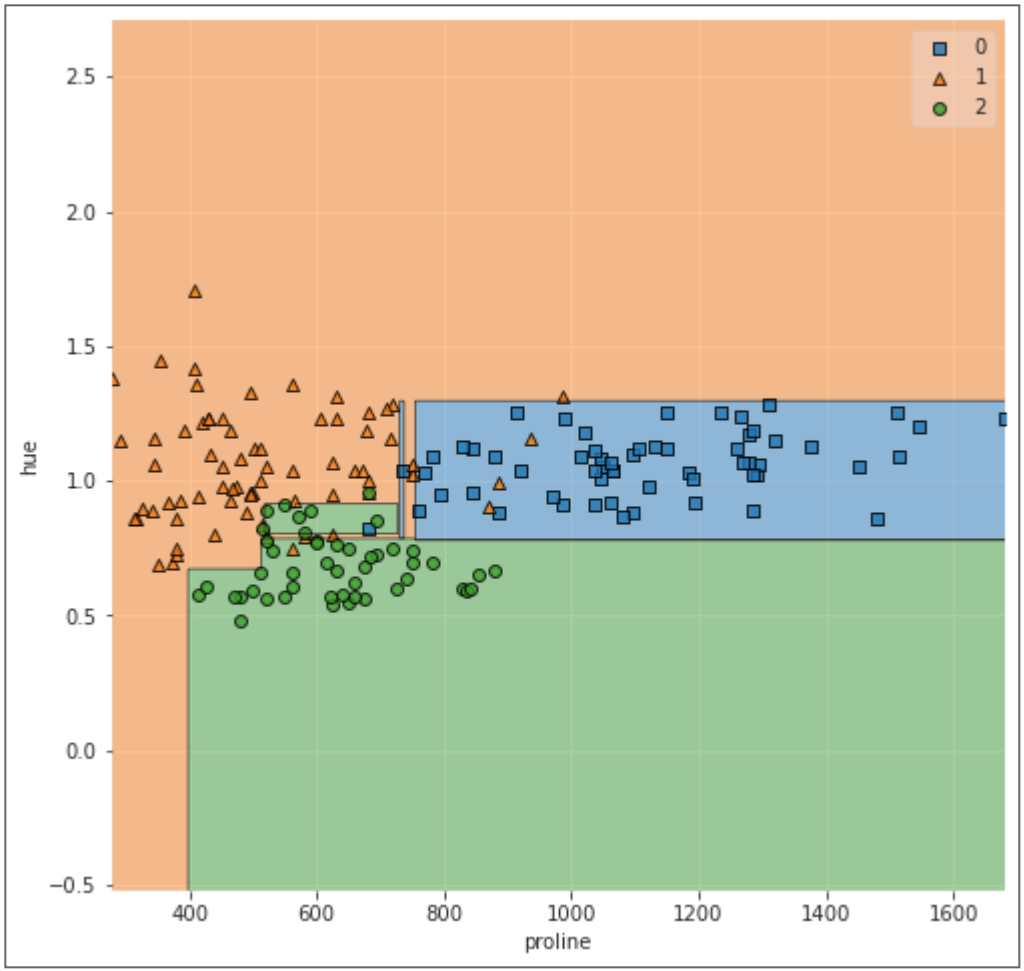


From PML

Gradient Boosted Trees in sklearn

In [20]:

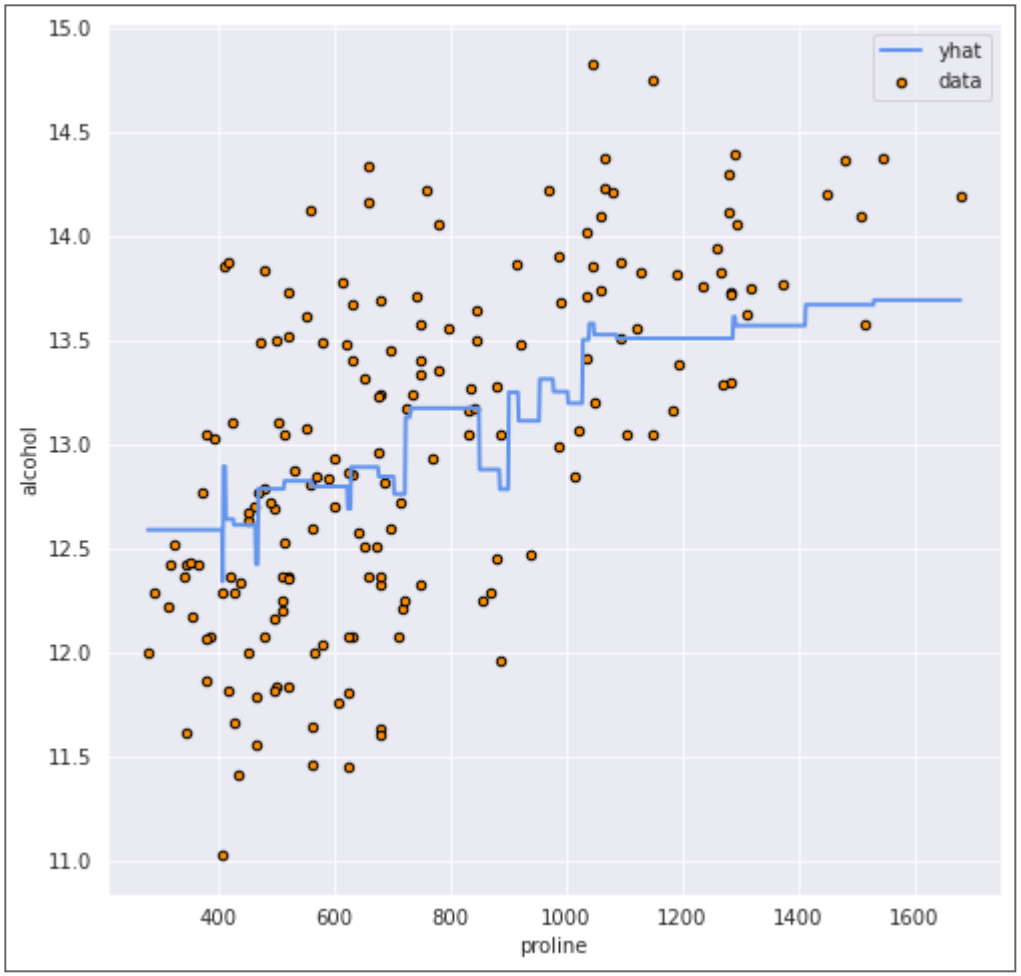
```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=10)
gbc.fit(X,y_c)

my_plot_decision_regions(X,y_c,gbc)
```


0
1
2
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
hue
400
600
800
1000
1200
1400
1600
proline

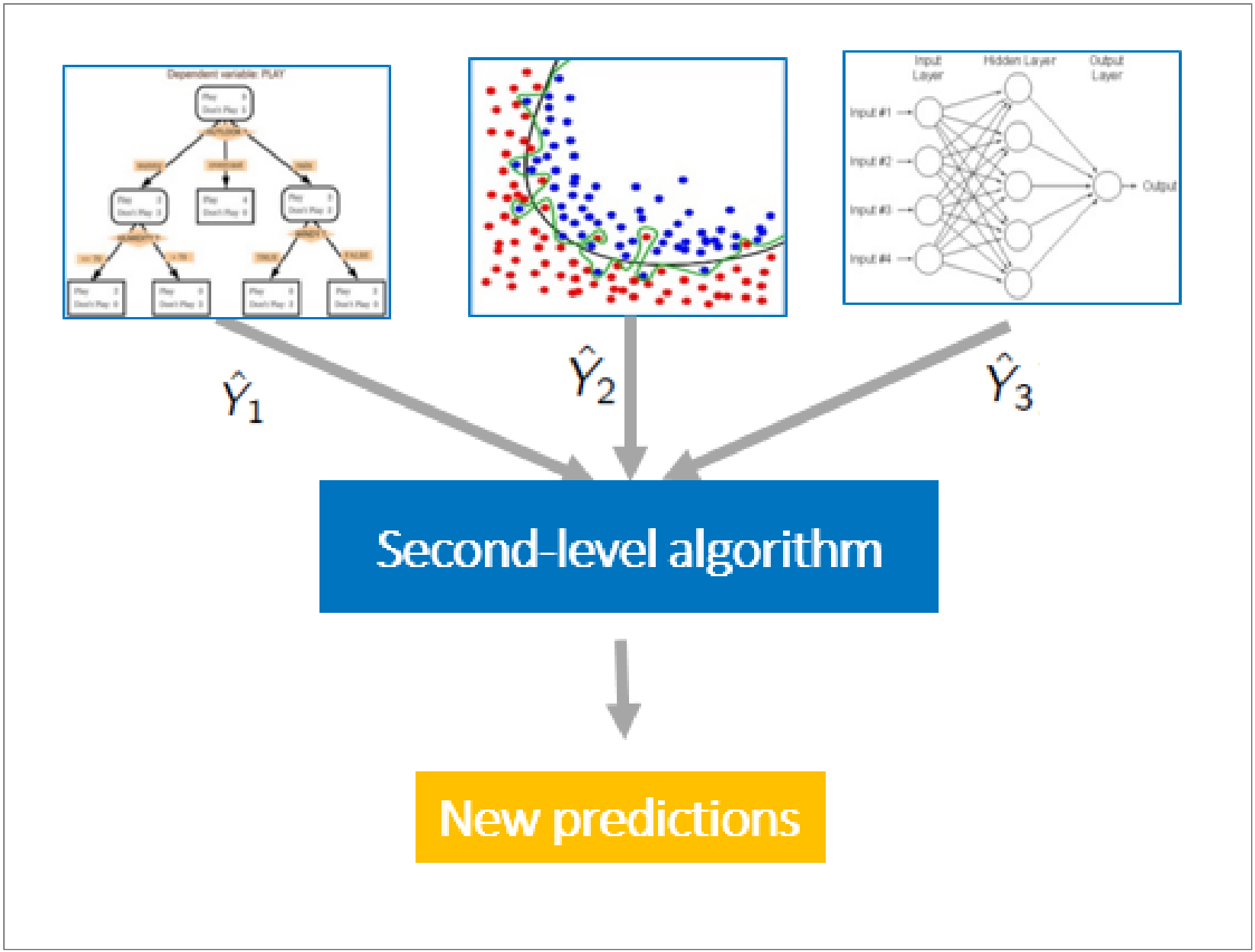
Regression with Gradient Boosted Trees

In [21]:

```
from sklearn.ensemble import GradientBoostingRegressor

gbr = GradientBoostingRegressor(n_estimators=10)
gbr.fit(df_wine.proline.values.reshape(-1,1),df_wine.alcohol)

my_plot_regression(df_wine[['proline']],df_wine.alcohol,gbr)
```


yhat
data
alcohol
proline
15.0
14.5
14.0
13.5
13.0
12.5
12.0
11.5
11.0
400
600
800
1000
1200
1400
1600

Stacking

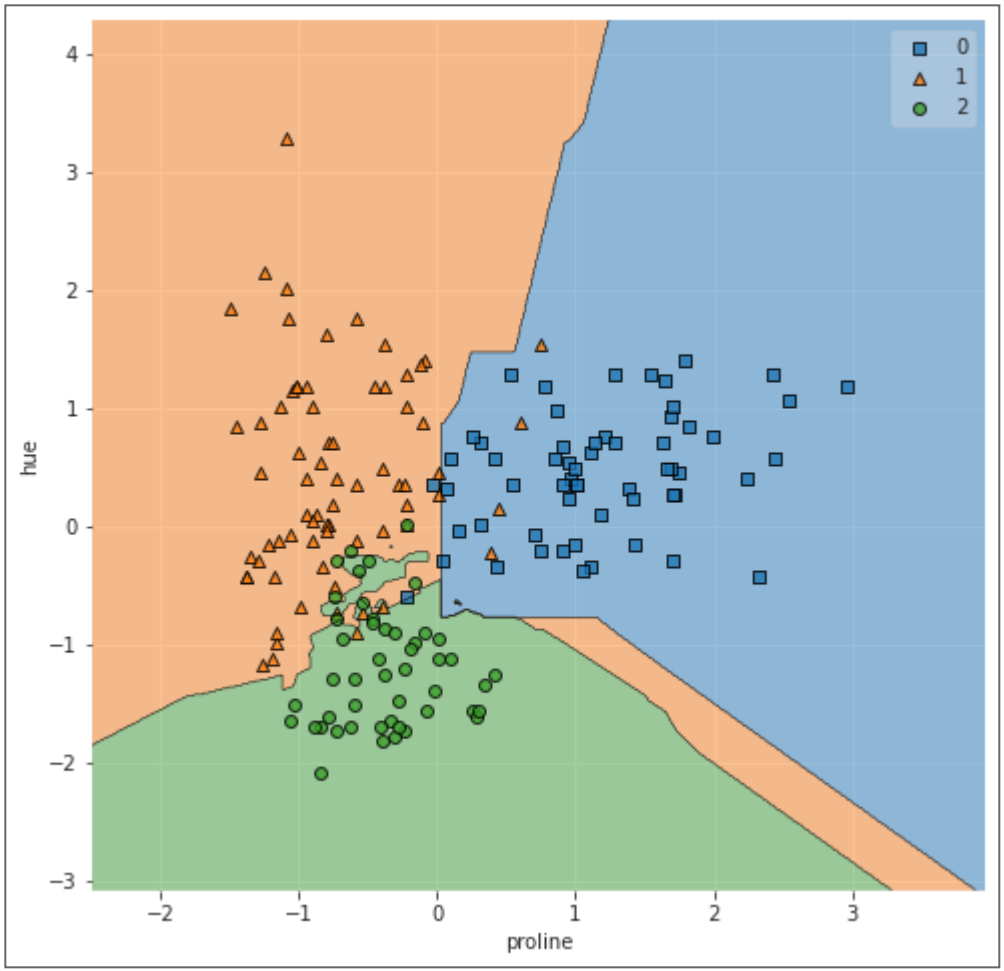


From https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

Stacking for Classification

In [22]:

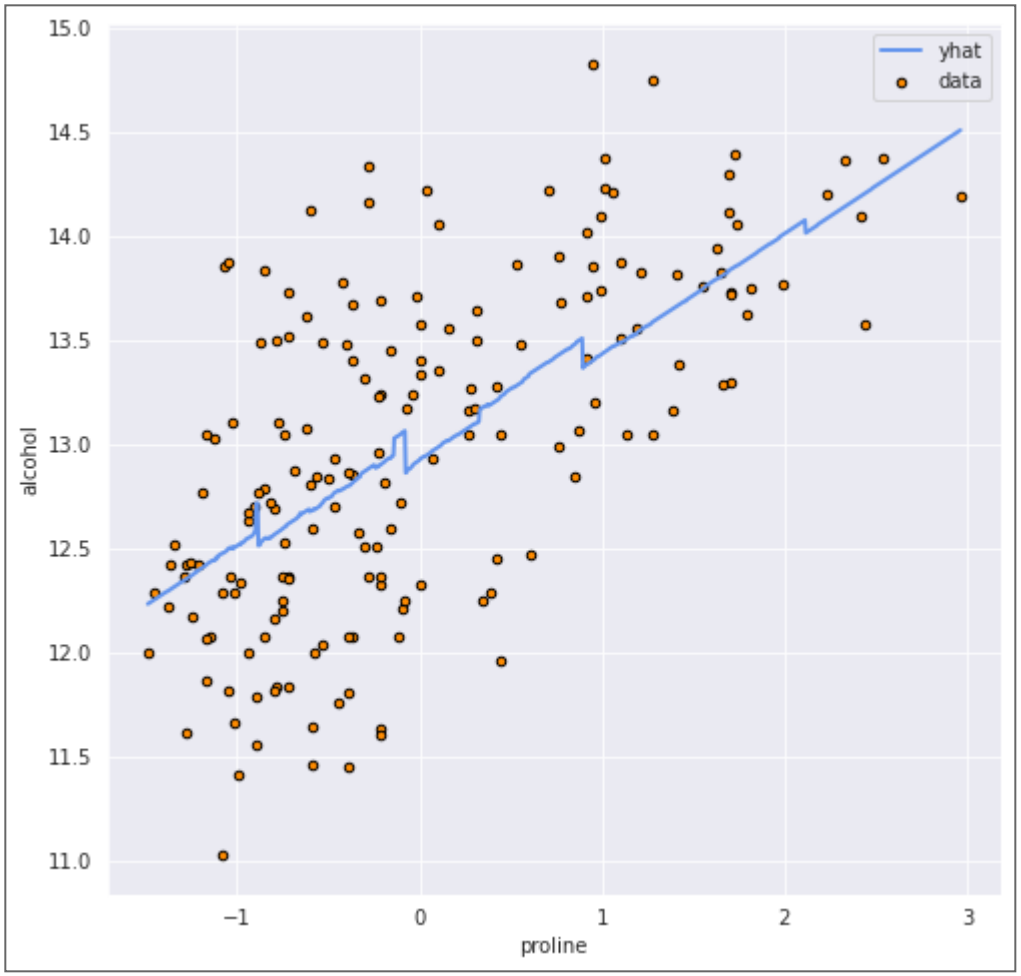
```
from sklearn.ensemble import StackingClassifier

ensemble = [('lr',LogisticRegression(max_iter=1000)),
            ('dt',DecisionTreeClassifier(max_depth=3)),
            ('knn',KNeighborsClassifier(n_neighbors=3))]

stackc = StackingClassifier(estimators=ensemble,
                            final_estimator=LogisticRegression())
stackc.fit(X_zscore,y_c)

my_plot_decision_regions(X_zscore,y_c,stackc)
```


0
1
2
hue
proline
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

Stacking for Regression

In [23]:

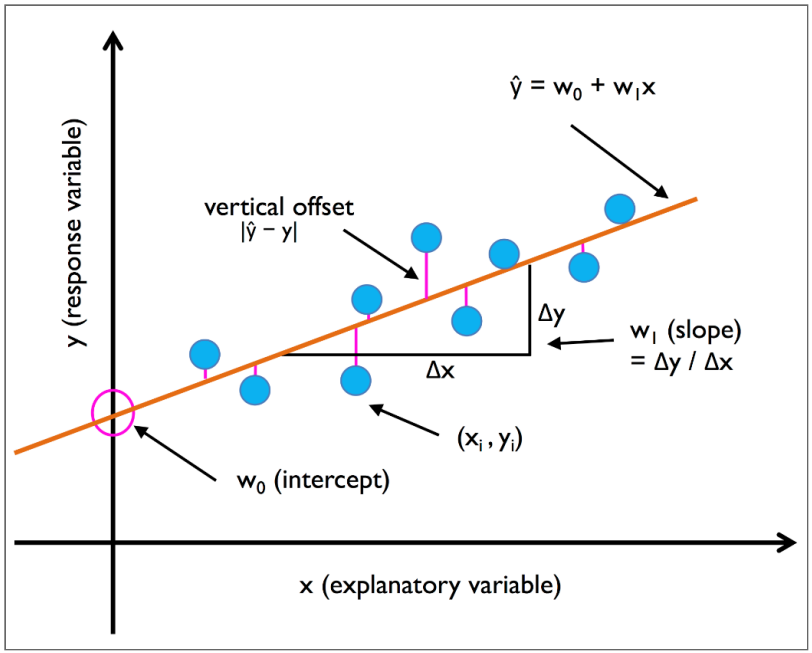
```
from sklearn.ensemble import StackingRegressor

ensemble = [('lr',LinearRegression()),
            ('dt',DecisionTreeRegressor(max_depth=3)),
            ('knn',KNeighborsRegressor(n_neighbors=6))]

stackr = StackingRegressor(estimators=ensemble,
                           final_estimator=LinearRegression())
stackr.fit(X_zscore[['proline']],df_wine.alcohol)

my_plot_regression(X_zscore[['proline']],df_wine.alcohol,stackr)
```


yhat
data
alcohol
proline
15.0
14.5
14.0
13.5
13.0
12.5
12.0
11.5
11.0
−1
0
1
2
3

Review of Models

Model Review: Simple/Multiple Linear Regression

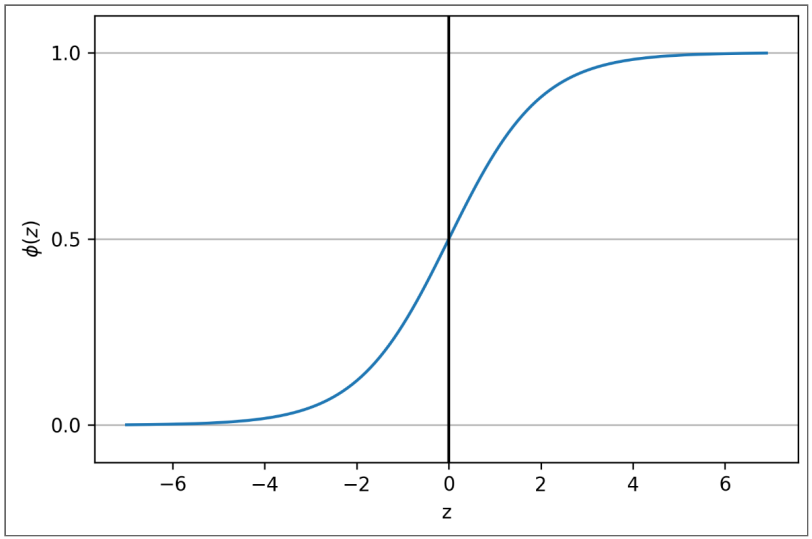
- Use for: Regression
- Pros:
    - fast to train
    - interpretable coefficients
- Cons:
    - assumes linear relationship
    - depends on removing colinear features


ŷ = w0 + w1x
vertical offset
|ŷ − y|
y (response variable)
Δy
Δx
w1 (slope)
= Δy / Δx
(xi , yi)
w0 (intercept)
x (explanatory variable)

from PML

Model Review: Logistic Regression

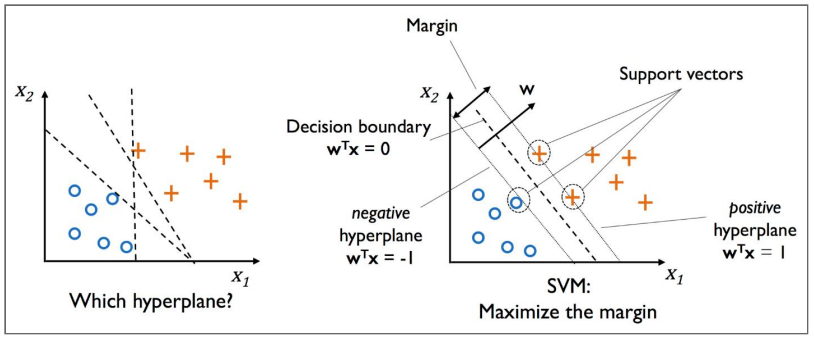
- Use for: Classification
- Pros:
    - fast to train
    - interpretable coefficients (log odds)
- Cons:
    - assumes linear boundary
    - depends on removing colinear features



from PML

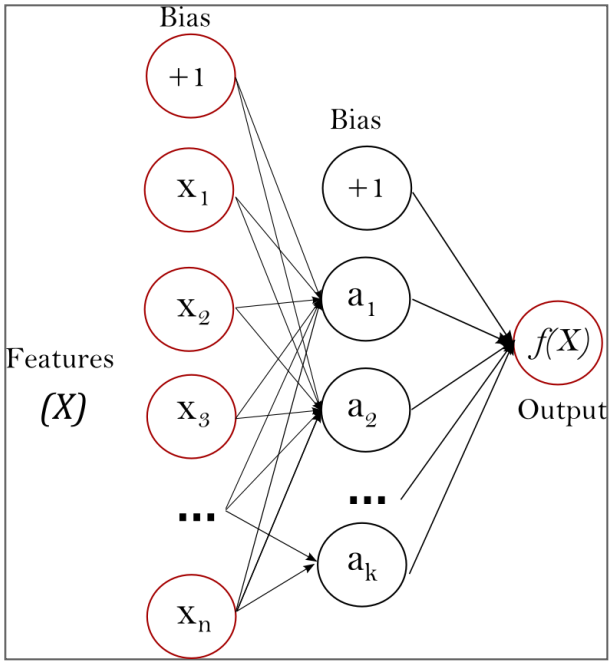
Model Review: Support Vector Machine (SVM)

- Use for: Classification and Regression
- Pros:
  - fast to evaluate
  - can use kernel trick to learn non-linear functions
- Cons:
  - slow to train
  - can fail to converge on very large datasets



Model Review: Multi-Layer Perceptron

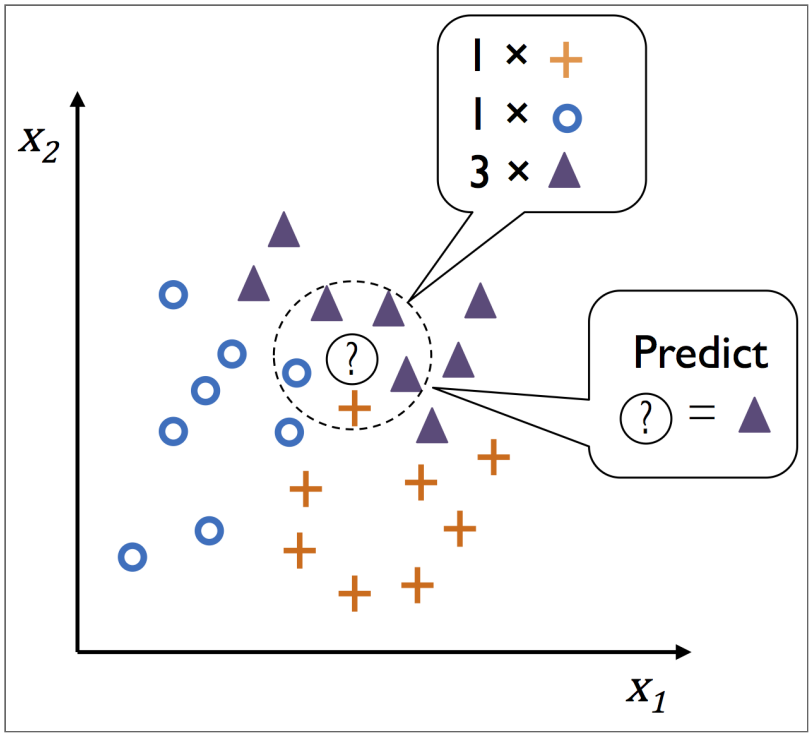
- Use for Classification or Regression
- Pros:
    - non-linear boundary
- Cons:
    - non-convex loss function (sensitive to initial weights)
    - sensitive to feature scaling
    - no GPU support in sklearn: use tensorflow or pytorch



From **https://scikit-learn.org/stable/_images/multilayerperceptron_network.png**

Model Review: k Nearest Neighbor (kNN)

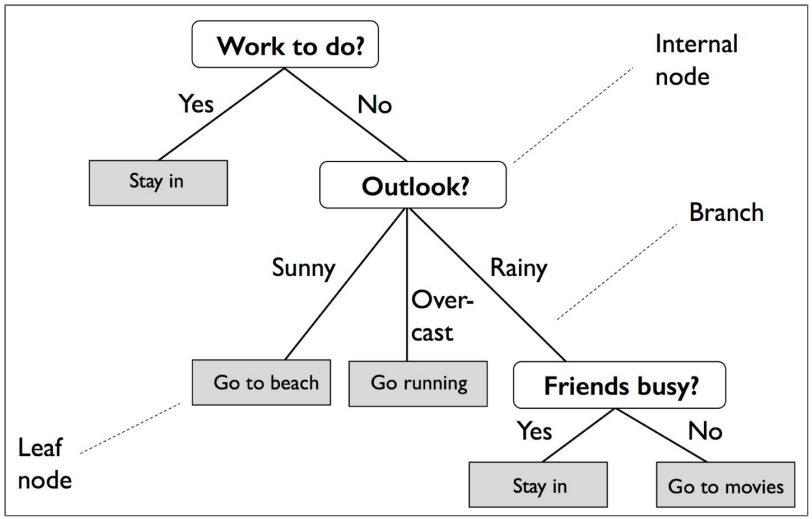
- Use for: Classification or Regression
- Pros:
    - fast to train
    - non-linear boundary
- Cons:
    - potentially slow to predict
    - curse of dimensionality



From PML

Model Review: Decision Tree

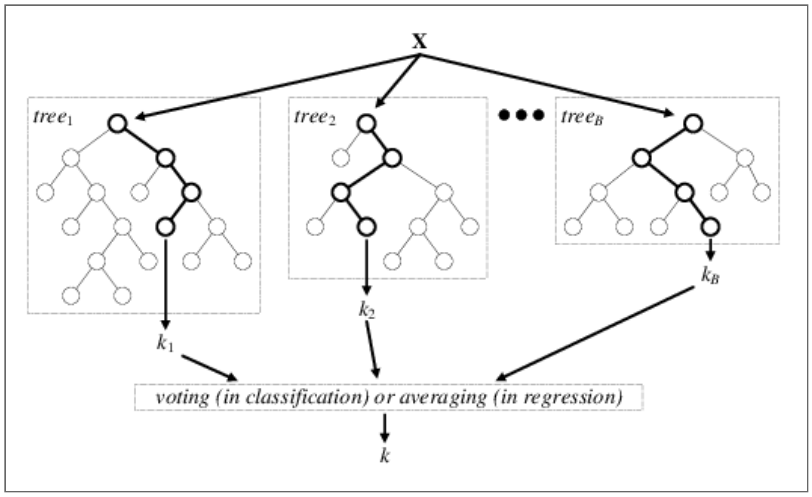
- Use for: Classification or Regression
- Pros:
    - very interpretable
    - quick to predict
    - can handle numeric and categorical variables without transformation
- Cons:
    - tendency to overfit (learn training set too well, more next class!)


Work to do?
Yes
No
Stay in
Outlook?
Internal node
Branch
Sunny
Over-cast
Rainy
Go to beach
Go running
Friends busy?
Yes
No
Stay in
Go to movies
Leaf node

From PML

Model Review: Random Forest (Ensemble via Bagging)

- Use for: Classification or Regression
- Pros:
    - less likely to overfit than decision tree
    - quick to train (through parallelization, quick to predict
- Cons:
    - less interpretible, though still possible

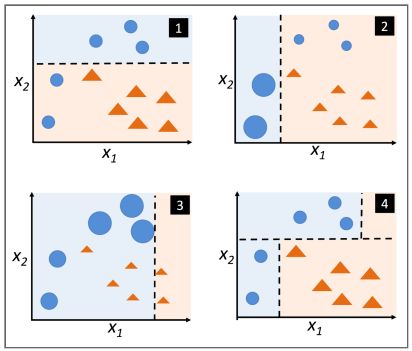


From

https://www.researchgate.net/publication/301638643_Electromyographic_Patterns_during_Golf_Swing_Act

Model Review: Gradient Boosted Trees (Ensemble via Boosting)

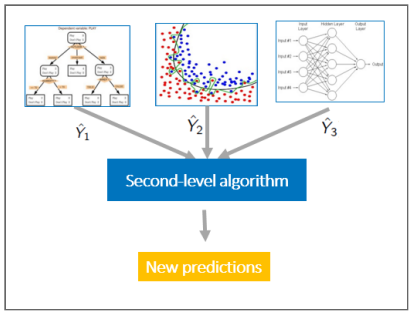
- Use for: Classification or Regression
- Pros:
    - pays more attention to difficult decision regions
    - quick to predict
    - tends to work well on difficult tasks
- Cons:
    - slow to train (parallelization not possible)
    - less interpretible, though still possible



From PML

Model Review: Ensemble via Stacking

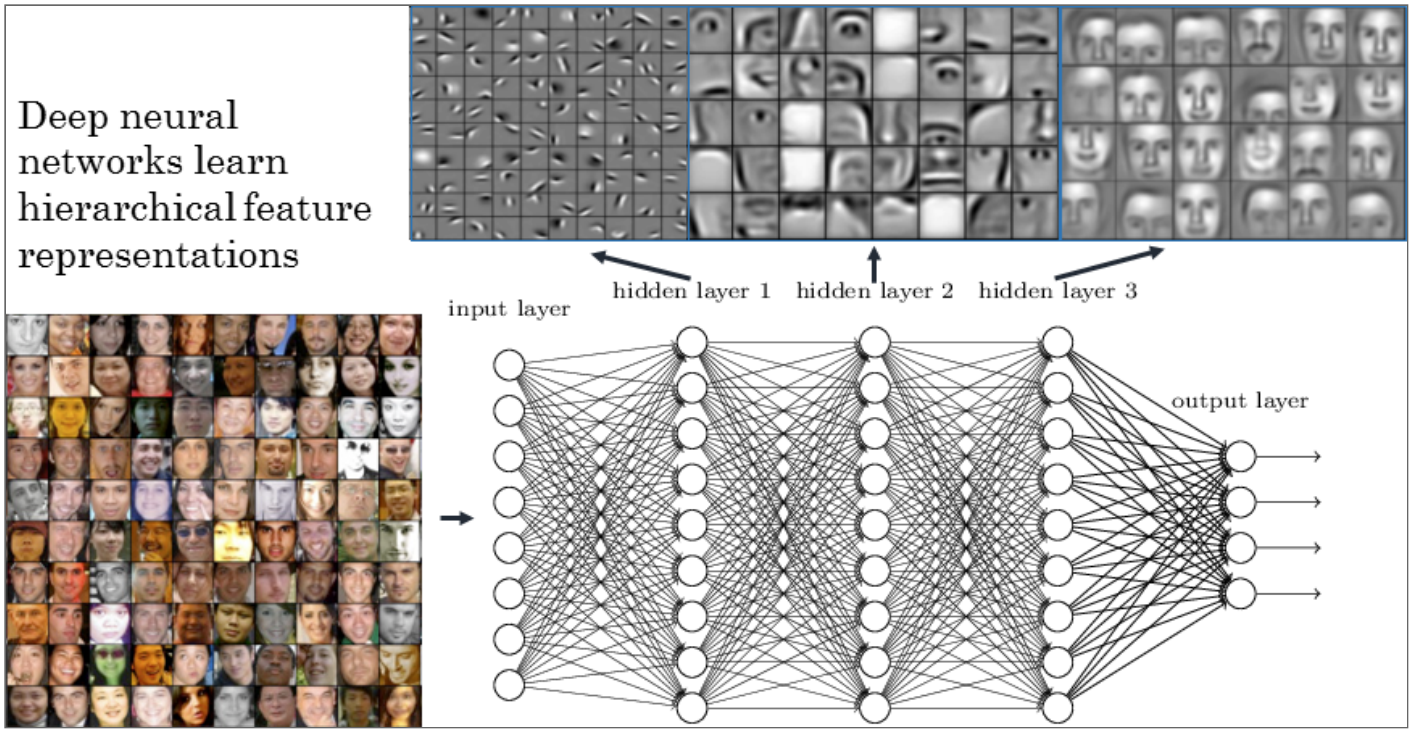
- Use for: Classification (or Regression)
- Pros:
    - combines benefits of multiple learning types
    - easy to implement
    - tends to win competitions
- Cons:
    - difficult to interpret
    - training/prediction time depends on component models



From https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

Neural Networks (aka Deep Learning)

- Pros and Cons of Deep Learning
    - sensitive to initialization and structure
    - high complexity -> needs more data
    - low interpretability
    - can learn complex interactions
    - performs well on tasks involving complex signals (ex images, sound, etc)


Deep neural networks learn hierarchical feature representations
hidden layer 1
hidden layer 2
hidden layer 3
input layer
output layer

Questions?