Elements Of Data Science - F2025

Week 6: Intro to Machine Learning Models Continued

10/21/2025

TODOs

- Readings:
  - PDSH 05.03 Hyperparameters and Model Validation
  - Recommended: PML Chapter 6 (Except for Pipelines) and sklearn - model selection
  - Reference: PML Chapter Chap 3, 7, and sklearn - supervised learning

Today

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

Questions?

Environment Setup

Environment Setup

In [1]:

```python
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,LogisticRegression

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

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,LogisticRegression

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

In [2]:

```
def my_plot_decision_regions(X,y,model,figsize=(5,5),ax=None):
    '''Plot classifier decision regions, classification predictions and training data'''
    if not ax:
        fig,ax = plt.subplots(1,1,figsize=figsize)
    # use mlxtend plot_decision_regions
    model = model.fit(X.values,y.values)
    plot_decision_regions(X.values,y.values,model,ax=ax)
    ax.set_xlabel(X.columns[0]); ax.set_ylabel(X.columns[1]);

def my_plot_regression(X,y,model,label='yhat',figsize=(5,5),ax=None):
    '''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)
    model = model.fit(X.values,y.values)
    y_hat = model.predict(X_test)
    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 (Review)

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

Wine as Binary Classification

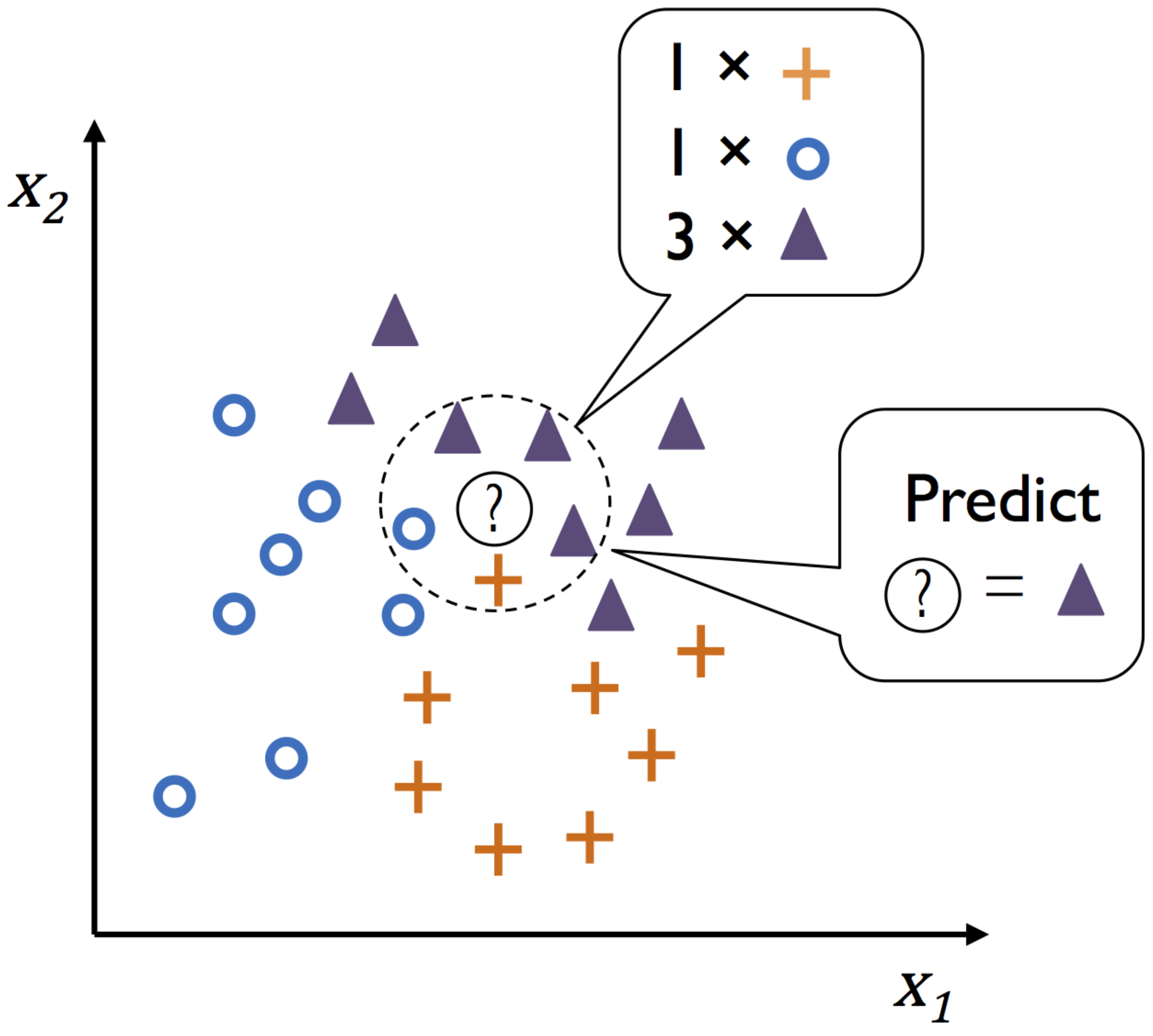
Wine as Binary Classification

In [3]:
```
df_wine = pd.read_csv('../data/wine_dataset.csv',usecols=['alcohol','ash','proline','hue','class'])
# rename 'class' as 'target', since class is a reserved python word
df_wine = df_wine.rename({'class':'target'},axis=1)

df_wine_2c = df_wine[df_wine.target < 2] # only keep classes 0 and 1

X_2c = df_wine_2c[['proline','hue']]
y_2c = df_wine_2c['target']

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

X_2c_zscore = X_2c.apply(zscore,axis=0)
alcohol_2c_zscore = zscore(df_wine_2c.alcohol)

y_2c.value_counts().sort_index()
```

Out[3]:
```
0    59
1    71
Name: target, dtype: int64
```

Distance Based: k-Nearest Neighbor (kNN)

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



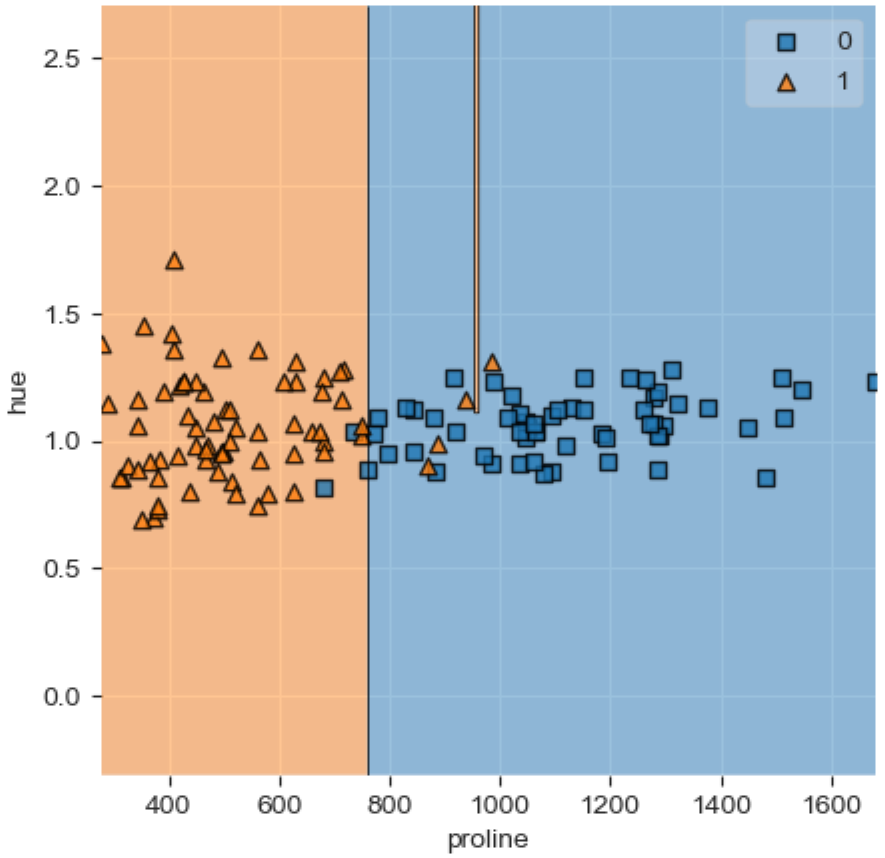
From PML

KNN in sklearn

KNN in sklearn

In [4]:

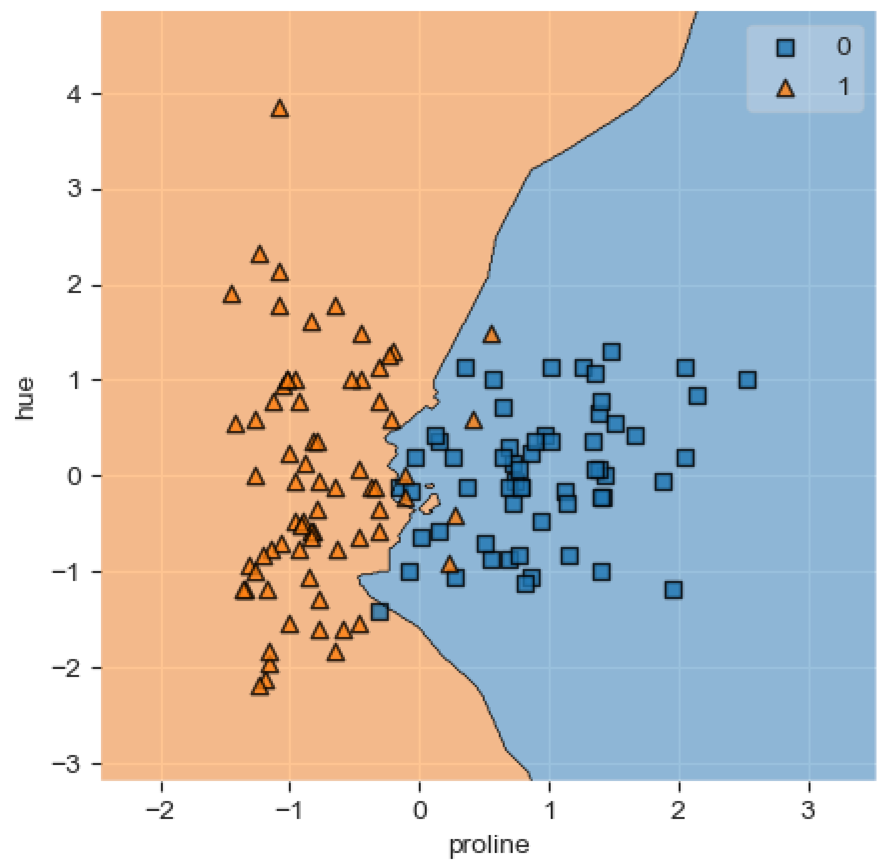
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_2c,y_2c)

my_plot_decision_regions(X_2c,y_2c,knn)
```



Effects of Standardization on Distance Based Methods

Effects of Standardization on Distance Based Methods

In [5]:
```
knn_z = KNeighborsClassifier(n_neighbors=3)
knn_z.fit(X_2c_zscore,y_2c)

my_plot_decision_regions(X_2c_zscore,y_2c,knn_z)
```

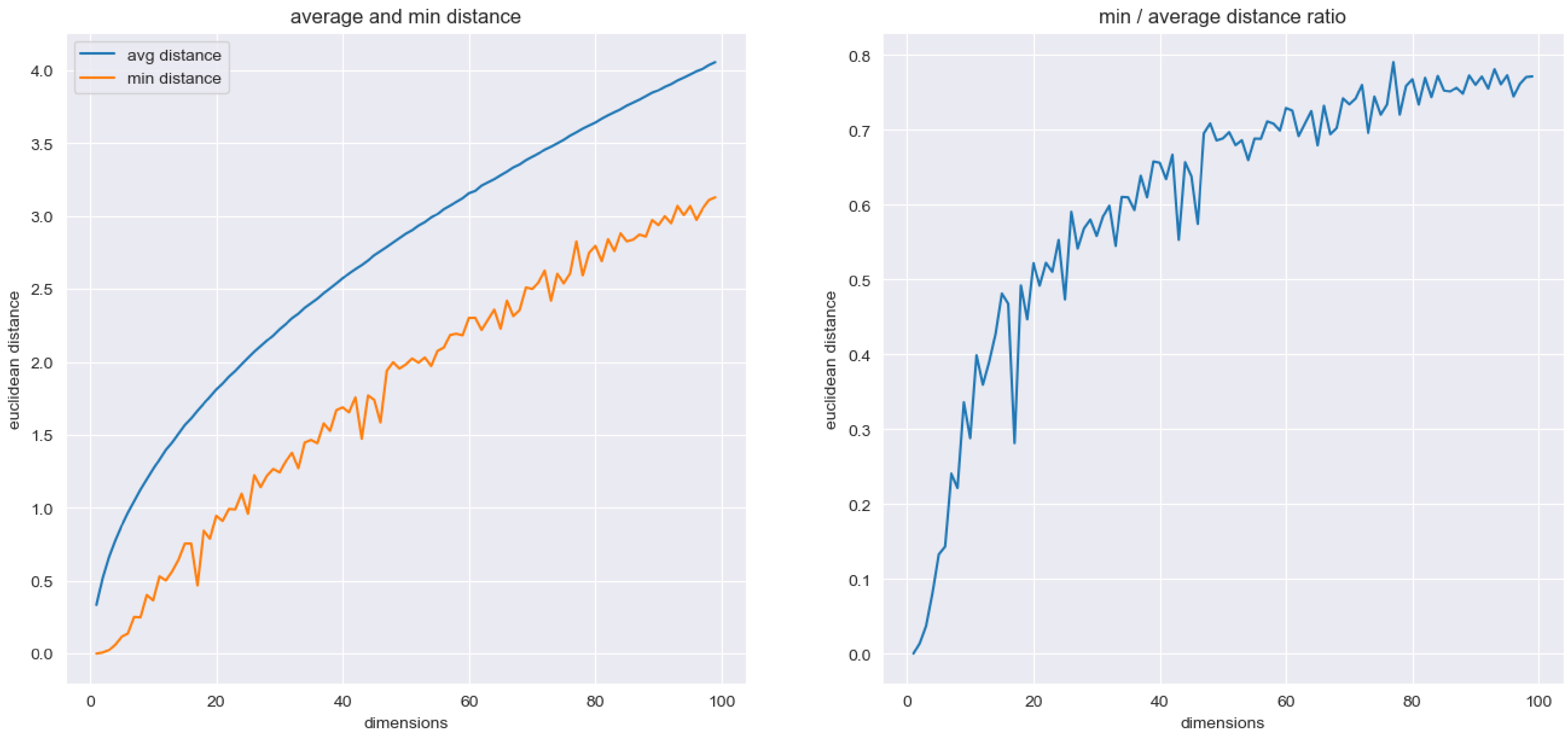


Curse of Dimensionality Cont.

Curse of Dimensionality Cont.

In [7]:

```
fig,ax = plt.subplots(1,2,figsize=(16,7))
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('average and min distance'); ax[0].set_xlabel('dimensions'); ax[0].set_ylabel('euclidean distance');
ax[1].plot(dimensions,min_avg_ratio)
ax[1].set_title('min / average distance ratio'); ax[1].set_xlabel('dimensions'); ax[1].set_ylabel('euclidean distance');
```



Regression with kNN

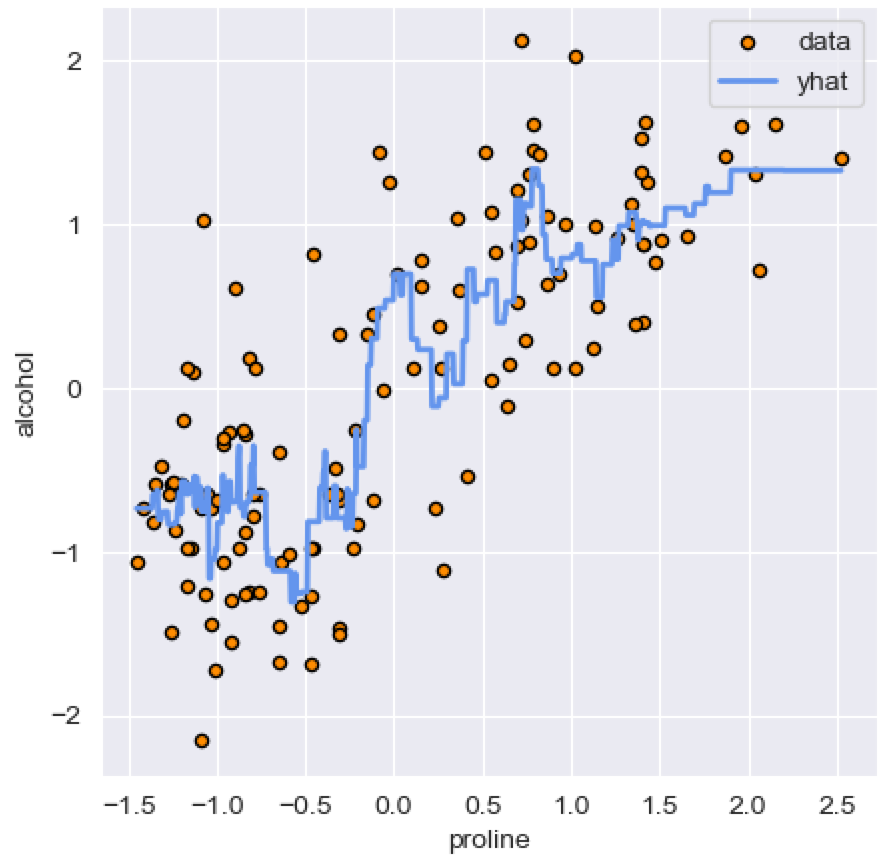
Approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighbourhood.

Regression with kNN

Approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighbourhood.

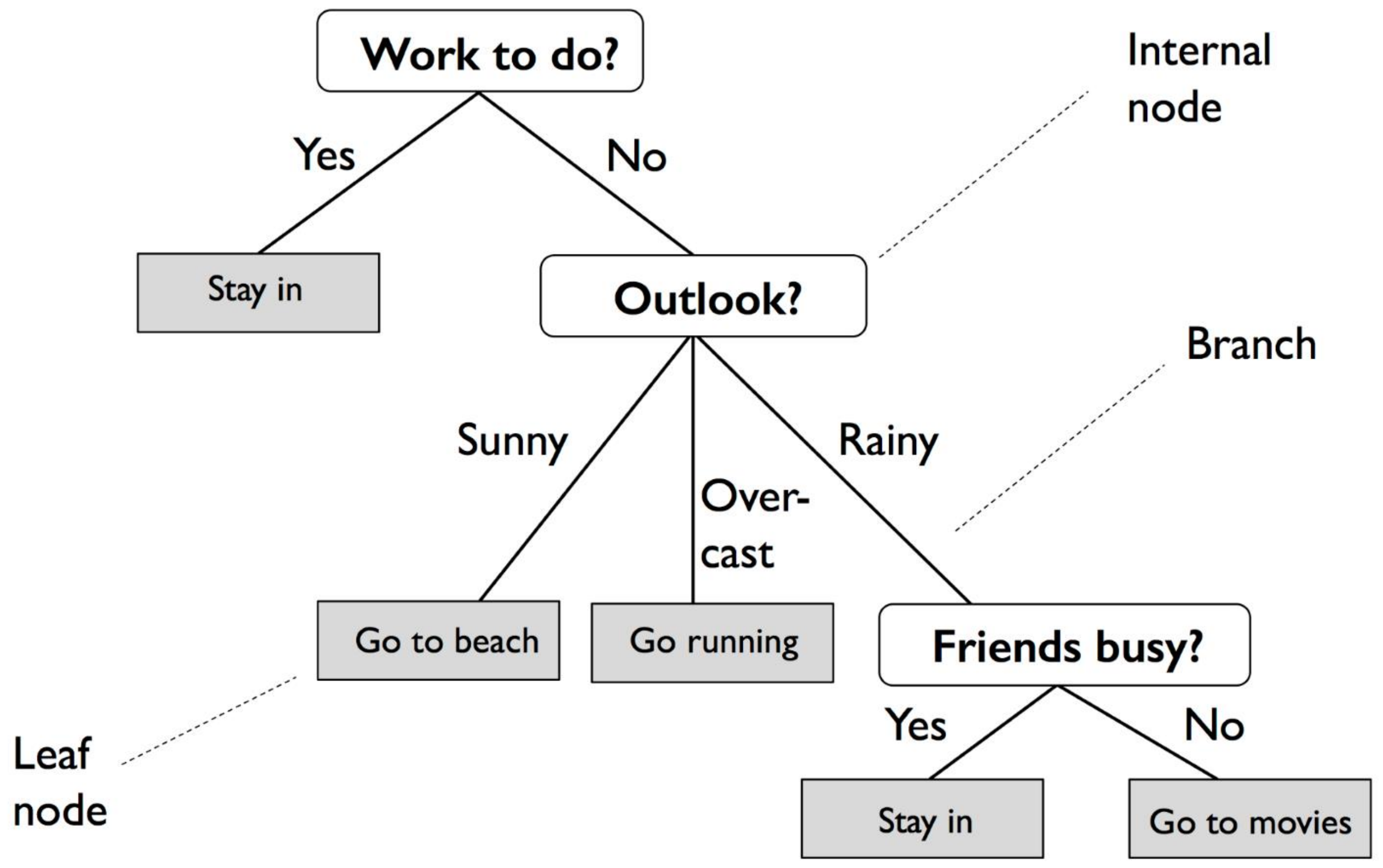
In [32]:

```
from sklearn.neighbors import KNeighborsRegressor

knnr = KNeighborsRegressor(n_neighbors=5)
knnr.fit(X_2c_zscore[['proline']],alcohol_2c_zscore)

my_plot_regression(X_2c_zscore[['proline']],alcohol_2c_zscore,knnr)
```



Decision Tree

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



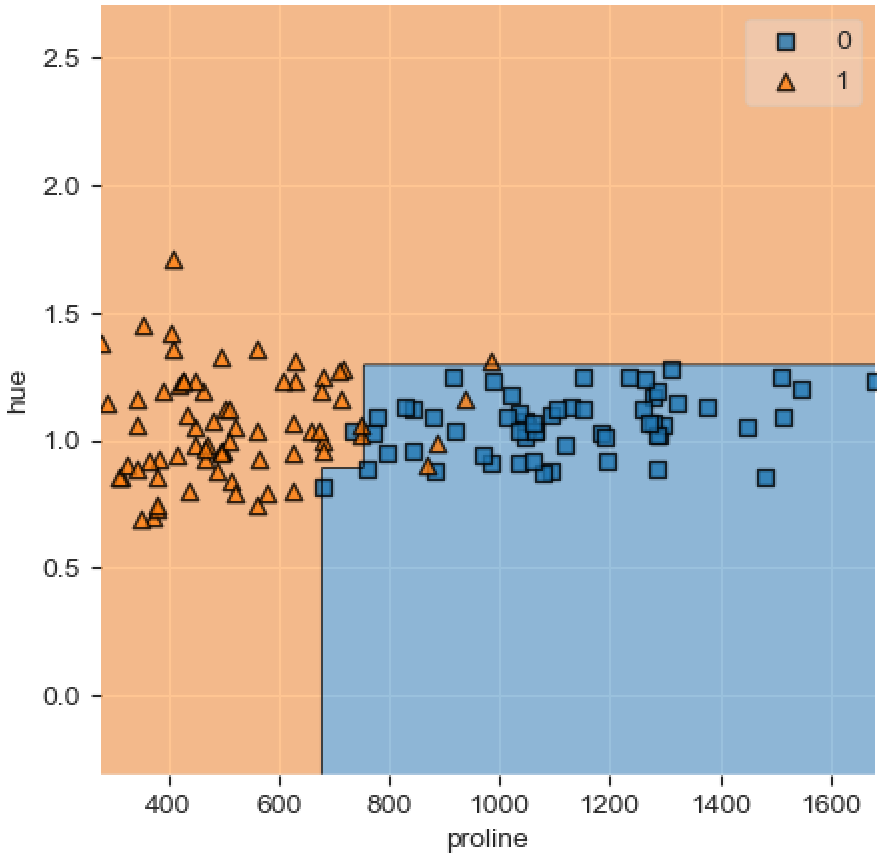
From PML

Decision Tree Classifier in sklearn

Decision Tree Classifier in sklearn

In [10]:

```
from sklearn.tree import DecisionTreeClassifier

dtc_md3 = DecisionTreeClassifier(max_depth=3) # max_depth: max number of questions
dtc_md3.fit(X_2c,y_2c)

my_plot_decision_regions(X_2c,y_2c,dtc_md3)
```

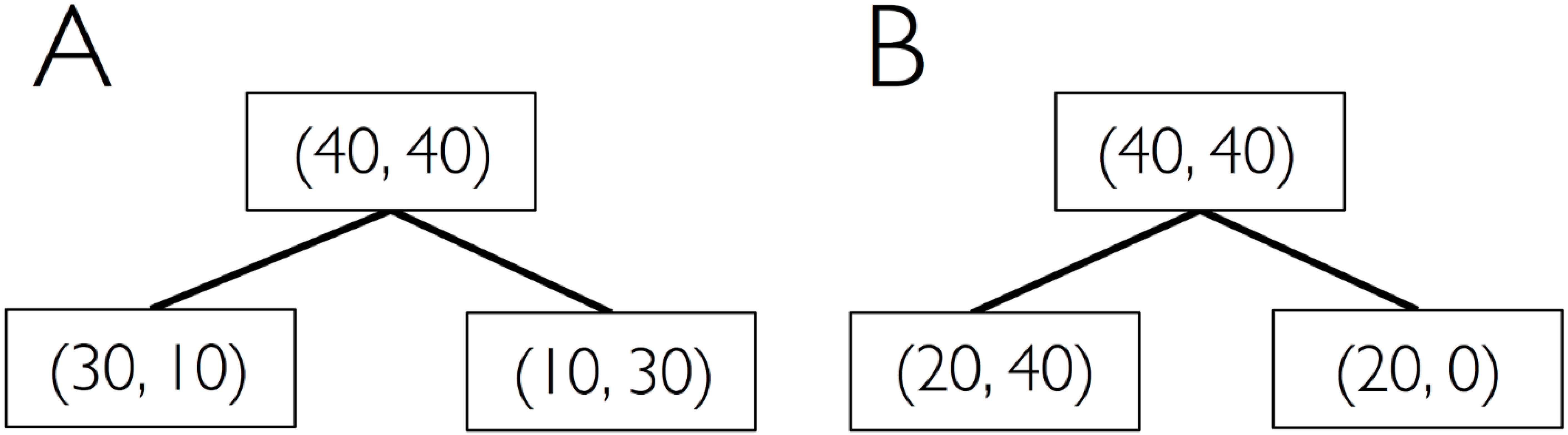


Building a Decision Tree

- How to decide which question to choose (eg. Should I choose question A or B)?
- **Reduce Impurity**

Building a Decision Tree

- How to decide which question to choose (eg. Should I choose question A or B)?
- **Reduce Impurity**



From PML

- Information Gain: Tie, *Gini*: B, Entropy: B

Plot Learned Decision Tree Using sklearn

In [ ]:

```

```

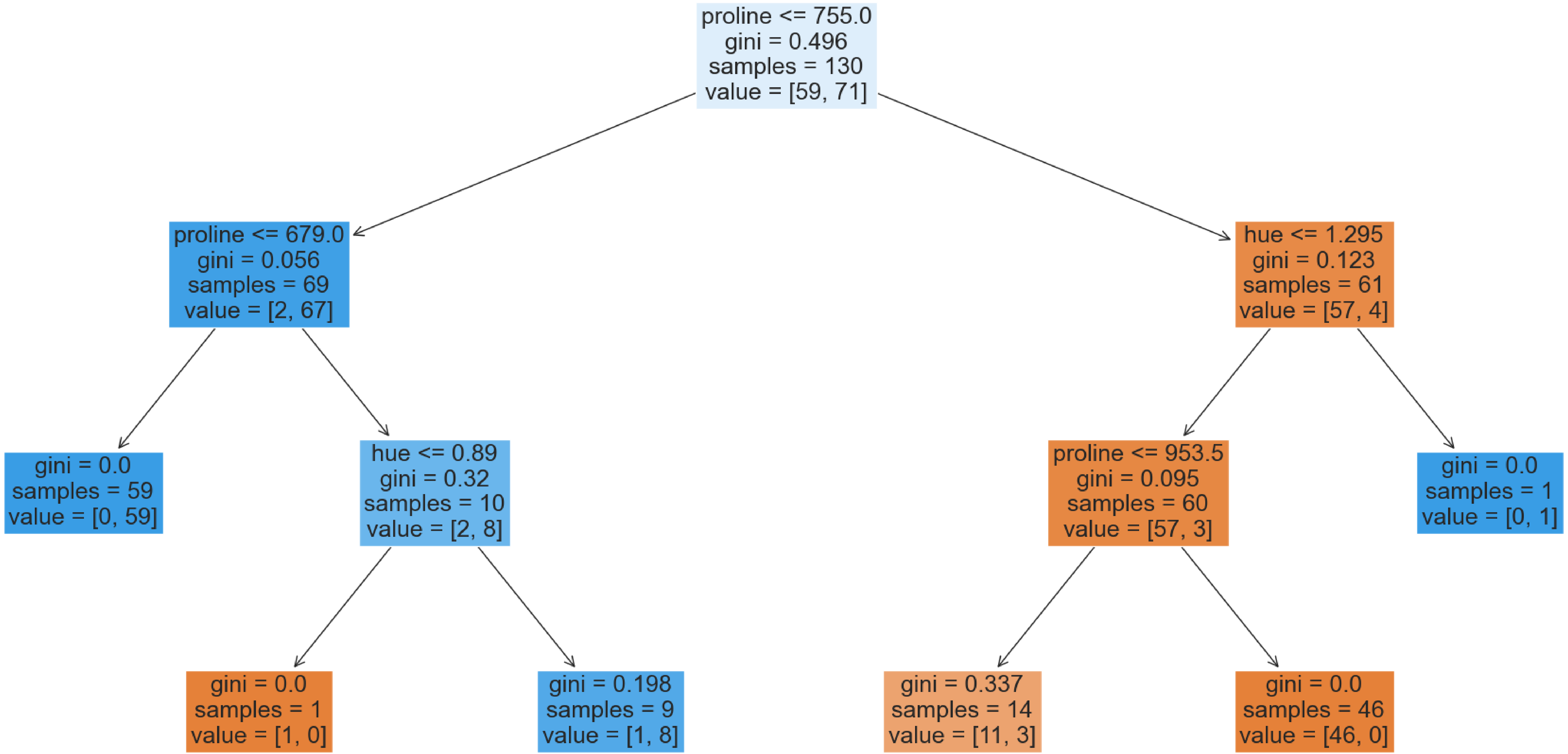
Plot Learned Decision Tree Using sklearn

In [ ]:

```
```

In [12]:

```
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_2c.columns,filled=True);
```

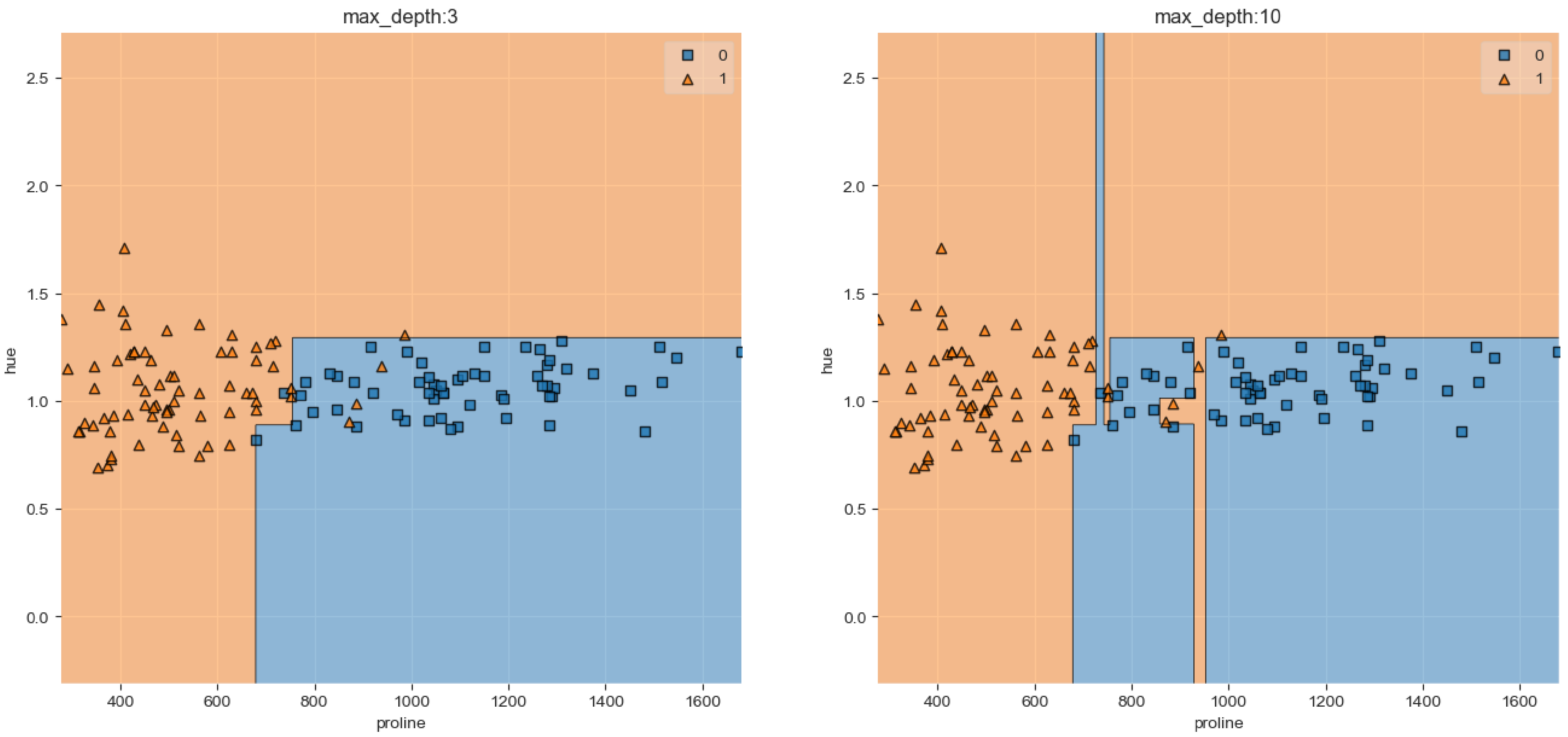


Decision Tree: Increase Maximum Depth

Decision Tree: Increase Maximum Depth

In [13]:

```
dtc_md10 = DecisionTreeClassifier(max_depth=10)
dtc_md10.fit(X_2c,y_2c)

fig,ax = plt.subplots(1,2,figsize=(16,7))
my_plot_decision_regions(X_2c, y_2c, model=dtc_md3, ax=ax[0]);
my_plot_decision_regions(X_2c, y_2c, model=dtc_md10, ax=ax[1]);
ax[0].set_title('max_depth:3');ax[1].set_title('max_depth:10');
```



Plot Learned Decision Tree Using sklearn

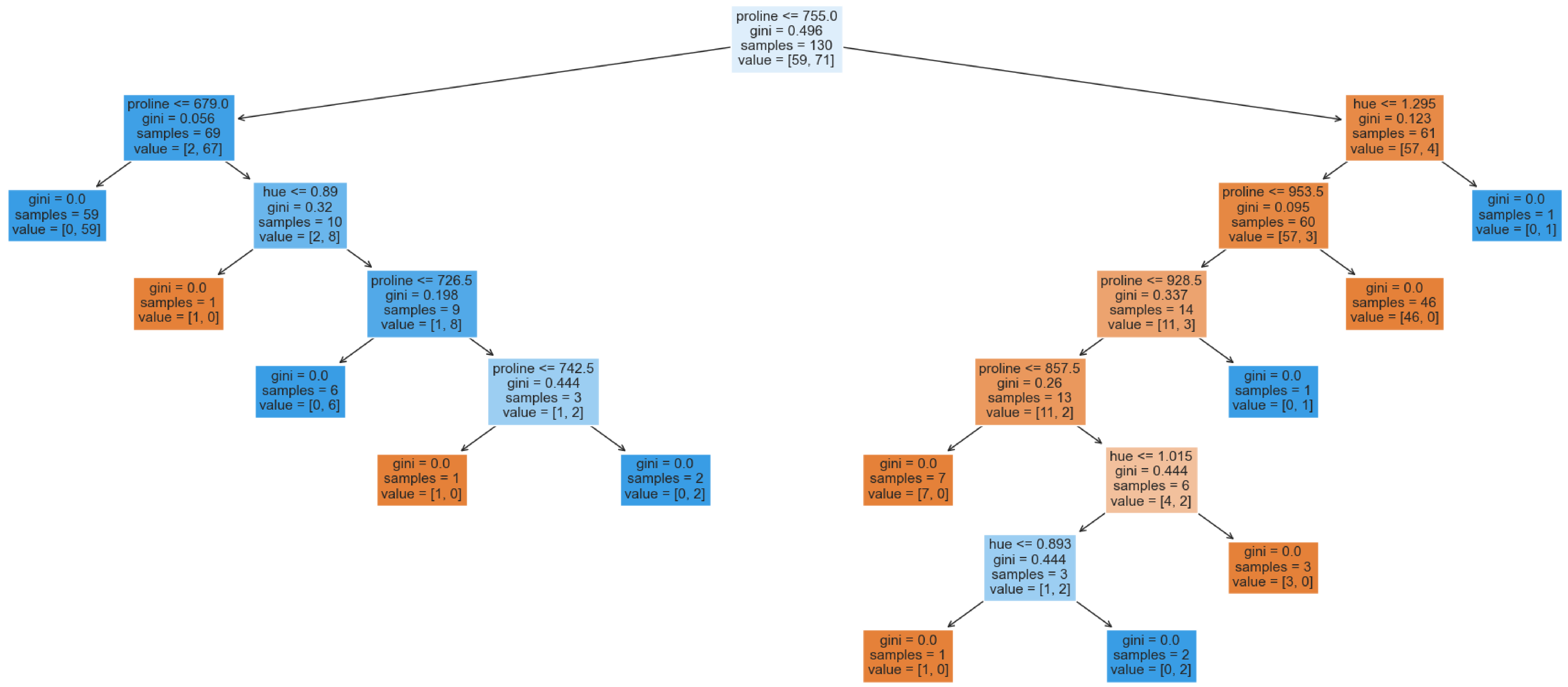
- For tree with max_depth=10

Plot Learned Decision Tree Using sklearn

- For tree with max_depth=10

In [14]:

```
fig,ax = plt.subplots(1,1,figsize=(24,10))
plot_tree(dtc_md10,ax=ax,fontsize=11,feature_names=X_2c.columns,filled=True);
```

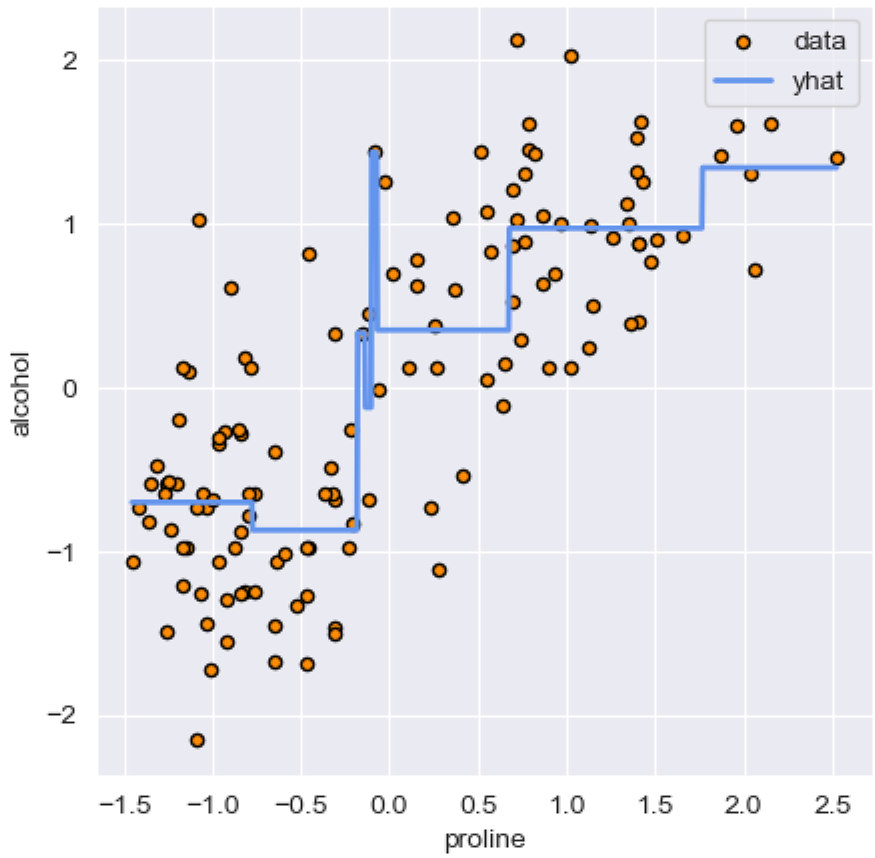


Regression with Decision Trees

Regression with Decision Trees

In [15]:

```
from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor(max_depth=3)
dtr.fit(X_2c_zscore[['proline']],alcohol_2c_zscore)

my_plot_regression(X_2c_zscore[['proline']],alcohol_2c_zscore,dtr)
```



Ensemble Methods

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

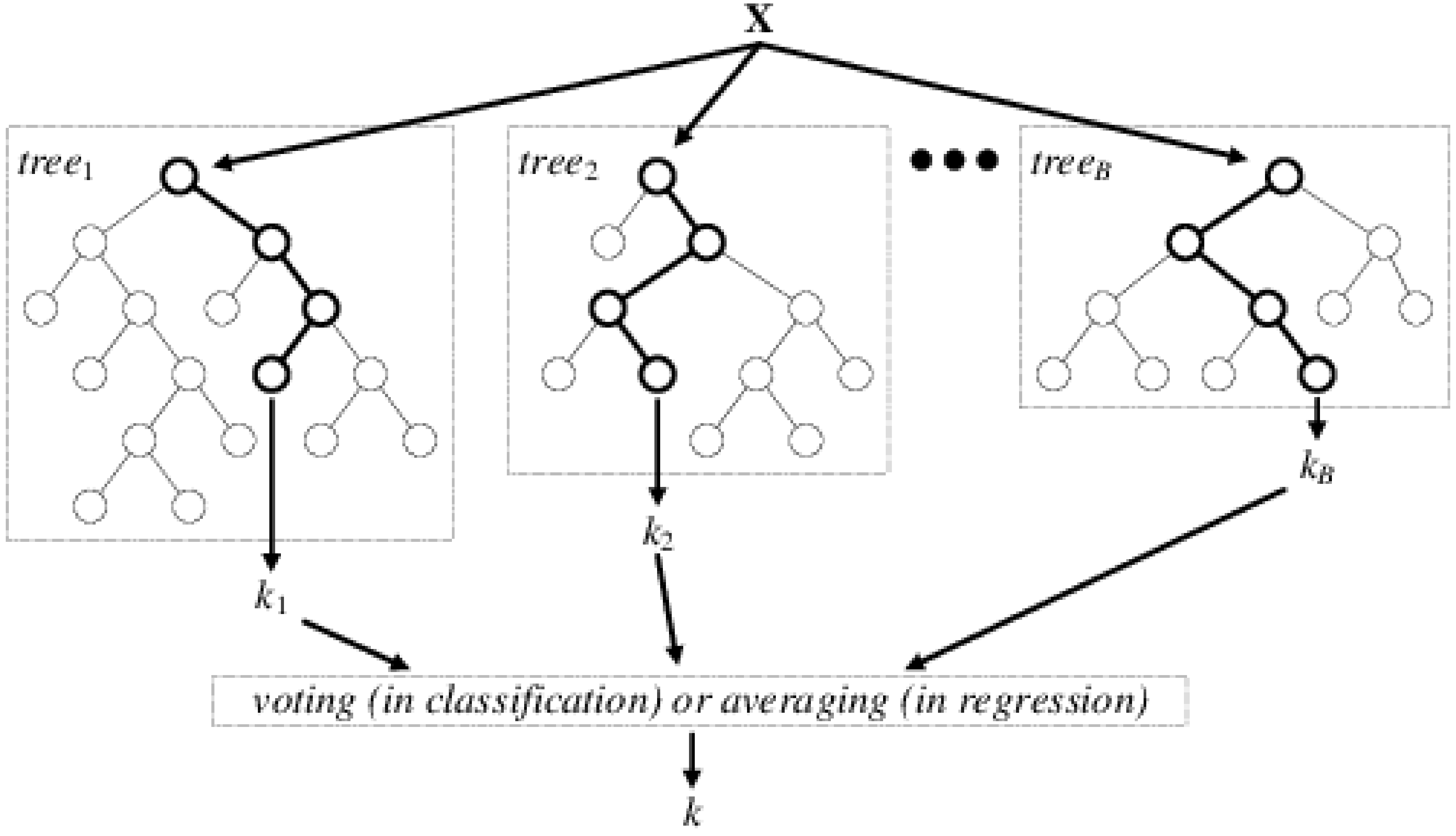
Ensemble Methods

- "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_Activation_Sequence_Profiling_and_Prediction_of_Shot_Effectiveness

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	…

C_1 C_2 C_m

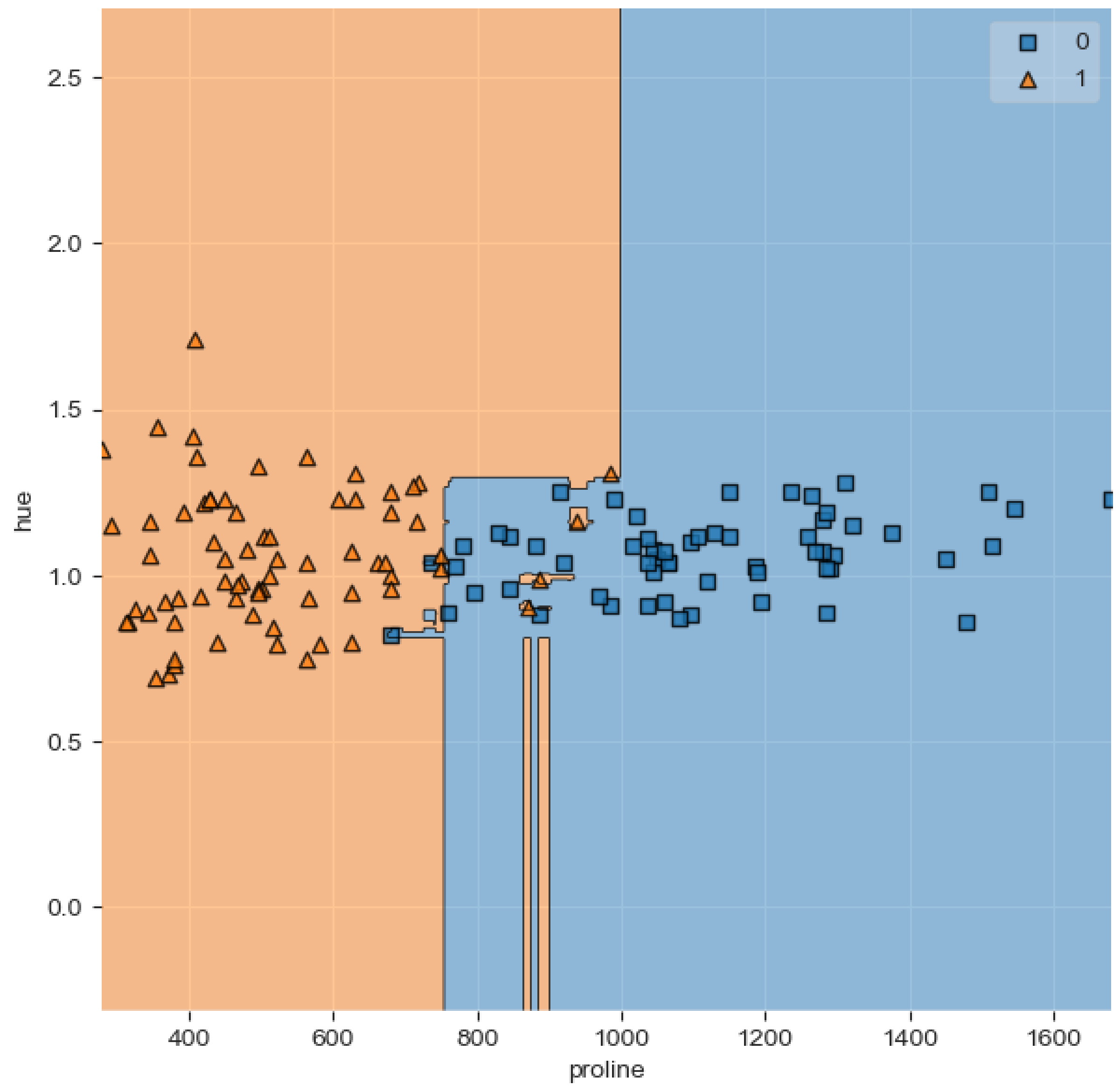
From PML

Random Forests with sklearn

Random Forests with sklearn

In [35]:

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=100, # number of trees in ensemble
                             max_depth=30,      # same as decision trees
                             n_jobs=-1,        # parallelize using all available cores, default: None=1
                             random_state=0)   # for demonstration only
rfc.fit(X_2c,y_2c)
my_plot_decision_regions(X_2c,y_2c,rfc,figsize=(7,7))
```

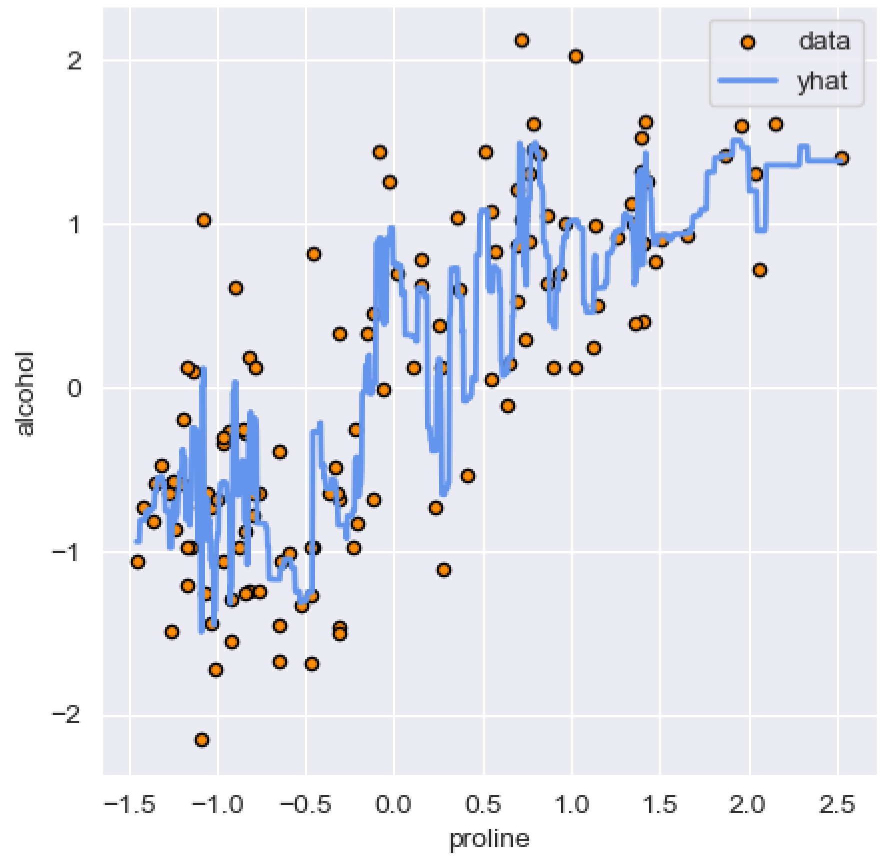


Regression with RandomForests

Regression with RandomForests

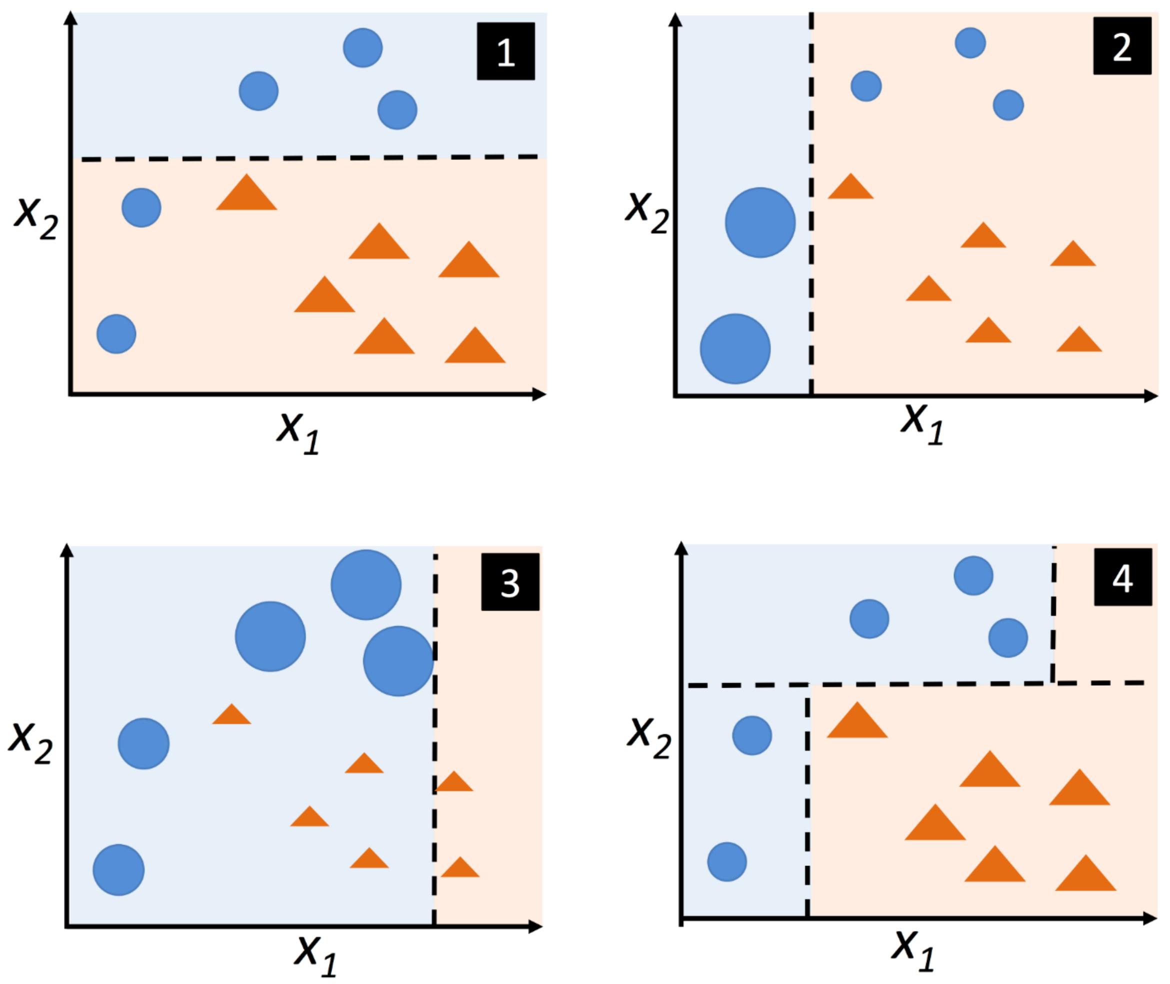
In [39]:

```
from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(n_estimators=1000, max_depth= 20 , n_jobs=-1)
rfr.fit(df_wine[['proline']],df_wine.alcohol)

my_plot_regression(X_2c_zscore[['proline']],alcohol_2c_zscore,rfr)
```



Gradient Boosted Trees

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



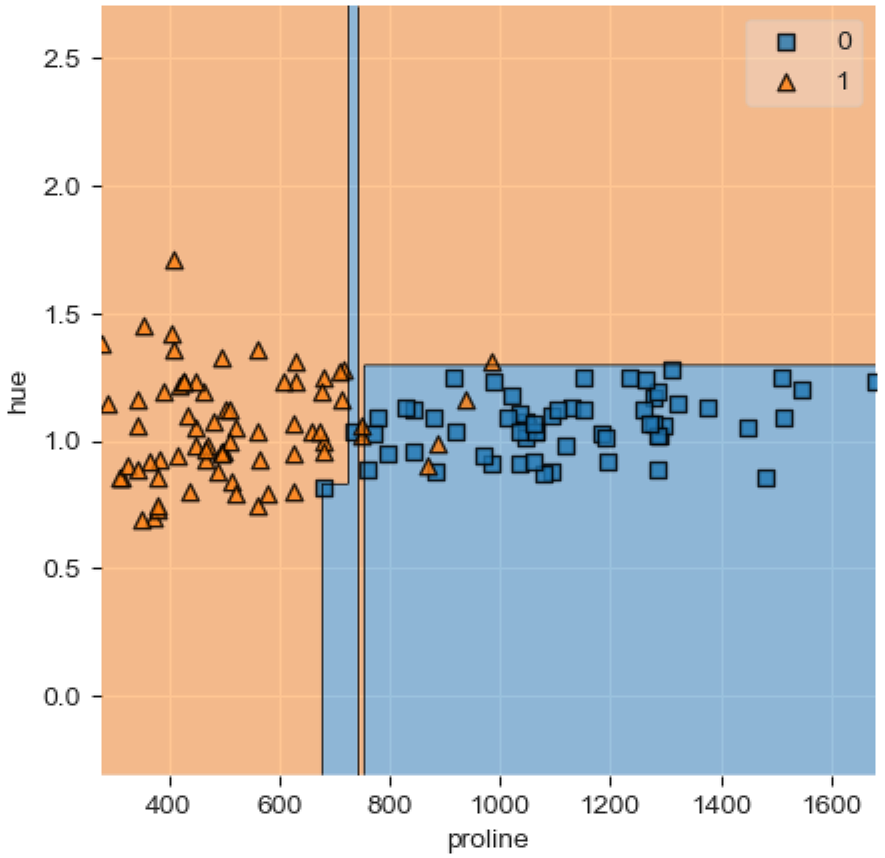
From PML

Gradient Boosted Trees in sklearn

Gradient Boosted Trees in sklearn

In [18]:

```python
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=10)
gbc.fit(X_2c,y_2c)

my_plot_decision_regions(X_2c,y_2c,gbc)
```

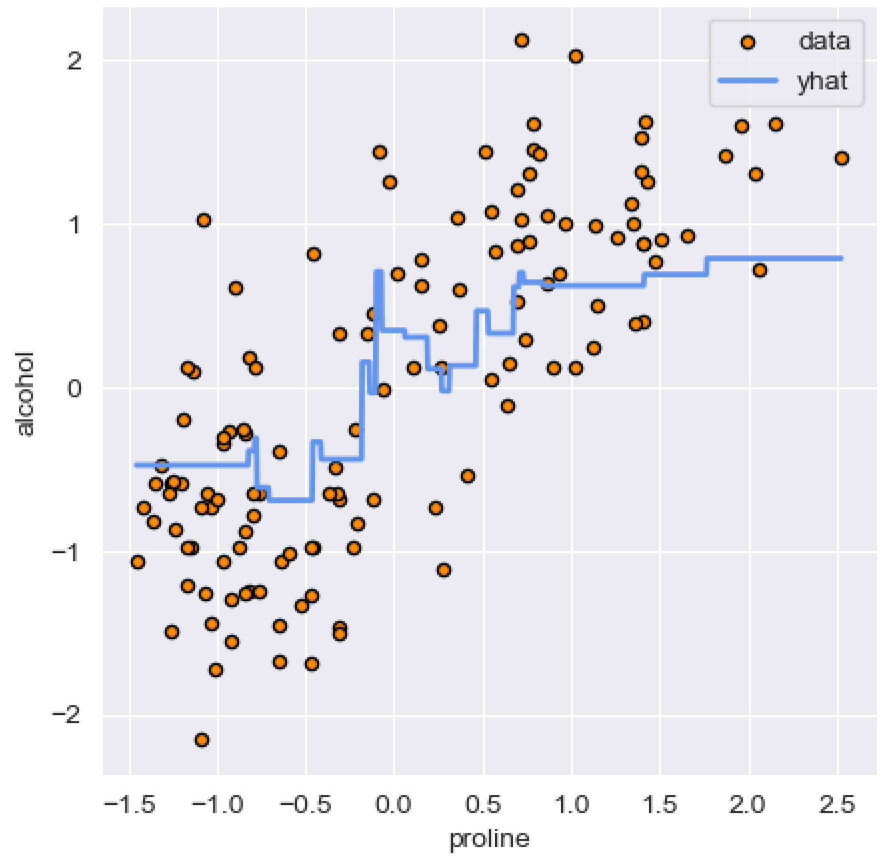


Regression with Gradient Boosted Trees

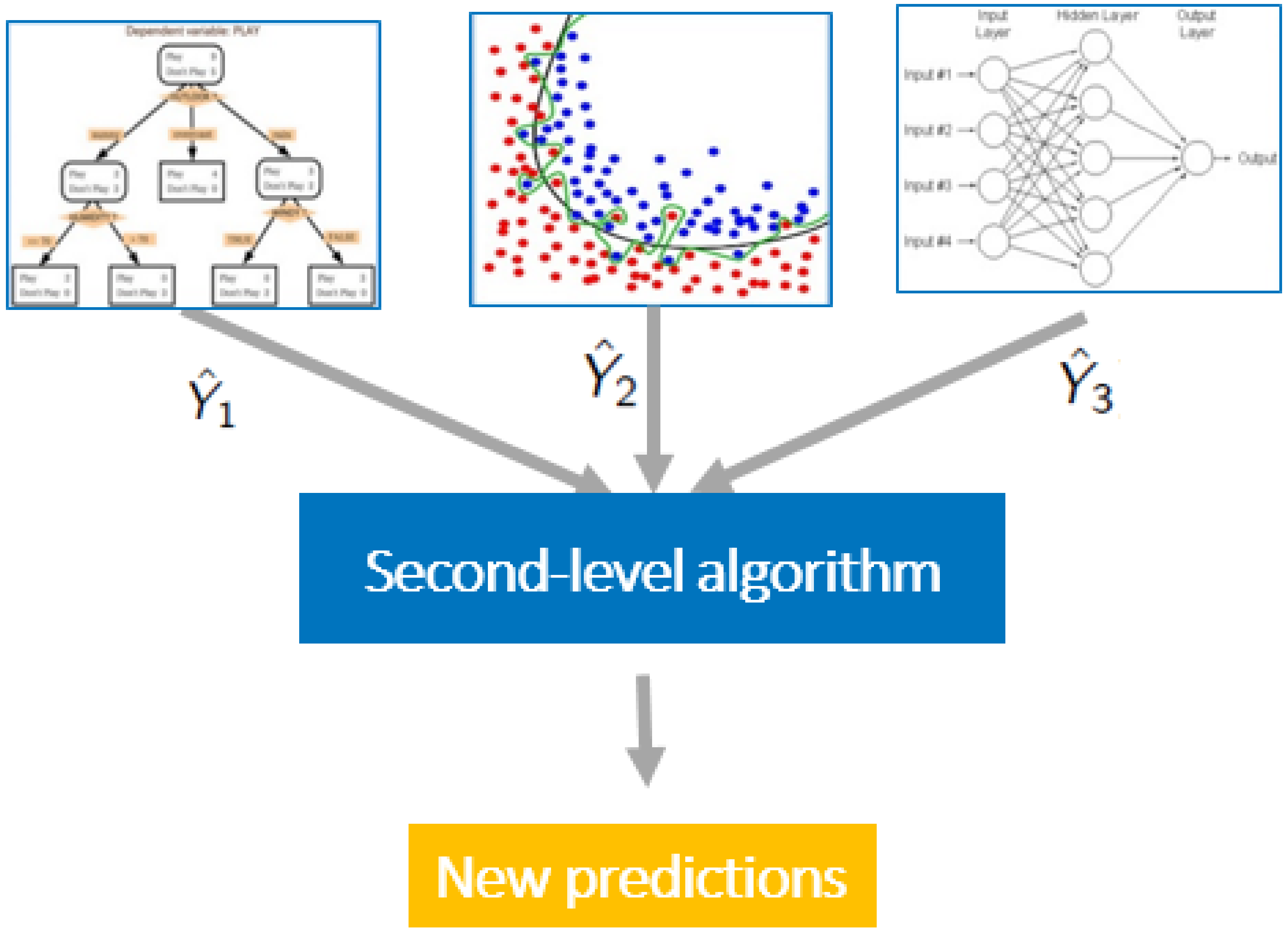
Regression with Gradient Boosted Trees

In [19]:

```
from sklearn.ensemble import GradientBoostingRegressor

gbr = GradientBoostingRegressor(n_estimators=10)
gbr.fit(X_2c_zscore[['proline']],alcohol_2c_zscore)

my_plot_regression(X_2c_zscore[['proline']],alcohol_2c_zscore,gbr)
```



Stacking



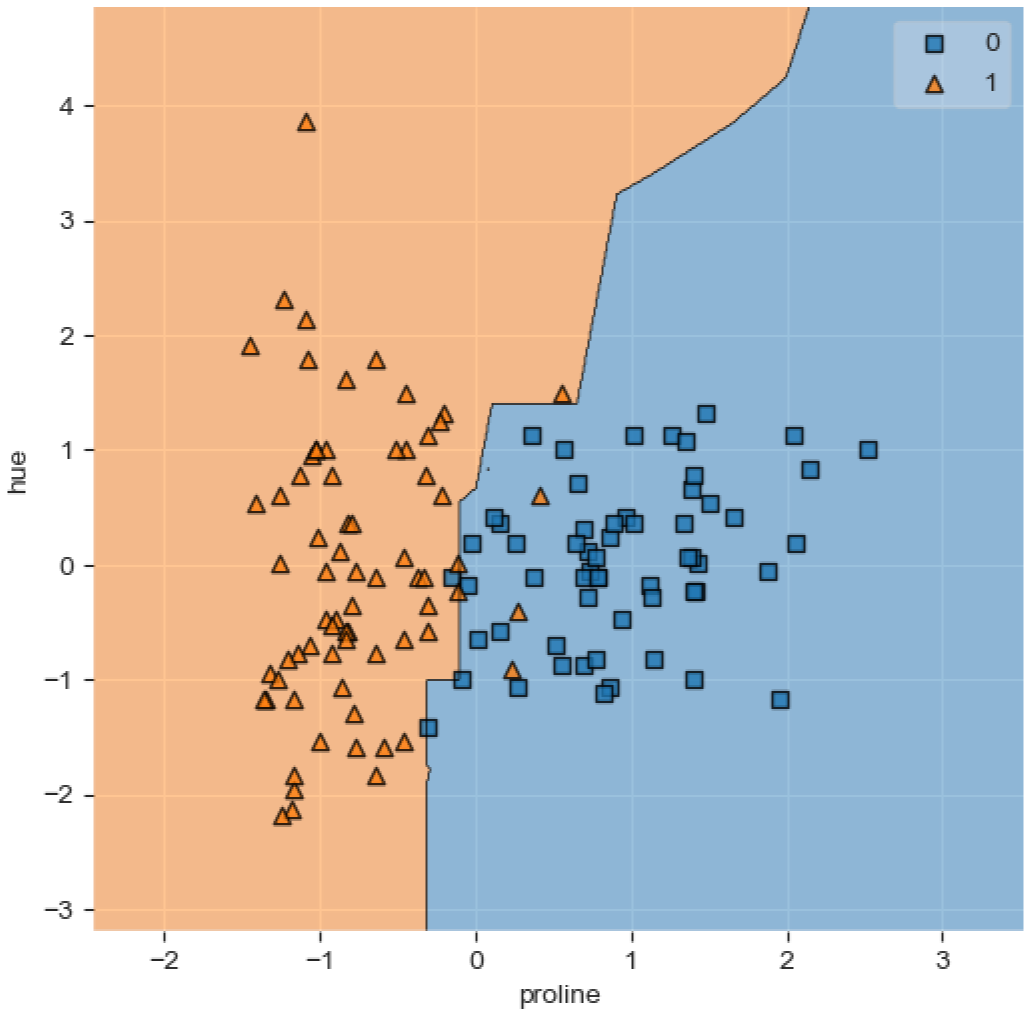
From https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

Stacking for Classification

Stacking for Classification

In [20]:

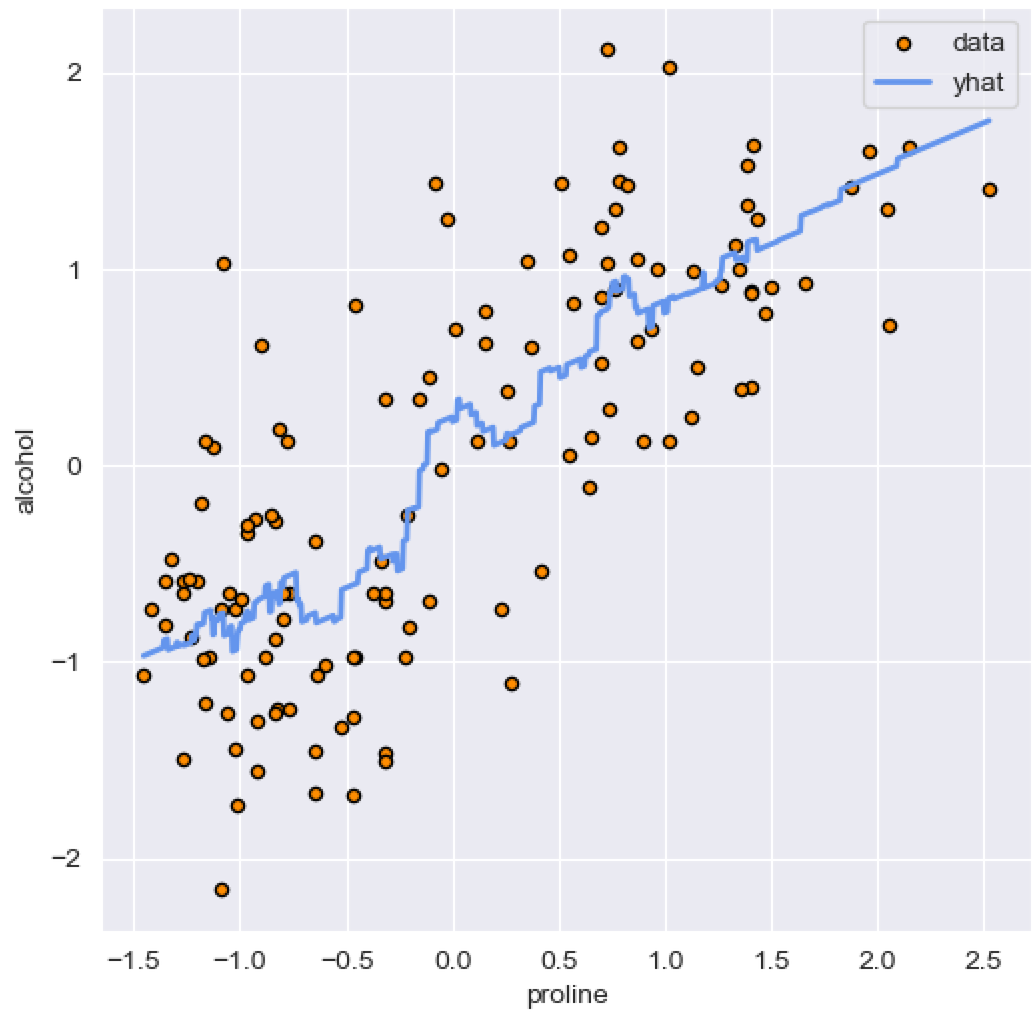
```
from sklearn.ensemble import StackingClassifier

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_2c_zscore,y_2c)

my_plot_decision_regions(X_2c_zscore,y_2c,stackc,figsize=(6,6))
```



Stacking for Regression

Stacking for Regression

In [21]:
```python
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_2c_zscore[['proline']],alcohol_2c_zscore)

my_plot_regression(X_2c_zscore[['proline']],alcohol_2c_zscore,stackr,figsize=(6,6))
```



Wine as Multi-Class Classification

Wine as Multi-Class Classification

In [22]:

```
X_mc = df_wine[['proline','hue']]
y_mc = df_wine.target

X_mc_zscore = X_mc.apply(zscore,axis=0)
alcohol_mc_zscore = zscore(df_wine.alcohol)

y_mc.value_counts().sort_index()
```

Out[22]:

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

Multiclass and Multilabel

- **Multiclass Classification** : more than two categories/classes
  - red/green/blue, flower type, integer 0-10
- **Multilabel Classification** : can assign more than one category to an instance
  - paper topics, entities in image

- **Multiclass-Multilabel/Multitask Classification** : >1 one property with >2 one categories
  - type of fruit AND color of fruit
- **Multioutput Regression** : more than one numeric targets
  - temperature AND humidity

See sklearn docs (https://scikit-learn.org/stable/modules/multiclass.html#)

Sklearn Inherently Multiclass

- `LogisticRegression(multi_class='multinomial')`
- `KNeighborsClassifier`
- `DecisionTreeClassifier`
- `RandomForestClassifier`

Sklearn Inherently Multiclass

- `LogisticRegression(multi_class='multinomial')`
- `KNeighborsClassifier`
- `DecisionTreeClassifier`
- `RandomForestClassifier`

In [50]:

```
dt_mc = DecisionTreeClassifier().fit(X_mc_zscore,y_mc) # fit on multiclass

# generate 3 predictions
y_hats = dt_mc.predict(X_mc_zscore.iloc[[82,15,166]])

# display target and prediction
pd.DataFrame({'y':y_mc.iloc[[82,15,166]],'y_hat':y_hats})
```

Out[50]:

	y	y_hat
82	1	1
15	0	0
166	2	2

One Vs. Rest (OvR) Classification For Multiclass

One Vs. Rest (OvR) Classification For Multiclass

- What about other models (eg Perceptron)?

- Can use any binary classifier for Multiclass 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)

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

Sklearn OvR for Multiclass

- `LogisticRegression(multi_class="ovr")`
- `GradientBoostingClassifier`
- `Perceptron`

OvR For Logistic Regression

OvR For Logistic Regression

In [24]:

```
logr_mc = LogisticRegression(multi_class='ovr', # default
                             max_iter=1000)     # to avoid timeout errors
logr_mc.fit(X_mc_zscore,y_mc)
y_hats = logr_mc.predict(X_mc_zscore.iloc[[82,15,166]]) # generate 3 predictions
y_prob = logr_mc.predict_proba(X_mc_zscore.iloc[[82,15,166]])
pd.DataFrame({'y_hat':y_hats,'p_c1':y_prob[:,0],'p_c2':y_prob[:,1],'p_c3':y_prob[:,2]})\
 .style.background_gradient(subset=['p_c1','p_c2','p_c3']).format('{:.2f}',subset=['p_c1','p_c2','p_c3'])
```

Out[24]:

	y_hat	p_c1	p_c2	p_c3
0	1	0.15	0.85	0.00
1	0	0.97	0.03	0.00
2	2	0.18	0.34	0.48

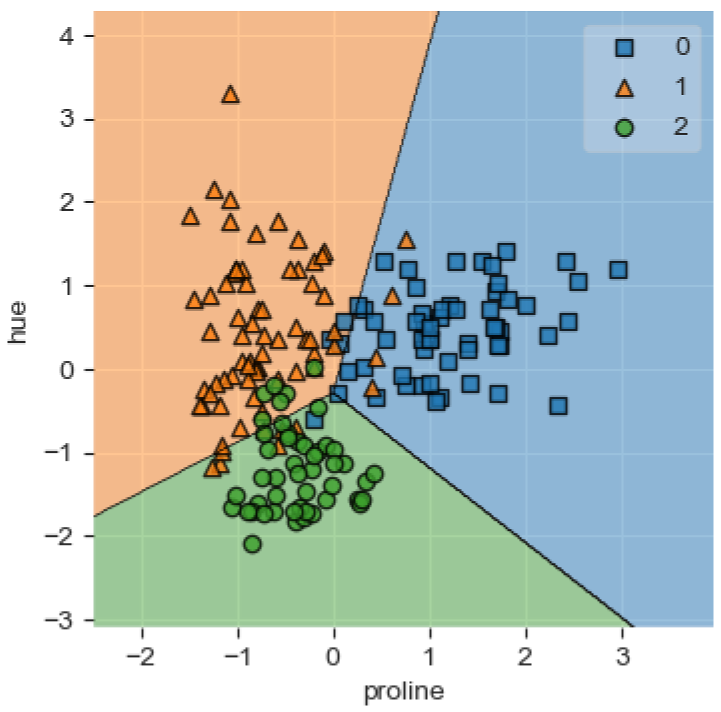
OvR For Logistic Regression

In [24]:
```
logr_mc = LogisticRegression(multi_class='ovr', # default
                             max_iter=1000)     # to avoid timeout errors
logr_mc.fit(X_mc_zscore,y_mc)
y_hats = logr_mc.predict(X_mc_zscore.iloc[[82,15,166]]) # generate 3 predictions
y_prob = logr_mc.predict_proba(X_mc_zscore.iloc[[82,15,166]])
pd.DataFrame({'y_hat':y_hats,'p_c1':y_prob[:,0],'p_c2':y_prob[:,1],'p_c3':y_prob[:,2]})\
 .style.background_gradient(subset=['p_c1','p_c2','p_c3']).format('{:.2f}',subset=['p_c1','p_c2','p_c3'])
```

Out[24]:

	y_hat	p_c1	p_c2	p_c3
0	1	0.15	0.85	0.00
1	0	0.97	0.03	0.00
2	2	0.18	0.34	0.48

In [25]:
```
my_plot_decision_regions(X_mc_zscore,y_mc,logr_mc,figsize=(4,4))
```



One vs. One Classification

- Train one classifier for each pair-wise comparison of classes

- `SVC`

Inherently Multilabel (aka Multioutput)

- `KNeighborsClassifier`
- `DecisionTreeClassifier`
- `MLPClassifier`
- `RandomForestClassifier`

Inherently Multilabel (aka Multioutput)

- `KNeighborsClassifier`
- `DecisionTreeClassifier`
- `MLPClassifier`
- `RandomForestClassifier`

In [26]:
```
X_ml = X_mc[['proline']].iloc[-3:] # get 3 samples
y_ml = np.array([[1, 0, 1], [0, 1, 1], [0, 0, 0]]) # generate 3 sets of random multi-labels for demonstration

dt_ml = DecisionTreeClassifier(max_depth=1).fit(X_ml,y_ml) # fit multilabel
dt_ml.predict(X_ml)
```

Out[26]:
```
array([[0, 0, 1],
       [0, 0, 1],
       [0, 0, 0]])
```

Sklearn `MultiOutputClassifier` meta-estimator

- fits one classifier per target (One vs. Rest)

Sklearn `MultiOutputClassifier` meta-estimator

- fits one classifier per target (One vs. Rest)

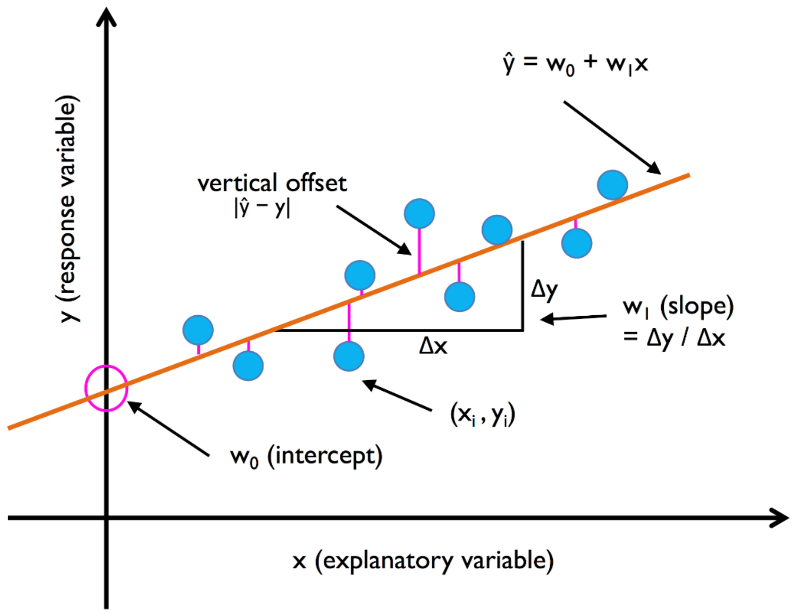
In [27]:
```python
from sklearn.multioutput import MultiOutputClassifier

# wrap your classifier with MultiOutputClassifier
mc_logr = MultiOutputClassifier(LogisticRegression())
mc_logr.fit(X_ml,y_ml)
mc_logr.predict(X_ml)
```

Out[27]:
```
array([[0, 0, 1],
       [1, 1, 1],
       [0, 0, 0]])
```

Review of Models

Model Review: Simple/Multiple Linear Regression

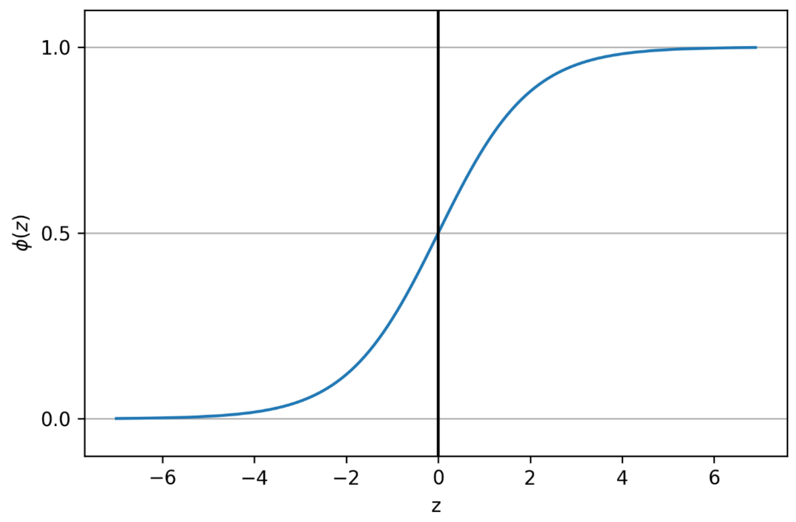
- Use for: Regression
- Pros:
    - fast to train
    - interpretable coefficients
- Cons:
    - assumes linear relationship
    - depends on removing colinear features



from PML

Model Review: Logistic Regression

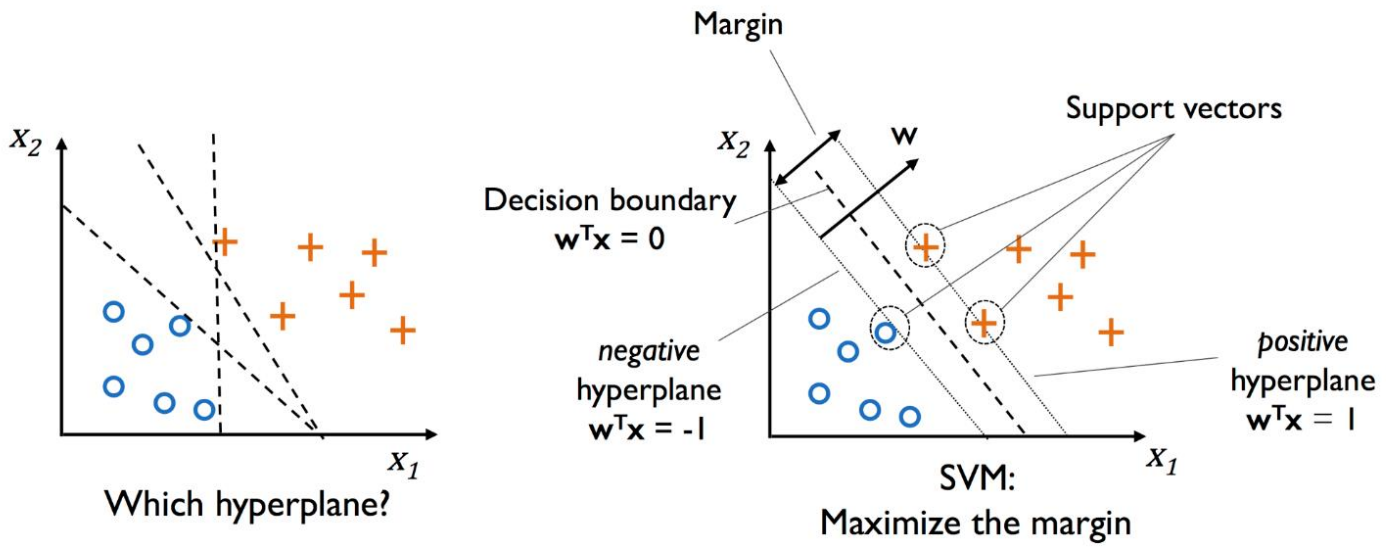
- Use for: Classification
- Pros:
  - fast to train
  - interpretable coefficients (log odds)
- Cons:
  - assumes linear boundary
  - depends on removing colinear features



from PML

Model Review: Support Vector Machine (SVM)

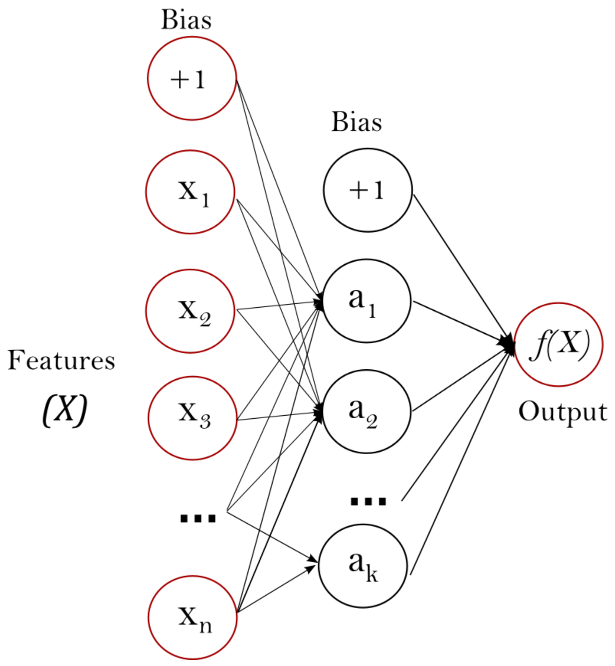
- Use for: Classification and Regression
- Pros:
  - fast to evaluate
  - can use kernel trick to learn non-linear functions
- Cons:
  - slow to train
  - can fail to converge on very large datasets



from PML

Model Review: Multi-Layer Perceptron

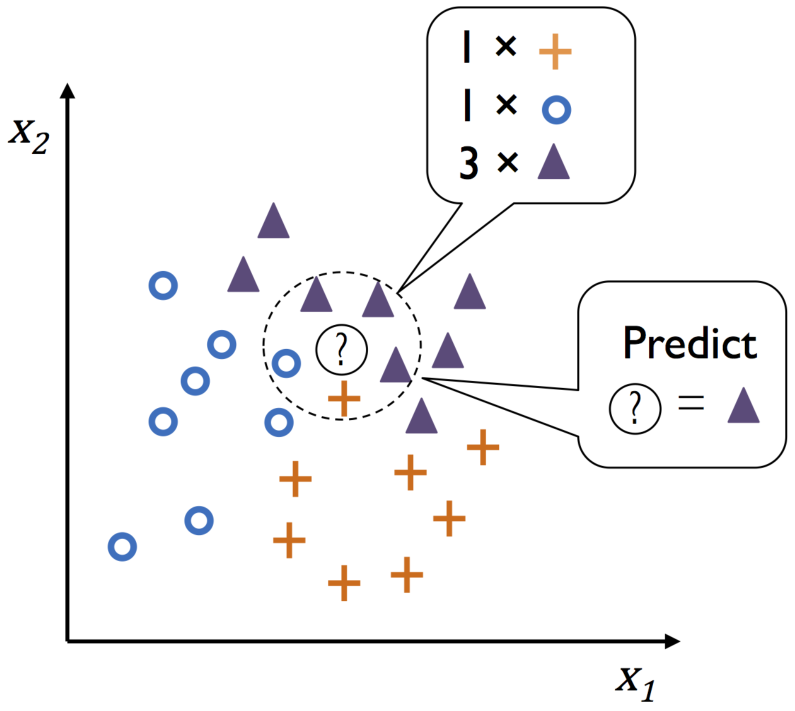
- Use for Classification or Regression
- Pros:
    - non-linear boundary
- Cons:
    - non-convex loss function (sensitive to initial weights)
    - sensitive to feature scaling
    - no GPU support in sklearn: use tensorflow or pytorch



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

Model Review: k Nearest Neighbor (kNN)

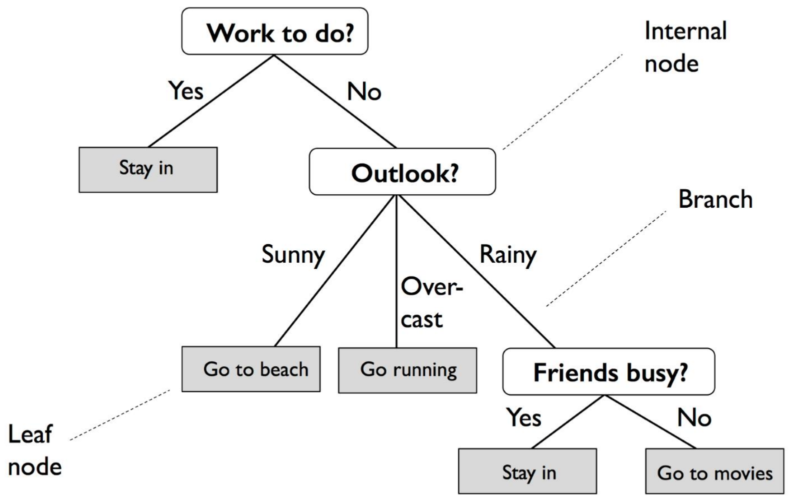
- Use for: Classification or Regression
- Pros:
  - fast to train
  - non-linear boundary
- Cons:
  - potentially slow to predict
  - curse of dimensionality



From PML

Model Review: Decision Tree

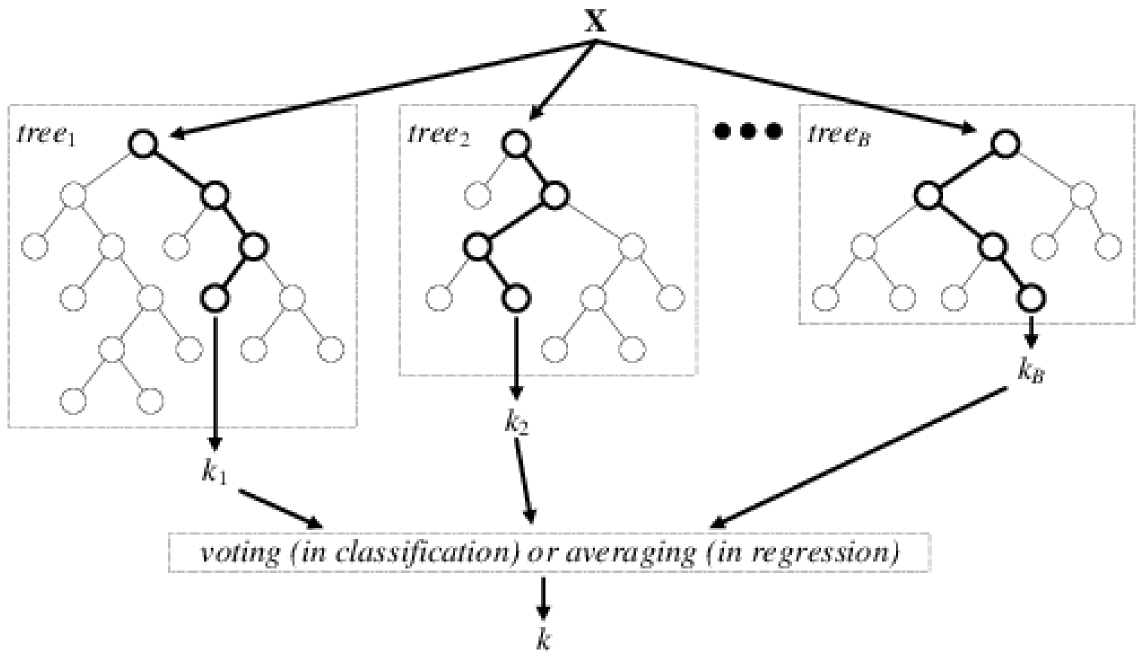
- Use for: Classification or Regression
- Pros:
    - very interpretable
    - quick to predict
    - can handle numeric and categorical variables without transformation
- Cons:
    - tendency to overfit (learn training set too well, more next class!)



From PML

Model Review: Random Forest (Ensemble via Bagging)

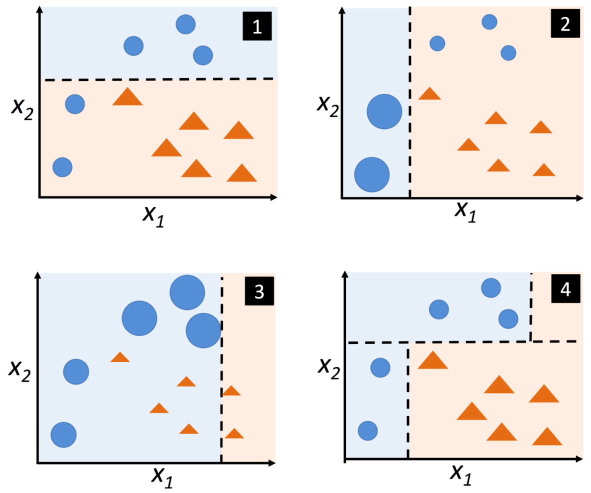
- Use for: Classification or Regression
- Pros:
    - less likely to overfit than decision tree
    - quick to train (through parallelization, quick to predict)
- Cons:
    - less interpretible, though still possible



From https://www.researchgate.net/publication/301638643_Electromyographic_Patterns_during_Golf_Swing_Activation_Sequence_Profiling_and_Prediction_of_Shot_Effectiveness

Model Review: Gradient Boosted Trees (Ensemble via Boosting)

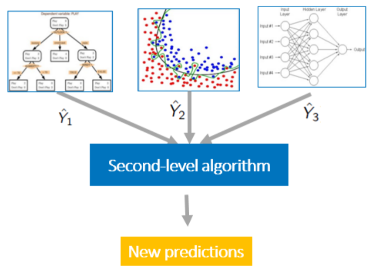
- Use for: Classification or Regression
- Pros:
    - pays more attention to difficult decision regions
    - quick to predict
    - tends to work well on difficult tasks
- Cons:
    - slow to train (parallelization not possible)
    - less interpretible, though still possible



From PML

Model Review: Ensemble via Stacking

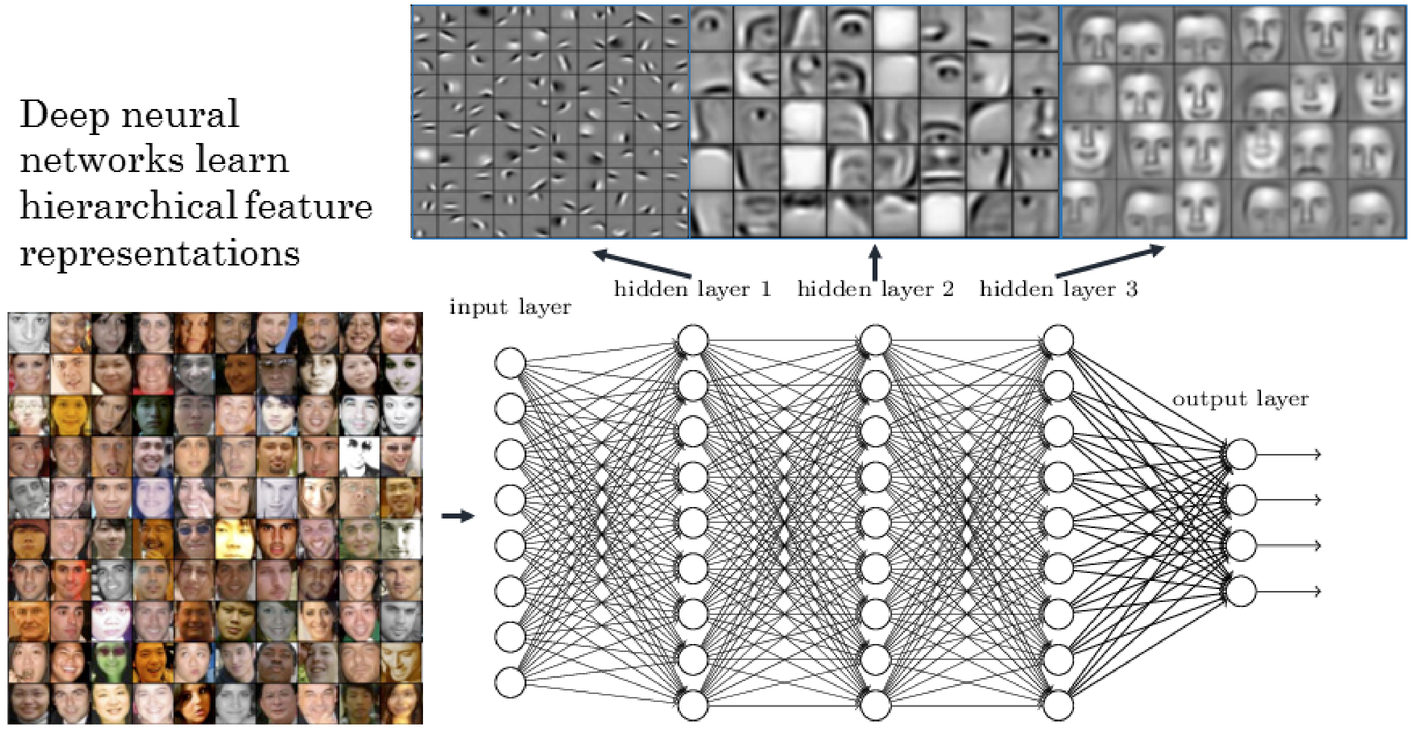
- Use for: Classification or Regression
- Pros:
  - combines benefits of multiple learning types
  - easy to implement
  - tends to win competitions
- Cons:
  - difficult to interpret
  - training/prediction time depends on component models



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)

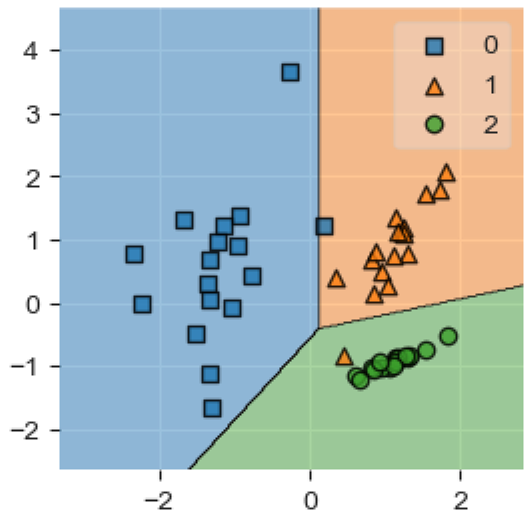


Playing with synthetic classification datasets

Playing with synthetic classification datasets

In [28]:

```
from sklearn.datasets import make_classification, make_multilabel_classification

X_syn,y_syn = make_classification(n_samples=50,
                                  n_features=2,
                                  n_informative=2,
                                  n_redundant=0,
                                  n_clusters_per_class=1,
                                  class_sep=1,
                                  n_classes=3,
                                  random_state=0,
                                 )
fig,ax = plt.subplots(1,1,figsize=(3,3))
plot_decision_regions(X_syn,y_syn,LogisticRegression().fit(X_syn,y_syn));
```



Playing with synthetic classification datasets - multilabel

Playing with synthetic classification datasets - multilabel

In [29]:

```
X_syn_ml,y_syn_ml = make_multilabel_classification(n_samples=100,
                                                   n_features=2,
                                                   n_classes=5,
                                                   random_state=0
                                                  )
print(X_syn_ml[:10])
print()
print(y_syn_ml[:10])
```

```
[[24. 25.]
 [38. 15.]
 [39. 14.]
 [23. 20.]
 [26. 29.]
 [30. 16.]
 [22. 30.]
 [25. 22.]
 [29. 12.]
 [25. 21.]]

[[0 0 0 0 0]
 [0 0 1 0 1]
 [0 1 1 1 1]
 [1 1 1 1 1]
 [1 0 0 1 0]
 [0 1 0 1 1]
 [1 0 0 0 0]
 [1 1 1 1 0]
 [0 0 0 0 0]
 [0 0 1 1 0]]
```

Questions?