Elements Of Data Science - F2021

Week 6: Intro to Machine Learning Models Continued

10/18/2021

TODOs

- Readings:
    - PDSH 05.03 Hyperparameters and Model Validation
    - Recommended: https://scikit-learn.org/stable/model_selection.html
    - Recommended: PML Chapter 6 (Except for Pipelines)
    - Reference: https://scikit-learn.org/stable/supervised_learning.html
    - Reference: PML Chapter Chap 3 and 7
- Quiz 6, Due Sun Oct 24th, 11:59pm
- HW2 out end of the week
- **Midterm**
    - Online via gradescope, open-book, open-note, open-python
    - Released Monday Oct 25th 11:59pm
    - Due Wednesday Oct 27th 11:59pm
    - Have 24hrs after starting exam to finish
    - 30-40 questions (fill in the blank/multiple choice/short answer)
    - Questions asked/answered **privately** via Ed

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

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
```

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

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

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

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))
```

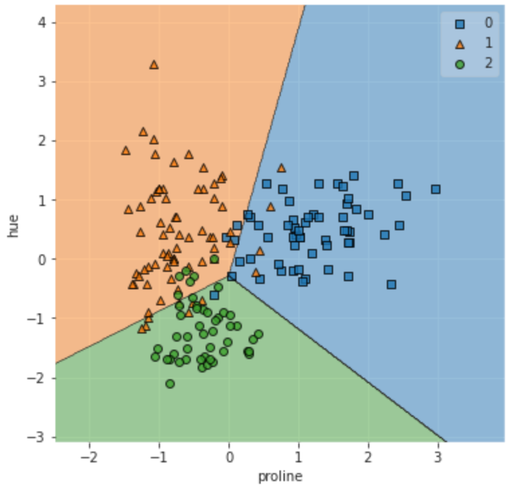
```
0 [0.967 0.031 0.001]
1 [0.146 0.853 0.001]
2 [0.176 0.344 0.48 ]
```

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))
```

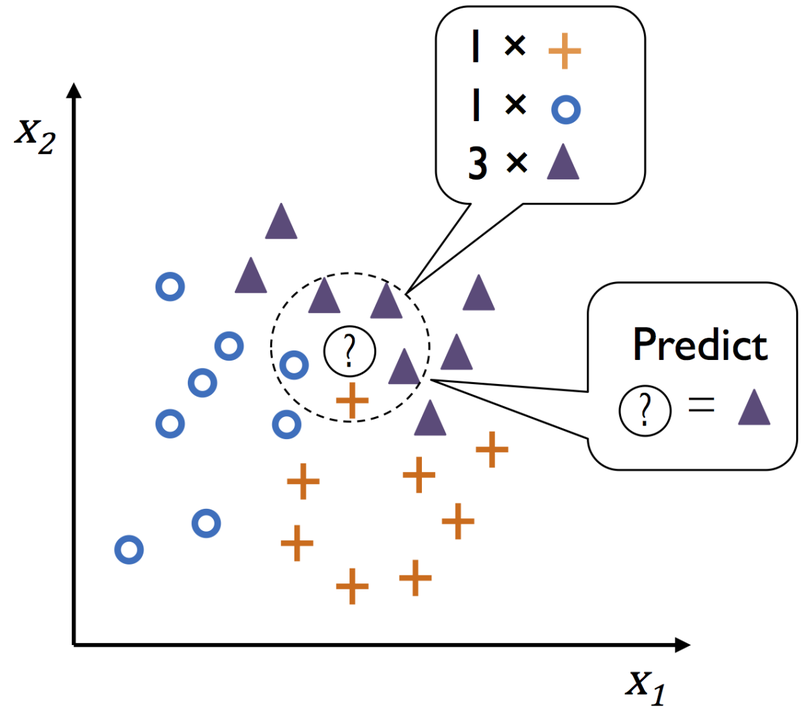
```
0 [0.967 0.031 0.001]
1 [0.146 0.853 0.001]
2 [0.176 0.344 0.48 ]
```

In [5]:
```
my_plot_decision_regions(X_zscore,y_c,logr,figsize=(6,6))
```



Distance Based: k-Nearest Neighbor (kNN)

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

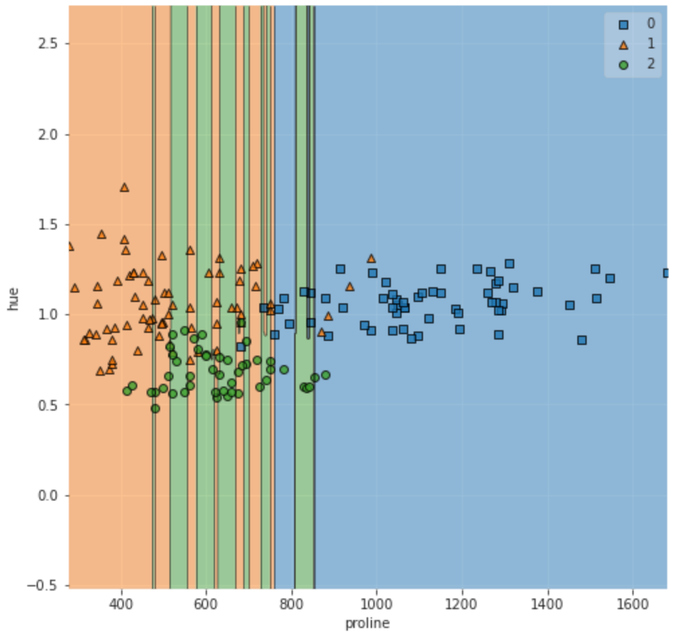


From PML

KNN in sklearn

KNN in sklearn

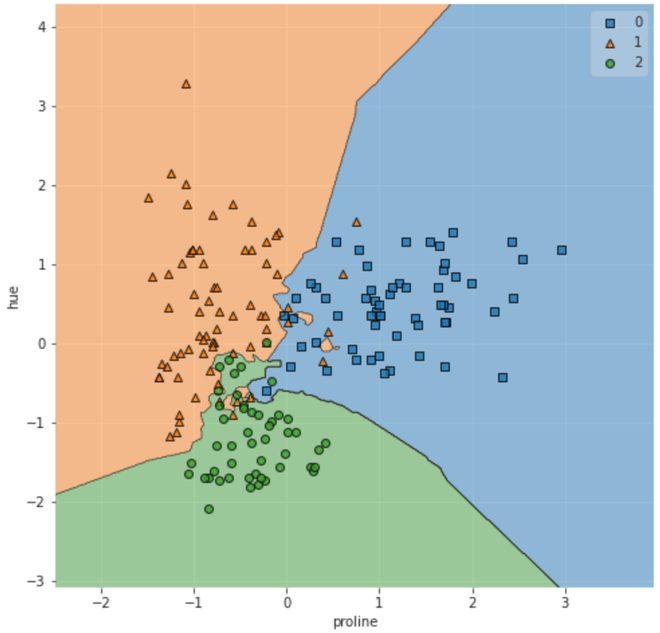
In [6]:
```
from sklearn.neighbors import KNeighborsClassifier

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

my_plot_decision_regions(X,y_c,knn)
```



Effects of Standardization on Distance Based Methods

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)
```



Curse of Dimensionality

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

Curse of Dimensionality

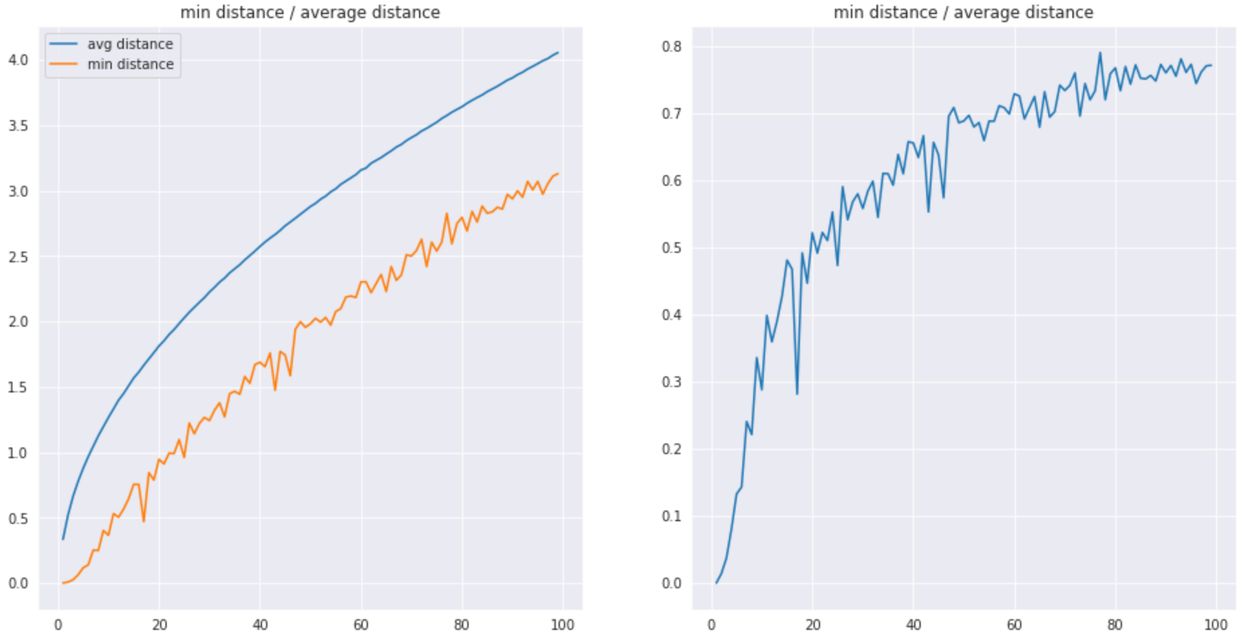
The more dimensions, the less likely points are "close" to each other.

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.

Curse of Dimensionality Cont.

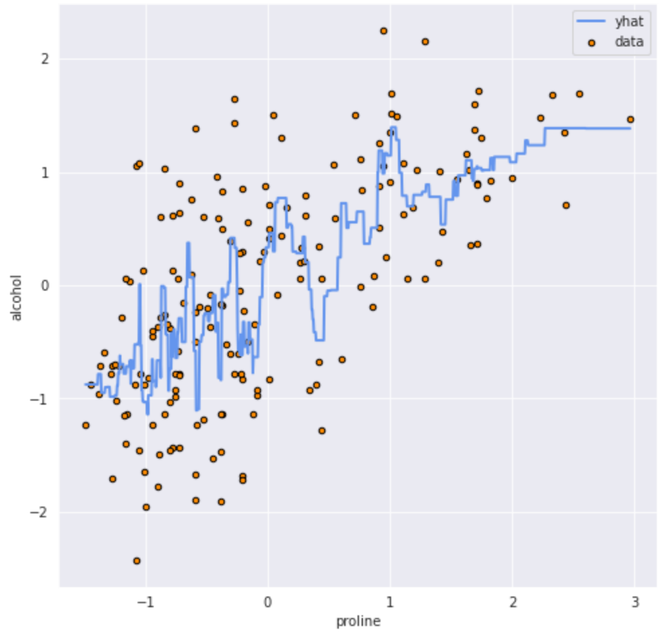
In [9]:
```
fig,ax = plt.subplots(1,2,figsize=(16,8))
ax[0].plot(dimensions,avg_distances,label='avg distance');
ax[0].plot(dimensions,min_distances,label='min distance');
ax[0].legend()
ax[0].set_title('min distance / average distance');
ax[1].plot(dimensions,min_avg_ratio)
ax[1].set_title('min distance / average distance');
```



Regression with kNN

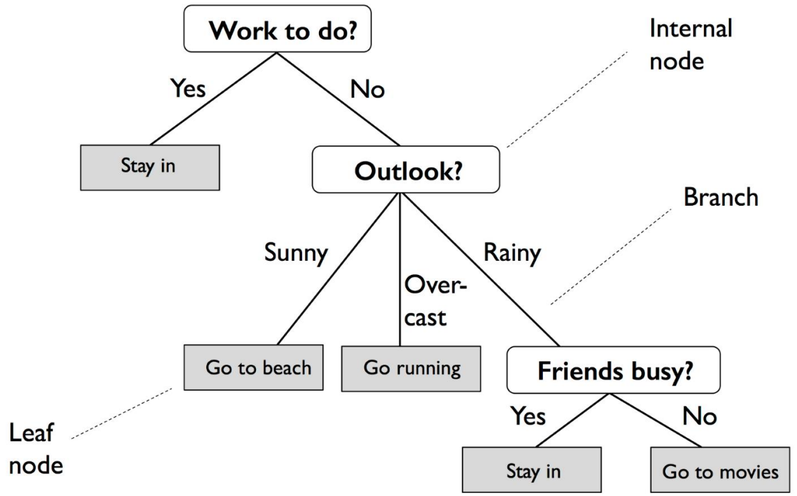
Regression with kNN

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)
```



Decision Tree

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

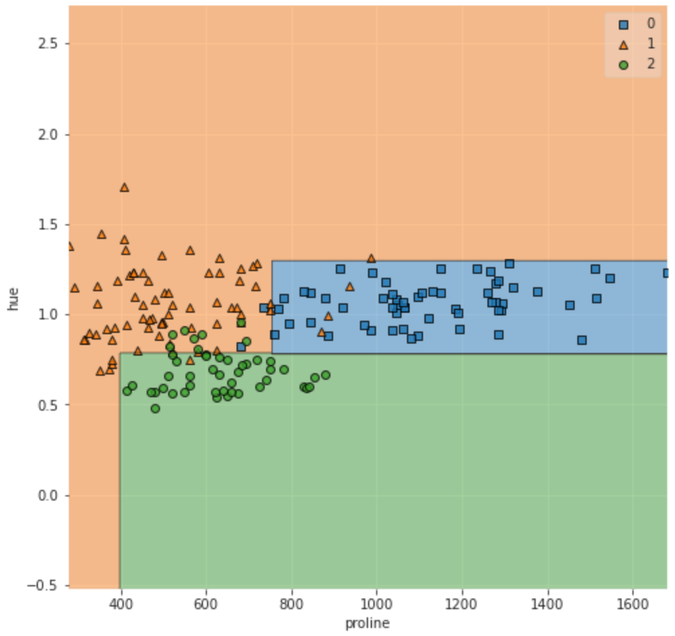


From PML

Decision Tree Classifier in sklearn

Decision Tree Classifier in sklearn

In [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)
```

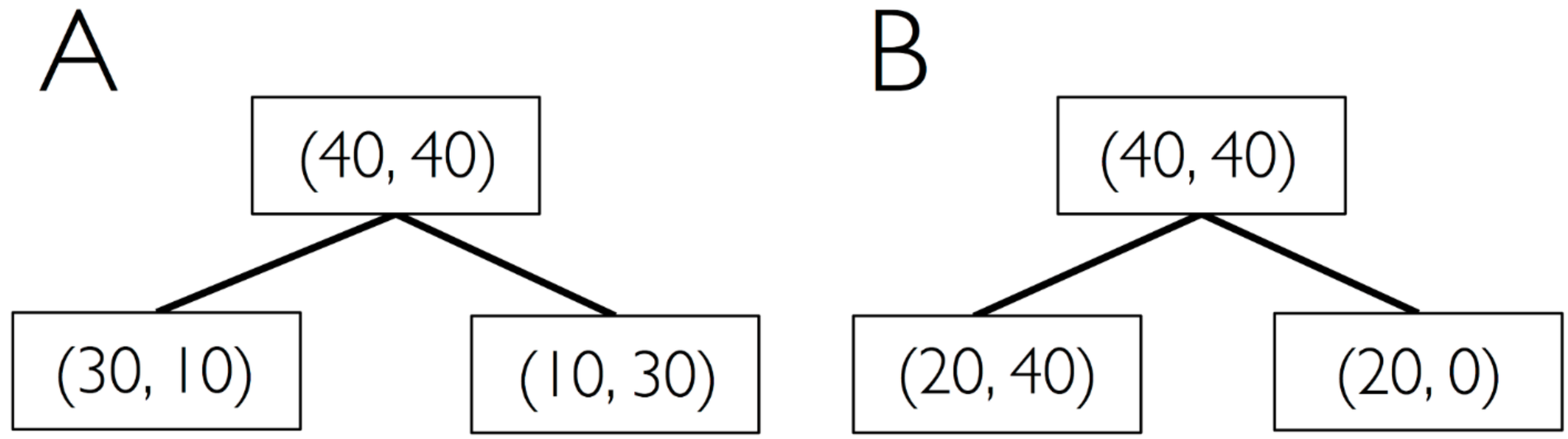


Building a Decision Tree

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

Building a Decision Tree

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

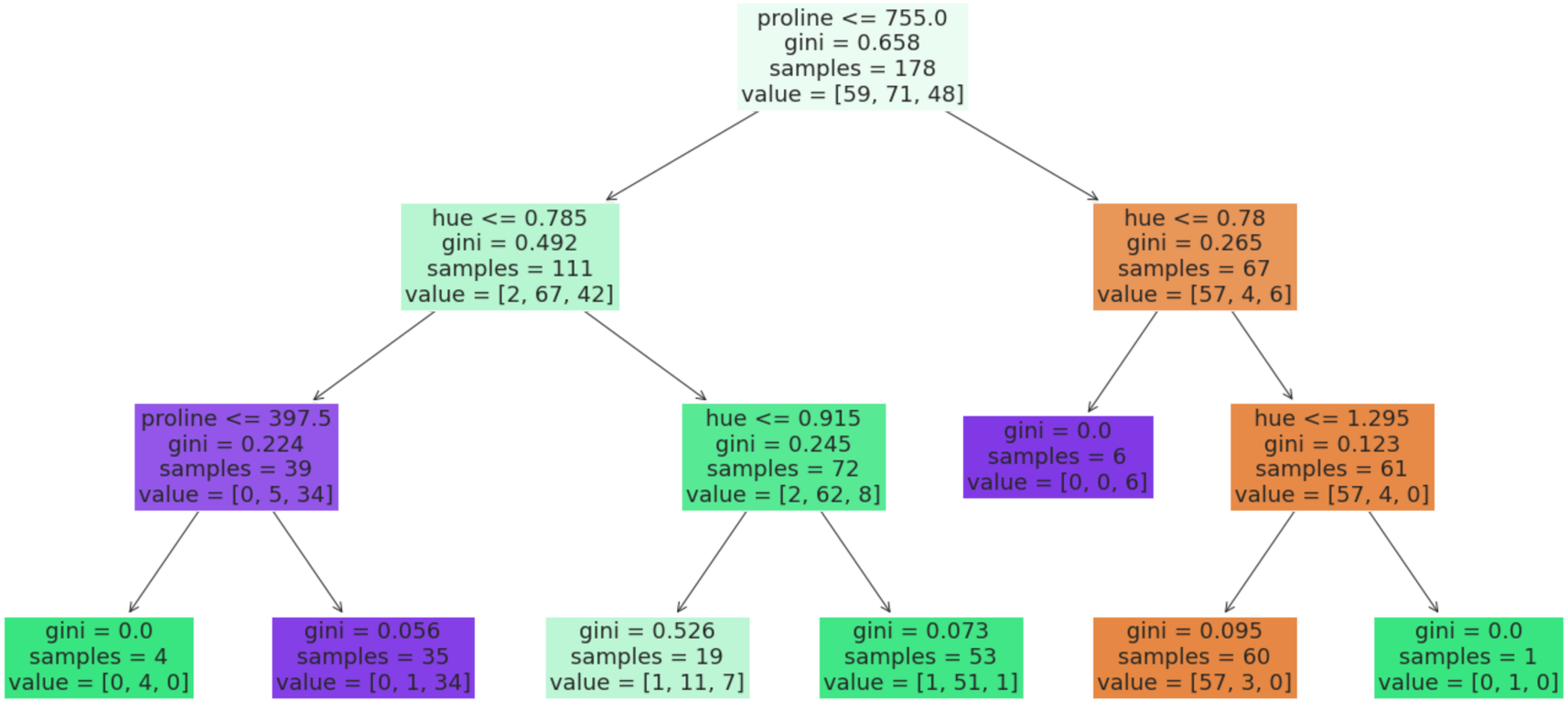


From PML

Plot Learned Decision Tree Using sklearn

Plot Learned Decision Tree Using sklearn

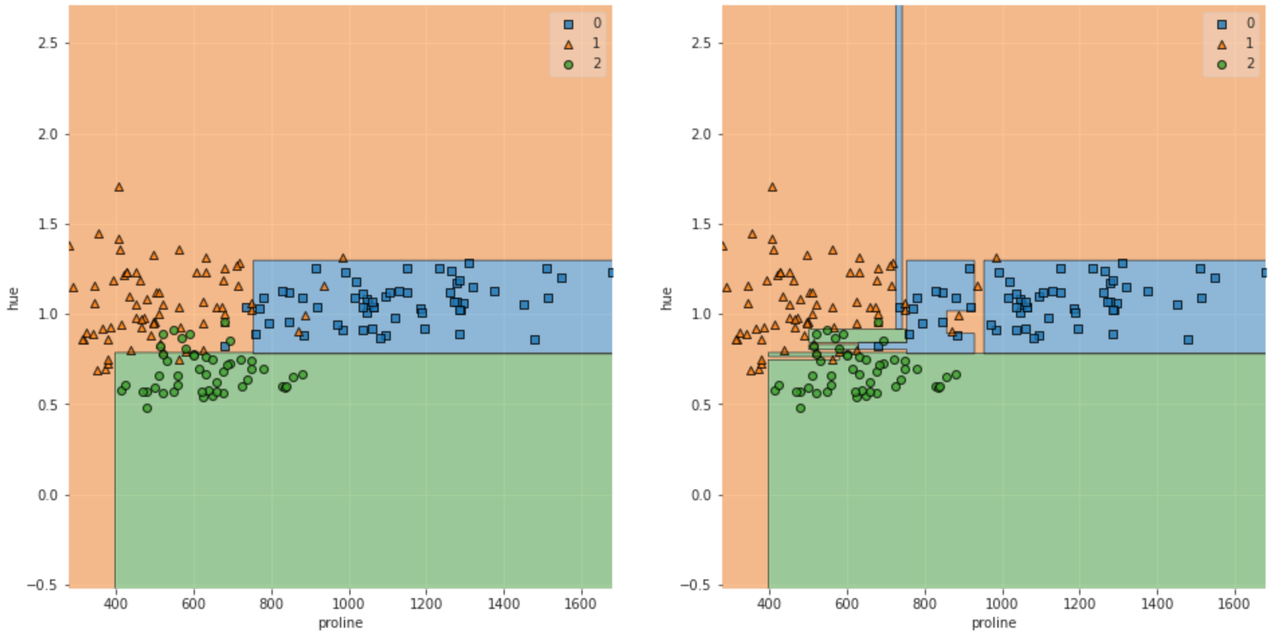
In [14]:
```
from sklearn.tree import plot_tree
fig,ax = plt.subplots(1,1,figsize=(24,12))
plot_tree(dtc_md3,ax=ax,fontsize=18,feature_names=X.columns,filled=True);
```



Decision Tree: Increase Maximum Depth

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]);
```



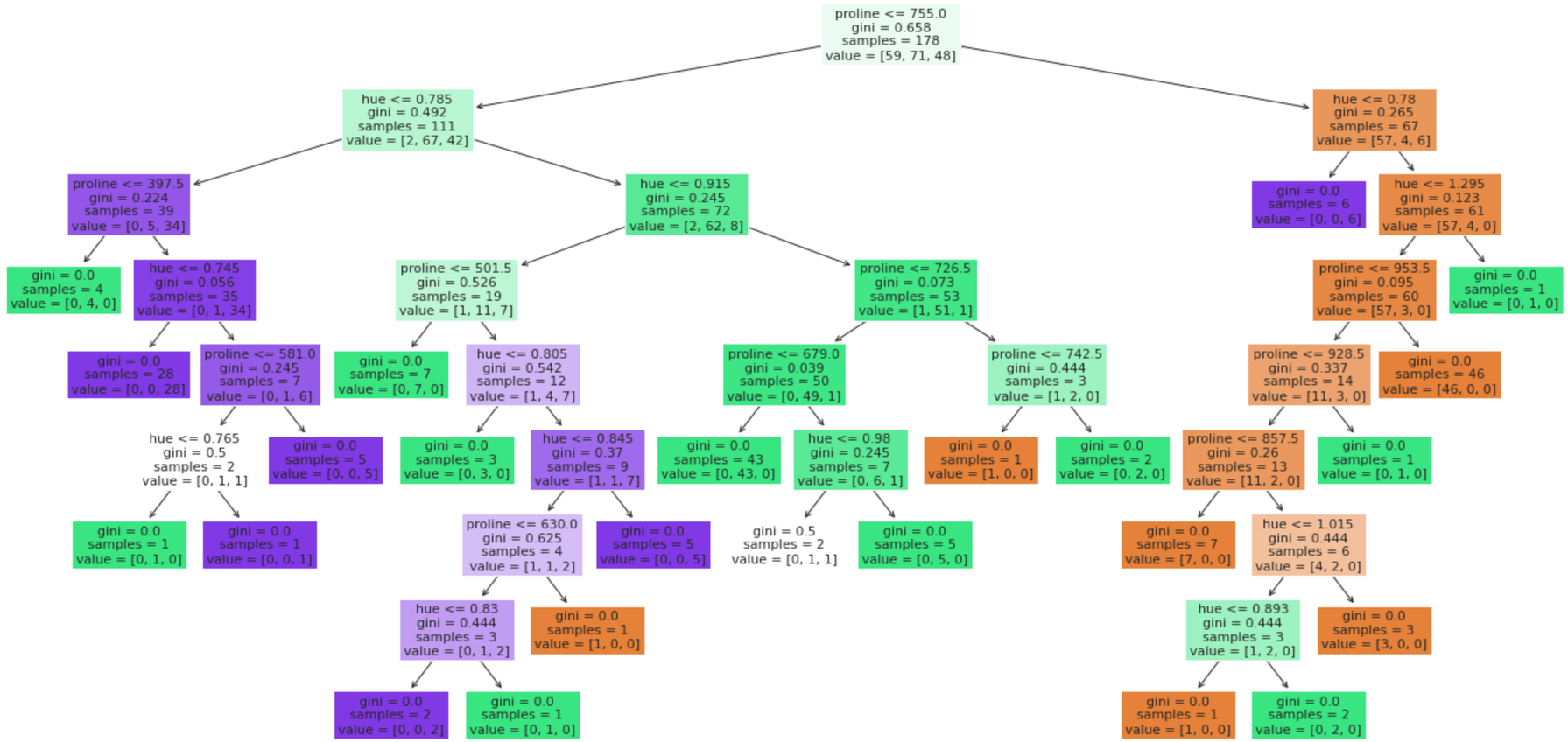
Plot Learned Decision Tree Using sklearn

- For tree with max_depth=10

Plot Learned Decision Tree Using sklearn

- For tree with max_depth=10

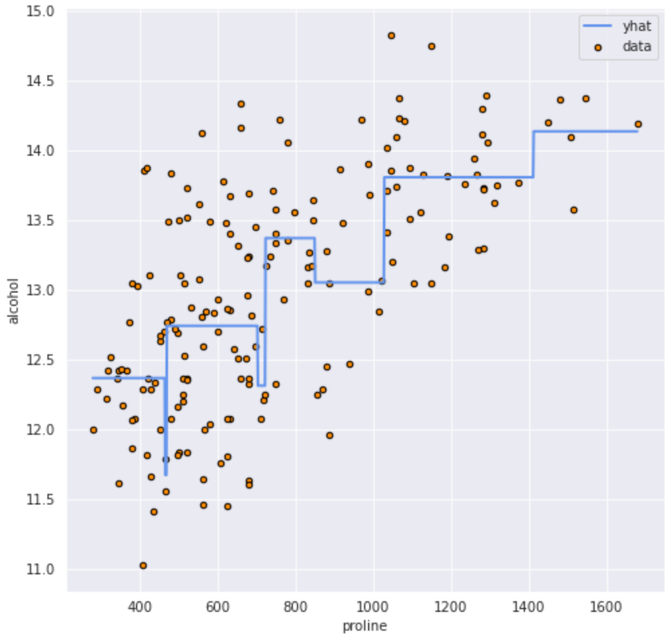
In [16]:
```
fig,ax = plt.subplots(1,1,figsize=(24,12))
plot_tree(dtc_md10,ax=ax,fontsize=11,feature_names=X.columns,filled=True);
```



Regression with Decision Trees

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)
```



Ensemble Methods

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

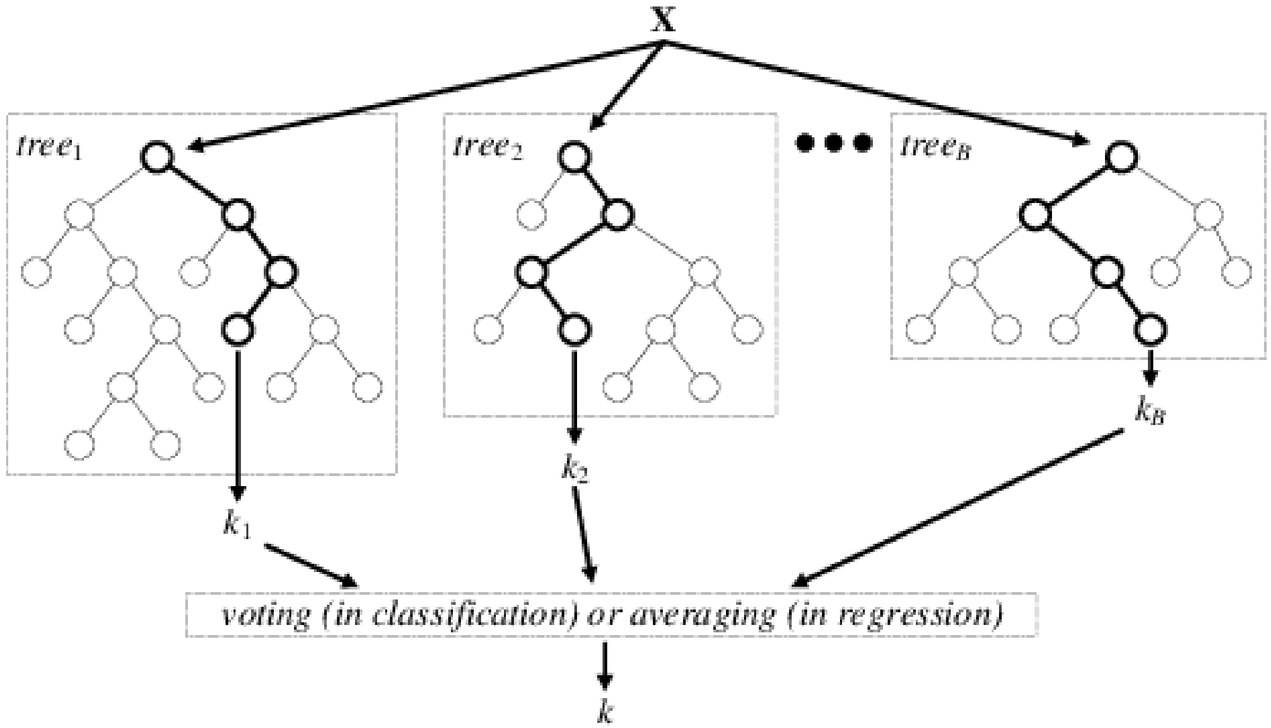
Ensemble Methods

- "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

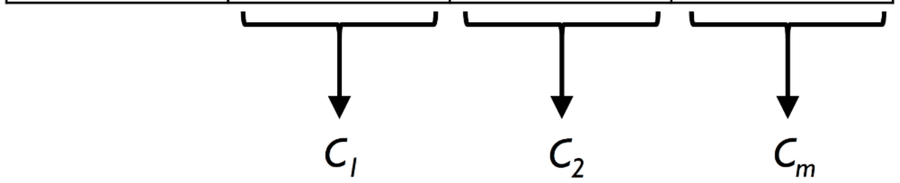


From https://www.researchgate.net/publication/301638643_Electromyographic_Patterns_during_Golf_Swing_Activation_Sequence_Profiling_and_Prediction_of_Shot_Effectiveness

Bagging with Random Forests

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

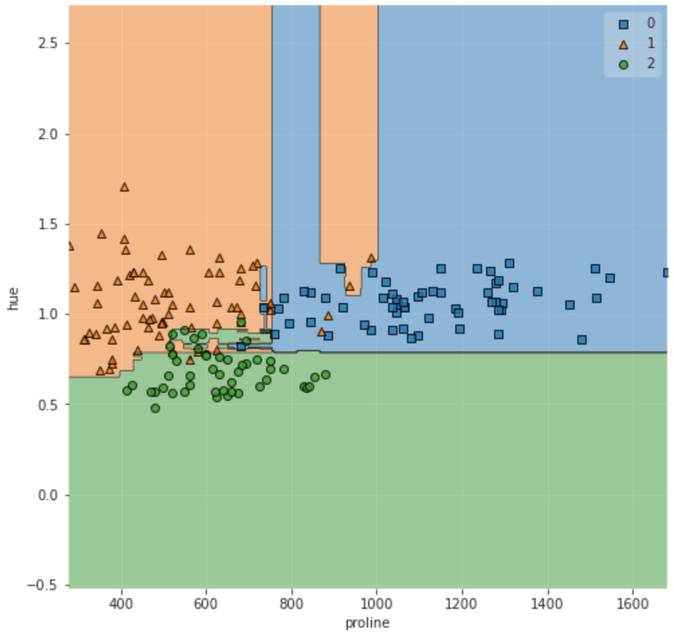
Sample indices	Bagging round 1	Bagging round 2	…
1	2	7	…
2	2	3	…
3	1	2	…
4	3	1	…
5	7	1	…
6	2	7	…
7	4	7	…



Random Forests with sklearn

Random Forests with sklearn

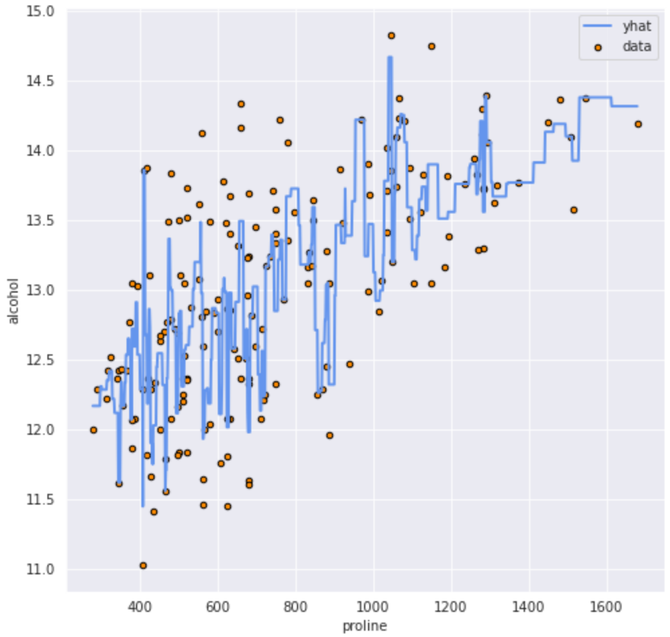
In [18]:
```
from sklearn.ensemble import RandomForestClassifier

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)
```



Regression with RandomForests

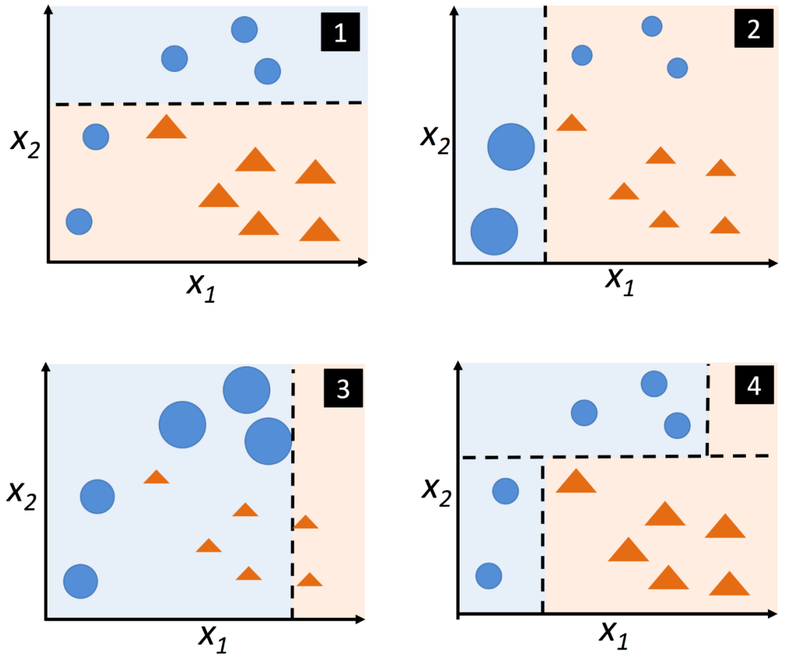
Regression with RandomForests

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)
```



Gradient Boosted Trees

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

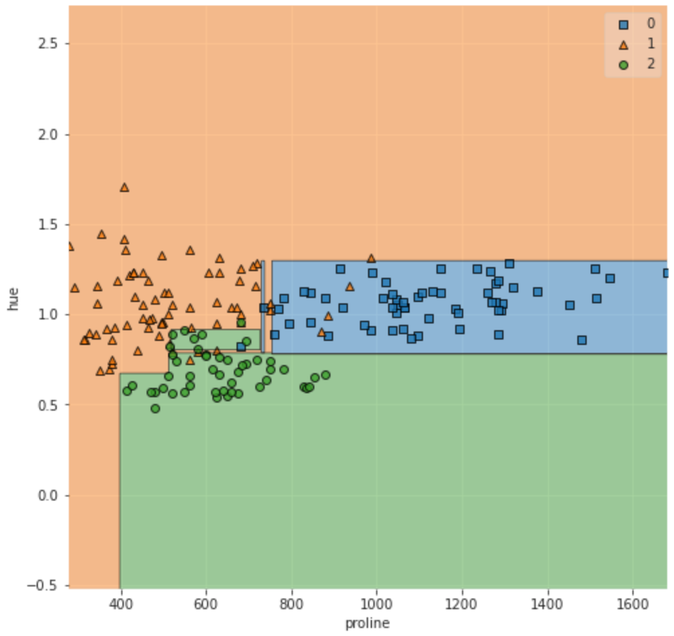


From PML

Gradient Boosted Trees in sklearn

Gradient Boosted Trees in sklearn

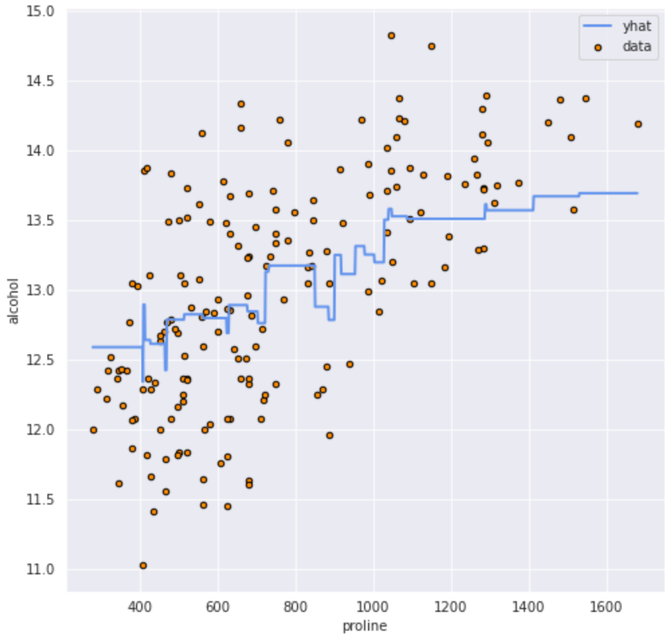
In [20]:
```
from sklearn.ensemble import GradientBoostingClassifier

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

my_plot_decision_regions(X,y_c,gbc)
```



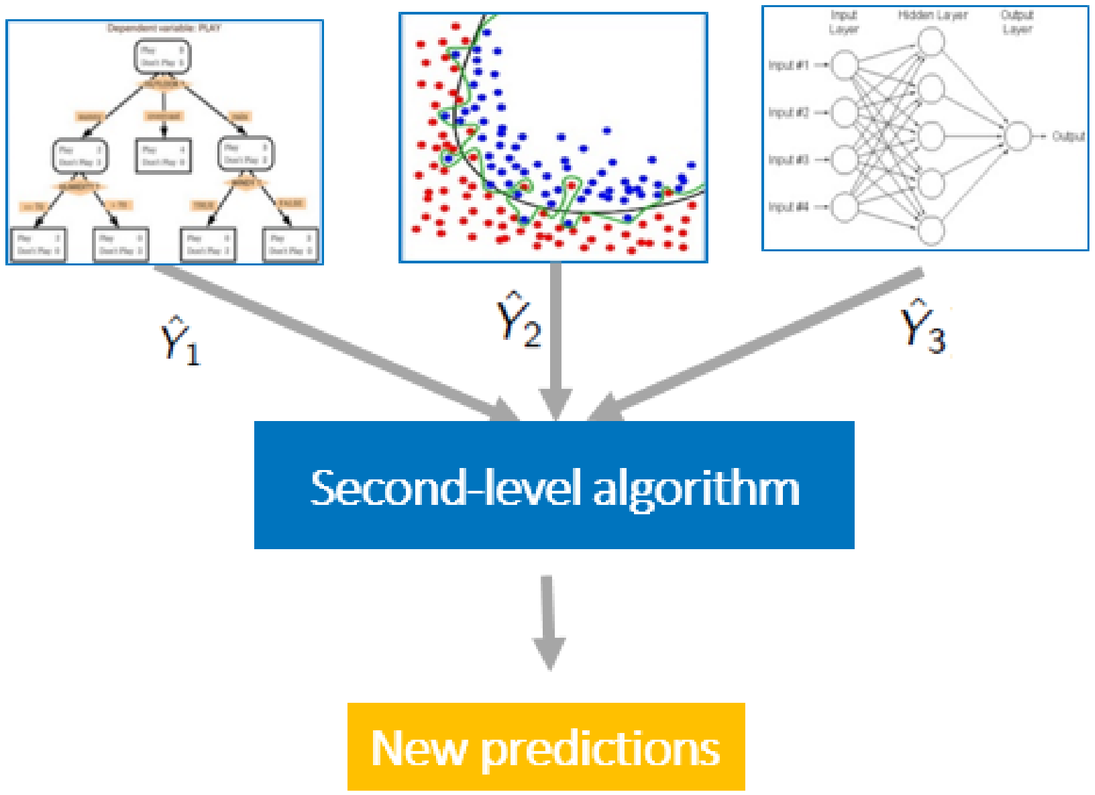
Regression with Gradient Boosted Trees

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)
```



Stacking

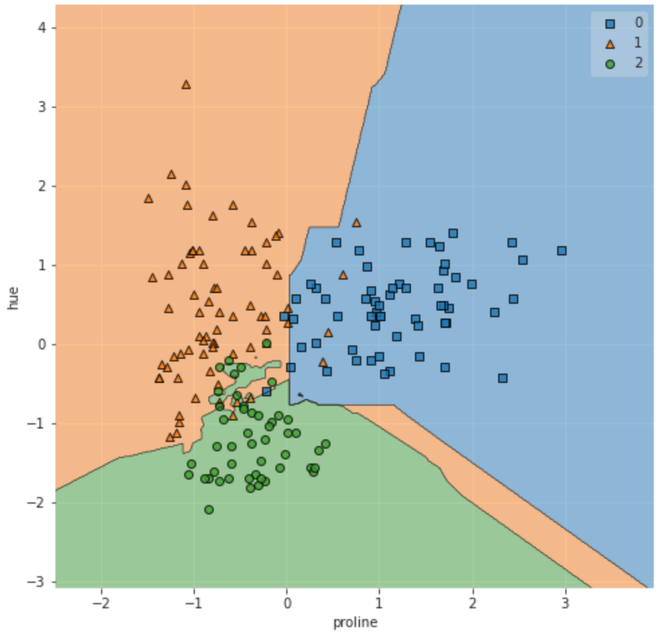


From https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

Stacking for Classification

Stacking for Classification

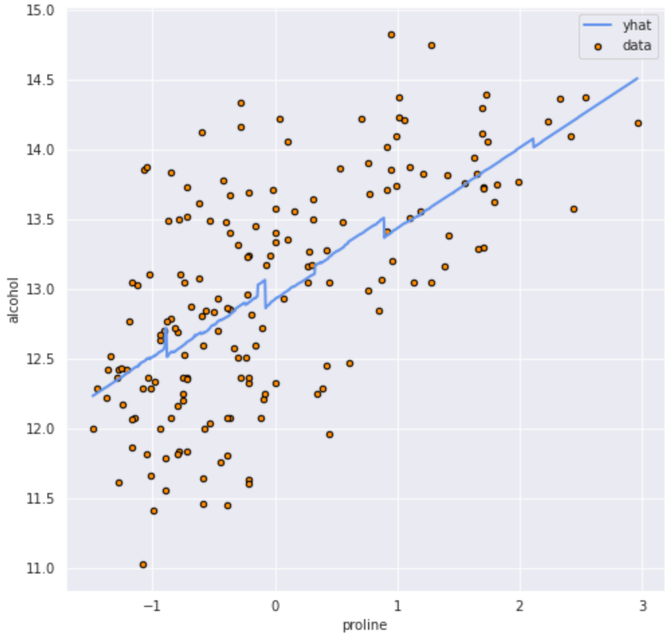
In [22]:
```
from sklearn.ensemble import StackingClassifier

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)
```



Stacking for Regression

Stacking for Regression

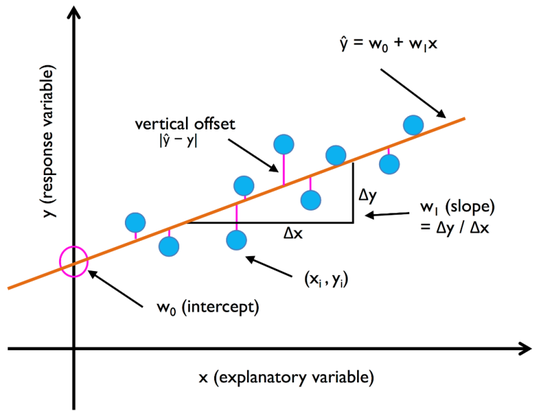
In [23]:
```
from sklearn.ensemble import StackingRegressor

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)
```



Review of Models

Model Review: Simple/Multiple Linear Regression

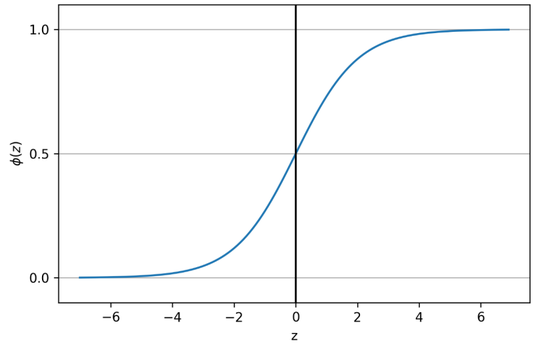
- Use for: Regression
- Pros:
    - fast to train
    - interpretable coefficients
- Cons:
    - assumes linear relationship
    - depends on removing colinear features



from PML

Model Review: Logistic Regression

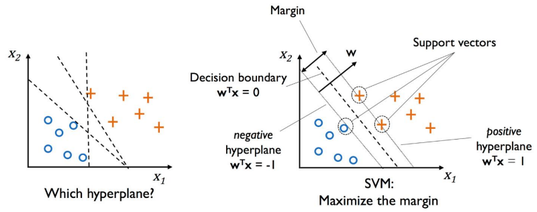
- Use for: Classification
- Pros:
    - fast to train
    - interpretable coefficients (log odds)
- Cons:
    - assumes linear boundary
    - depends on removing colinear features



from PML

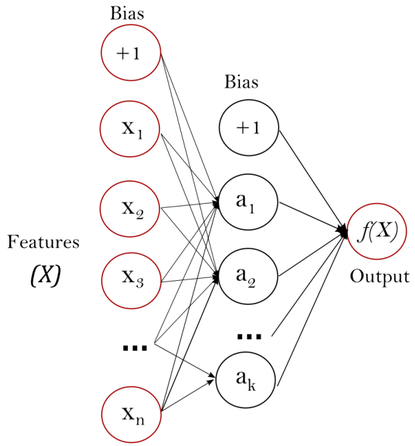
Model Review: Support Vector Machine (SVM)

- 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



Model Review: Multi-Layer Perceptron

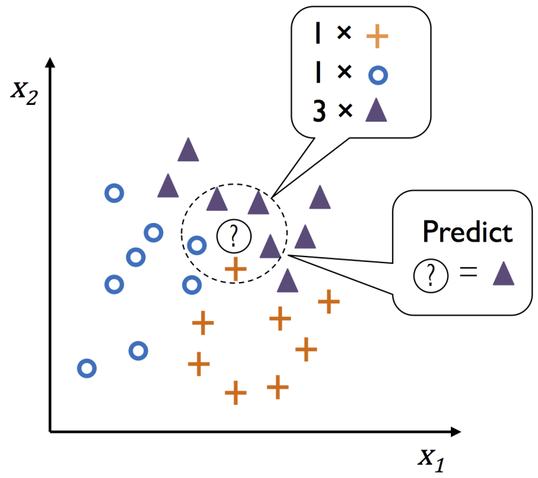
- Use for Classification or Regression
- Pros:
    - non-linear boundary
- Cons:
    - non-convex loss function (sensitive to initial weights)
    - sensitive to feature scaling
    - no GPU support in sklearn: use tensorflow or pytorch



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

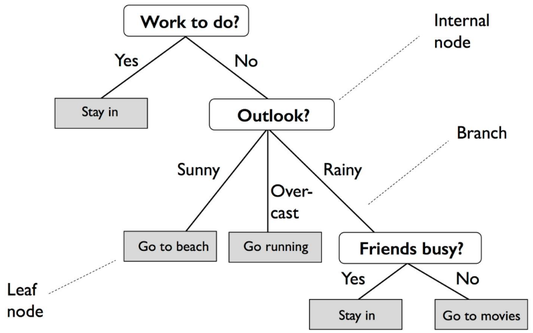
Model Review: k Nearest Neighbor (kNN)

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



Model Review: Decision Tree

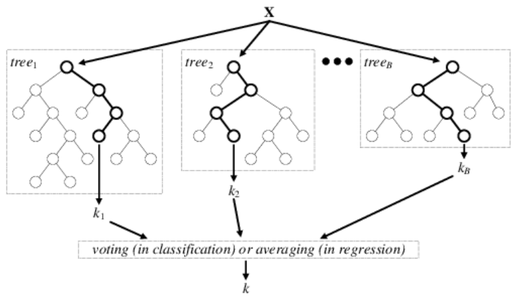
- Use for: Classification or Regression
- Pros:
    - very interpretable
    - quick to predict
    - can handle numeric and categorical variables without transformation
- Cons:
    - tendency to overfit (learn training set too well, more next class!)



From PML

Model Review: Random Forest (Ensemble via Bagging)

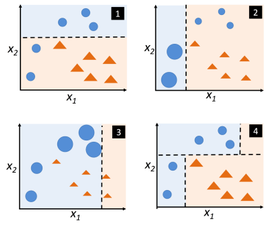
- Use for: Classification or Regression
- Pros:
    - less likely to overfit than decision tree
    - quick to train (through parallelization, quick to predict
- Cons:
    - less interpretible, though still possible



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)

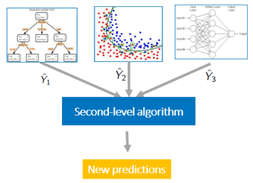
- Use for: Classification or Regression
- Pros:
    - pays more attention to difficult decision regions
    - quick to predict
    - tends to work well on difficult tasks
- Cons:
    - slow to train (parallelization not possible)
    - less interpretible, though still possible



From PML

Model Review: Ensemble via Stacking

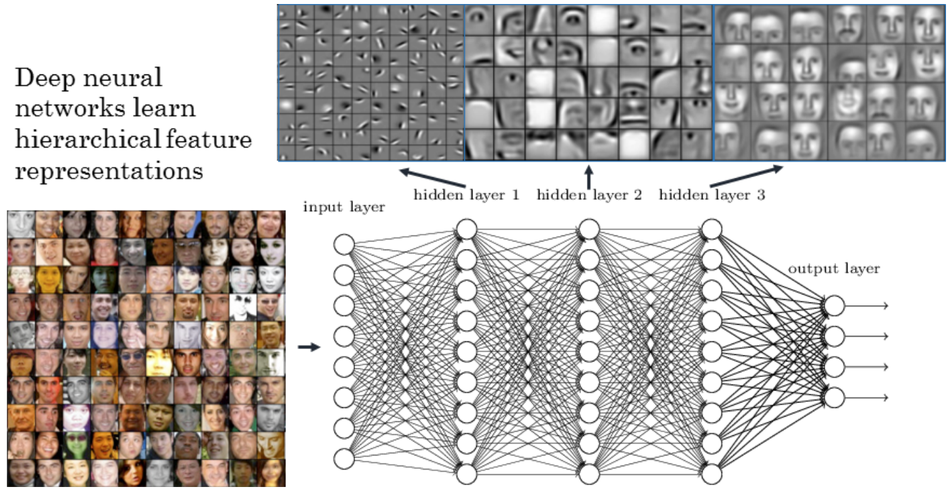
- Use for: Classification (or Regression)
- Pros:
    - combines benefits of multiple learning types
    - easy to implement
    - tends to win competitions
- Cons:
    - difficult to interpret
    - training/prediction time depends on component models



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)



Questions?