# Determing if a Olympic Weightlifter will place

## Written by Taylor Smith

For my project, I wanted to determine if there are certain qualities for olympic weightlifters that can be used to predict if they will place in the standings. The CSV file I have imported contains all Olympic Athletes so I simplified it immediatly to just the weightlifters I was concerned with.

In [31]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score
from sklearn.ensemble import RandomForestClassifier
```

```
Populating the interactive namespace from numpy and matplotlib
```

# Preprocessing Olympics Medal Data

The order in which I processed/ the way I seperated the data is as follows

1. Read the Olympics CSV with pandas
2. Grab all of the row entries where the sport is Weightlifting (Only interested in using weightlifting)
3. Drop all of the pandas rows that shouldn't effect whether or not a lifter placed (got a medal)
4. Use LabelEncoder to turn the sex column into 0 for female and 1 for male
5. For simplicity of the model, I turned my dependent column into whether or not the lifter placed (got any medal) instead of buidling a model for just gold medals
6. Lastly, I split the data into training and test splits

In [32]:
```
data = pd.read_csv('athlete.csv')

#Grabbing all of the Weightlifting Data
data = data.loc[data['Sport'] == 'Weightlifting']

#Dropping Columns that shouldnt effect classification
data = data.drop(['Name','ID','Team', 'Games', 'Sport','City','Season','Event','N

#Label Encoding by sex
data['Sex'] = LabelEncoder().fit_transform(data['Sex'])

#Reclassifing the Medals Column as either a 0 for no medal, or 1 for got medal
data = data.replace(np.nan, 0)
data = data.replace('Bronze', 1)
data = data.replace('Silver', 1)
data = data.replace('Gold', 1)
x = data.iloc[:,:4]
y = data['Medal']

#Splitting up the dataset
X_train, X_test, medal_train, medal_test = train_test_split(x, y, test_size=0.25,
```

# Running a Grid Search using a Decision Tree Classifer

The first and simpliest idea for predicting if an athlete will place (get gold, silver, or bronze) was to use a Decision Tree Classifer, so I ran a Grid Search to find the best hyperparameters for the Decision Tree.

The Best peramters were

- min_samples_leaf = 2
- max_depth = 3

Below the Grid search I ran the best Decision Tree and recorded the best score

In [40]:
```
tree_clf = DecisionTreeClassifier()
tree_search_params = {'min_samples_leaf':[1,2,3,4,5,10],'max_depth':[3,4,5,6,7,10
tree_search = GridSearchCV(tree_clf, tree_search_params, cv=5, verbose=0)
tree_search.fit(x, y)
print("The best Hyper Parameters are " + str(tree_search.best_params_))

#Running with the best Hyper Parameters
tree = tree_search.best_estimator_
print("The Best Decisions Tree Model's Accuracy Score is "+ str(tree_search.best_
```

```
The best Hyper Parameters are {'max_depth': 3, 'min_samples_leaf': 10}
The Best Decisions Tree Model's Accuracy Score is 0.8359156718313436
```

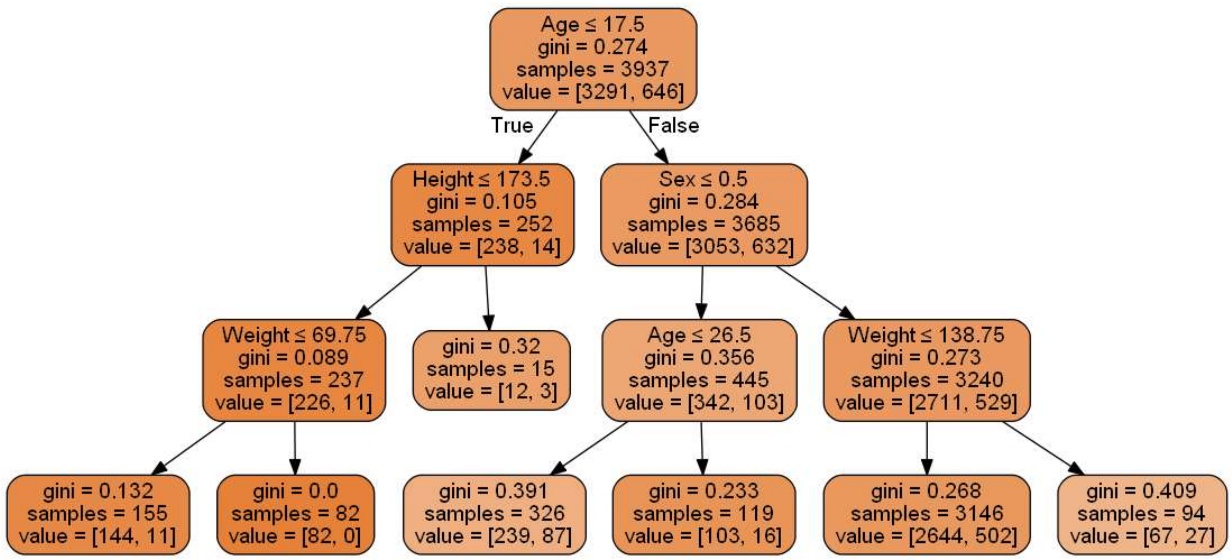
# Printing out the Decision Tree

Below I Print the Decision Tree visually to see what is was using to classify the data

In [41]:
```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

dot_data = StringIO()
export_graphviz(tree, out_file=dot_data, feature_names=list(x.columns),
 filled=True, rounded=True,
 special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[41]:




# Running a Grid Search with Random Forest

In [42]:
```
#Running a Grid search on the Random Forest Algorithm
forest_clf = RandomForestClassifier()
forest_search_params = {'min_samples_leaf':[1,2,3,4,5,10],'n_estimators':[10,20,3
forest_search = GridSearchCV(forest_clf, forest_search_params, cv=5, verbose=0)
forest_search.fit(x, y)
print("The best Hyper Parameters are " + str(forest_search.best_params_))
```

```
The best Hyper Parameters are {'max_depth': 3, 'min_samples_leaf': 4, 'n_estima
tors': 10}
```

In [44]:
```
#Running a Random Forest with the best Hyper Parameters
forest = forest_search.best_estimator_
print("The Best Random Forest Classifier has a accuarcy score of "+ str(forest_se
```

```
The Best Random Forest Classifier has a accuarcy score of 0.8361696723393447
```

# Comparing the Random Forest With Just the Decision Tree

After Running a grid search on each model to find the best hyperparameters for both, it turns out the the Random Forest took different hyperparameters as the decision tree but it wasn't able to improve upon the orginial decision tree. This is evident by nearly the same accuracy scores for the best decision tree and the best random forest. We can conclude that the most effective way of predicting whether or not a weightlifter will place (get a medal) based on

- Age
- Weight
- Height

is the Decision Tree classifier. My predicition for why the Random Forest did not improve the performance of the Decision Tree is that anything more complext than the Decision Tree (Greater Depth) Began to hardcode the weightlifters based on which person was generally a better weightlifter, instead the classifier I have made here seems to do better by keeping the Gini values low, and not hardcoding the specific weightlifters who were the best or were the worst.

In conclusion, I am confident that this Decision Tree classifer would do a good job at predicting future olympians on whether or not they will recieve a medal by their age weight and height, due to the fact that the Gini values are low, but not completely homogenous.