Lab 3 – Predicting a categorical target, performance (95)

In this lab we are going to use a model to predict the gender (male =0 or male=1) of people from the Howell data set. We will get performance metrics and create graphs.

Analysis prelims

We are going to make a copy of our data exploration notebook (lab2.ipynb) and name it lab3_starter.ipynb. We will trim that notebook down and do our training and analysis there.

We need to change the intro cell. Double click on it and change it to:

```
# Lab 3 - Exploring and manipulating data.
Here is what we will do:
1. Prepare the data
2. Train 3 models
    - Decision Tree
    - SVM using rgb
    - Neural Net
3. Get model performance on train and test sets
4. Create appropriate graphs
```

Click on play while the cell is selected and it will re-render it.

Change the next cell to the following and then re-render it

```
### Prepare the data
```

Removing exploration code.

There is a lot of code that we don't need when creating our models, so let's get rid of it.

In the third cell, get rid of the line howell_full.info().

Select and cut the two cells for **Quick Look at Distributions:**

Select and cut the two cells for **Quick Visualization:**

In the code for **A Better Plot** comment out the first and third plot.

Select and cut the two cells for **Handling Missing Data**: (Normally we would keep any cleaning needed for the data, but the Howell dataset was fine as is.)

While we will not use BMI for our models, it does not hurt to include it. Similarly for the BMI class.

Remove the two print statements from **Adding a new Feature** and paste what remains at the end of the #third cell. When you run the third cell, the head should have the bmi feature.
You may now cut the two cells for **Adding a new Feature.**

Remove the print statement from the first code cell for **Creating a Categorical Feature** and paste what remains at the end of the #third cell. When you run the #third cell, category counts should match from before.
You may now cut the three cells for **Creating a Categorical Feature.**

Keep **Splitting the Data by Age**. (Remove or comment out the print statements.) We will just work with the adults since there is a nice break in the data that is not there for the children.

Keep **Plot with masking**

Keep the two **Train/Test Data Split** cells. Remove the last three print statements.

Remove the cells from **Stratified Train/Test Data Split** cells.

You should see train/test counts of

```
Train size:  276 Test size:  70
```

Submission 1 of 12: Progress mark (10)
Screen shot preparation code.

Train and evaluate the model.

Create a new text cell after **Train/Test Data Split** cell and add in the following

```
### Train and evaluate a Linear Regression Model
```

Create a new code cell and add in the following:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import precision_score, recall_score
```

1. These imports give us our model and the metric functions.

Add the following lines to the code cell:

```
X = train_set[['height']]
y = train_set['male']

X_test = test_set[['height']]
y_test = test_set['male']
```

1. Predict on the feature height.
2. Use male as the target feature
3. Test the trained model height
4. Use male as the target feature

Add the following lines to the code cell:

```
tree_model = DecisionTreeClassifier()
tree_model.fit(X,y)
```

1. Create an instance of a decision tree model.
2. Train the model on the training data

Add the following lines to the code cell:

```
y_pred = tree_model.predict(X)
print('Results for decision tree on training data')
print('  Default settings')
print("Confusion Matrix")
print(confusion_matrix(y, y_pred))
print('Accuracy is  ', accuracy_score(y, y_pred))
print('Precision is ', precision_score(y, y_pred))
print('Recall is    ', recall_score(y,y_pred))
print('F1 is        ', f1_score(y, y_pred))
```

1. Get the predictions of the model on the training data
2. Print the results for the model on the training data. Notice that all of the performance metrics need the actual value and predicted value for the target.
3. Train the model on the training data

Run the code cell and you should see something like

```
Results for decision tree on training data
  Default settings
Confusion Matrix
[[139   6]
 [ 22 109]]
Accuracy is   0.8985507246376812
Precision is  0.9478260869565217
Recall is     0.8320610687022901
F1 is         0.8861788617886178
```

:
Go to the end of the note book and add in a markdown cell. This is where we are going to store our results.
Add the following lines to the cell:

```
# Results
Basic results for our classification model to predict gender
on the Howell data.

| Model | Training Features | Set | Accuracy | F1 |
|:---|:---|:---|:---|:---|
|Decision Tree|Height|Training|89.85|88.62|
```

1. We are going to record the accuracy and f1 scores as a baseline. Depending on the goals of your analysis, you may want to record other values as well.
2. I am going to record values as percentages rounded to the second place. Even though the model reports more precise values, with such a small data set the extra places are probably not significant.

Add the following lines to the code cell:

```
y_test_pred = tree_model.predict(X_test)
print('Results for decision tree on test data')
print('  Default settings')
print("Confusion Matrix")
print(confusion_matrix(y_test, y_test_pred))
print('Accuracy is  ', accuracy_score(y_test, y_test_pred))
print('Precision is ', precision_score(y_test, y_test_pred))
print('Recall is    ', recall_score(y_test,y_test_pred))
print('F1 is        ', f1_score(y_test, y_test_pred))
```

1. This is a modified copy of the performance reporting code for the training set. Feel free to copy and modify or retype.

Run the code cell. You should get something like:

```
Results for decision tree on test data
  Default settings
Confusion Matrix
[[33  4]
 [ 9 24]]
Accuracy is   0.8142857142857143
Precision is  0.8571428571428571
Recall is     0.7272727272727273
F1 is         0.7868852459016394
```

Submission 2 of 12: Progress mark (10)
Screen shot of train/test performance.

Submission 3 of 12: Analysis (5)
Looking at the performance of the model on the training and test sets and make an argument on whether the model is overfitting the training set.

Record the results in the final cell by adding another row in the table

```
|Decision Tree|Height|Test|81.43|78.69|
```

Train on a different feature.

We have a couple options here. We can make a copy of the entire cell for training and performance reporting or we can comment out the old code which is replaced with the new. We are going to take the second approach here, since there are only a couple lines that need to be changed. Change the code to:

```
#X = train_set[['height']]
X = train_set[['weight']]
y = train_set['male']

#X_test = test_set[['height']]
X_test = test_set[['weight']]
y_test = test_set['male']
```

1. Use # to comment out the selection on height
2. Instead, select weight
3. Do the same for the test set.

Run the code cell.

Record the results in the final cell.

Submission 4 of 12: Analysis (10)

- Look at the performance of the models and state which model you prefer
- Propose an explanation of why one feature was better than the other.

Train on multiple features.

Change the code to:

```
#X = train_set[['height']]
#X = train_set[['weight']]
X = train_set[['height', 'weight']]
y = train_set['male']

#X_test = test_set[['height']]
#X_test = test_set[['weight']]
X_test = test_set[['height', 'weight']]
y_test = test_set['male']
```

Run the code cell.

Record the results in the final cell.

Submission 5 of 12: Analysis (5)
Which features would you propose are the best to train on?

Something to explore later: Does age improve the predictive power of our decision tree?

Scatter plot for Binary Categories.

In the notebook lab3_utility.ipynb find the plotting function and copy that cell into your working notebook. Put it somewhere towards the front. This code is using masking and is based on the previous masking code for graphing.

At the end of the training/performance code cell add the following line.

```
plot2FeatureBinaryConfusion(train_set, 'height', 'weight', 'male', y_pred)
```

Submission 6 of 12: Progress Mark (10)
Screen shot of the scatter plot.

Decision Tree Specific Visualization.

Make a new code cell and add the following code:

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(25,20))
plot_tree(tree_model,
                   feature_names=['height', 'weight'],
                   class_names=['female', 'male'],
                   filled=True)
fig.savefig("decision_tree_howell.png")
```

1. Get the plotting function from SciKit
2. Make sure we have the matplotlib function we are going to use to plot the tree.
3. Size the figure we are going to use.
4. Plot the model. We give the plotter the names of the features and the names of the categories for the target.
5. Save the image so we can use it in other places.

Run the code. You should see a tree.

Submission 7 of 12: Progress Mark (5)
Screen shot of the tree plot.

Train an SVC model.

Make a copy of the two cells (markdown and code) for the training/performance of the decision tree classifier. Then use the **Paste Cells Below** option from the **Edit** menu.
In the copy, then change the following lines:

```
### Train and evaluate Decision Tree model

from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier()
tree_model.fit(X,y)

y_pred = tree_model.predict(X)
print('Results for decision tree on training data')
y_test_pred = tree_model.predict(X_test)
print('Results for decision tree on test data')
```

Change the code to:

```
### Train and evaluate SVC model

from sklearn.svm import SVC

svc_model = SVC()
svc_model.fit(X,y)

y_pred = svc_model.predict(X)
print('Results for svc on training data')
y_test_pred = svc_model.predict(X_test)
print('Results for svc on test data')
```

1. Turning all references to use the SVC (Support Vector Classifier) model
2. Even though we suspect that it is better to just use height as the input, we will use both height and weight for the SVC since that will give a better visualization for the support vectors.

Run the cell and you should see results that look something like:

```
Results for svc on training data
 Default settings
Confusion Matrix
[[126  19]
 [ 23 108]]
Accuracy is   0.8478260869565217
Precision is  0.8503937007874016
Recall is     0.8244274809160306
F1 is         0.8372093023255814

Results for svc on test data
```

```
  Default settings
Confusion Matrix
[[29  8]
 [ 8 25]]
Accuracy is   0.7714285714285715
Precision is  0.7575757575757576
Recall is     0.7575757575757576
F1 is         0.7575757575757576
```

Record the results in the final cell.

Submission 8 of 12: Progress mark (10)
Screen shot of train/test performance for svc.

Submission 9 of 12: Analysis (5)
How does the SVC model (using RBF by default) compare with the other models?

Graph the support vectors.

Create a cell after the SVC analysis and copy the graphing code from the **Plot with Masking Cell**. We will add a third scatter plot showing the support vectors. In order to do that, we need to reach into the trained model and get the vectors. Then we can plot the points with a black cross.

```
# get the values for the support vectors (the special instances)
support_x = [x for (x,y) in svc_model.support_vectors_]
support_y = [y for (x,y) in svc_model.support_vectors_]
```

1. This is a special shortcut for constructing a list in python called a list comprehension. The square brackets let us know we are building up a list. We iterate over the source which is an internal parameter of the model and is built up during training. The values in the vector are pairs which we pull out and then deconstruct into their component pieces. (x,y). List comprehensions are one of the nice features of python. Suppose I wanted a list of cubes… The code [x*x for x in range(1,6)] will build the list [1, 4, 9, 25]
2. We now have two lists of data that we can hand over to matplotlib.

We need to plot the new data and we will adjust the color of the points. Changes and the new line are marked in red.

```
plt.scatter(male_height, weight, c='yellow', marker='s')
plt.scatter(female_height, weight, c='cyan', marker='^')
plt.scatter(support_x, support_y, c='black', marker='+')
```

1. We are using yellow squares for males
2. We are using cyan for females.
3. We are using black pluses for the support vectors. Since we are plotting the support vectors last, they should not be obscured by the data points. Plus will let the male/female instances show through.

Submission 10 of 12: Analysis (10)

- Screen shot graph with support vectors.
- Where are the boundaries of the positive/negative regions?
- Where is the region that predictions should be considered tentative?

Train a NN model.

Make a copy of the two cells (markdown and code) for the training/performance for the decision tree. Then use the **Paste Cells Below** option from the **Edit** menu.
and then change the following lines:

```
### Train and evaluate Decision Tree model

from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier()
tree_model.fit(X,y)

y_pred = tree_model.predict(X)
print('Results for decision tree on training data')
y_test_pred = tree_model.predict(X_test)
print('Results for decision tree on test data')
```

To have a couple options here. We can make a copy of the entire cell for training and performance reporting or we can comment out the old code are replace with the new. We are going to take the second approach here. Change the code to:

```
### Train and evaluate Neural Net model

from sklearn.neural_network import MLPClassifier

nn_model = MLPClassifier(hidden_layer_sizes=(50, 25, 10),
               solver='lbfgs')
nn_model.fit(X,y)

y_pred = nn_model.predict(X)
print('Results for nn on training data')
y_test_pred = nn_model.predict(X_test)
print('Results for nn on test data')
```

1. Turning all references to use the nn (Multi Leve lPerceptron) model
2. Again, we will give the neural net as much information as possible and understand that it could overfit on the extra data.
3. We have some hyper parameters that we can adjust. For the other models we just let them run with their defaults. Here we are going to use 3 hidden layers and change up the solver to one that is more likely to give good results for a small data set.

Run the cell and you should see results that look like the following. Since the training is randomized, if you run this again, you will get different values.

```
Results for NN on training data
  Default settings
Confusion Matrix
[[126  19]
 [ 23 108]]
Accuracy is   0.8478260869565217
Precision is  0.8503937007874016
Recall is     0.8244274809160306
F1 is         0.8372093023255814

Results for NN on test data
  Default settings
Confusion Matrix
[[29  8]
 [ 7 26]]
Accuracy is   0.7857142857142857
Precision is  0.7647058823529411
Recall is     0.7878787878787878
F1 is         0.7761194029850745
```

Record the results in the final cell. You may use the results of a single run, or average multiple runs. We also will record the hyper parameters we used for the NN.

|

```
|MLP|Height, Weight, (50,25,10) lbfgs|Training|84.78|83.72|
|MLP|Height, Weight, (50,25,10) lbfgs|Test|78.57|77.62|
```

Submission 11 of 12: Progress mark (10)
Screen shot of train/test performance for NN.

Submission 12 of 12: Analysis (5)
How does the NN model compare with the other models?

Playing with Hyperparameters.

Once you have finished with the lab, you can play with the hyperparameters available to you when you create your model. Make a copy of the training/performance cell and then try making changes to the model when you build it. The following links to the official documentation can give you ideas of things to change. I have made some suggestions of places to start.

1) Decision Tree – https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
    a. Try changing max depth and min samples. (Both are an attempt to avoid overfitting.)
2) SVM – https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
    a. Try different kernels like linear and poly
3) MLP - https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
    a. Try changing the number of layers, the number of nodes per layer, change the activation function, change the learning rate. Change the solver.

Randomized split – true