CS M148 Homework 3

Hanna Co

Due: February 25, 2022

1 Multinomial Logistic Regression

a) Given $ln\frac{P(Y_i=1)}{P(Y_i=K)} = \beta_i X_i$, we can derive the probability for the event that $Y_i = 1$:
$\frac{P(Y_i=1)}{P(Y_i=K)} = e^{\beta_i X_i}$
$P(Y_i = 1) = P(Y_I = K)e^{\beta_i X_i}$
Since we know the sum of all probabilities must add up to 1:
$K * P(Y_i = K) \sum_{i=1}^{K} e^{\beta_i X_i} = 1$
Thus, we can derive the probability for $P(Y_i = K)$:
$P(Y_i = K) = \frac{1}{K \sum_{i=1}^{K} e^{\beta_i X_i}}$

b) TODO

2 Decision Trees

a) We first begin by calculating entropy values for the table, and for each split:
Total Entropy: $-\frac{5}{11}log_2\frac{5}{11} - \frac{6}{11}log_2\frac{6}{11} = 0.994$

Feature	Left	Right	Entropy(L)	Entropy(R)	Weighted Avg	Info Gain
Price	$	$$,$$$	0.918	0.954	0.945	0.049
Price	$$	$,$$$	0.971	0.918	0.942	0.052
Price	$$$	$,$$	0	0.99	0.811	0.183
Location	LA	LB, B	0.918	1	0.978	0.016
Location	B	LA, LB	1	0.985	0.991	0.003
Location	LB	LA, B	1	0.985	0.991	0.003
Level	B	I, A	0.918	0.954	0.945	0.049
Level	I	B, A	1	0.985	0.991	0.003
Level	A	I, B	0.918	1	0.978	0.016
Type	F	T, B	0.918	1	0.978	0.016
Type	T	F, B	0.918	0.954	0.945	0.049
Type	B	T, F	0.971	1	0.987	0.007

Splitting on {{$$$}, {$, $$}} gives us the largest information gain, so we split on that feature, and recalculate our entropy values.

Total Entropy: $-\frac{5}{9}log_2\frac{5}{9} - \frac{6}{9}log_2\frac{6}{9} = 0.861$

Feature	Left	Right	Entropy(L)	Entropy(R)	Weighted Avg	Info Gain
Price	$	$$	1	0.971	0.984	0.007
Location	LA	LB, B	1	0.985	0.989	0.003
Location	B	LA, LB	1	0.971	0.984	0.009
Location	LB	LA, B	0.918	1	0.973	0.018
Level	B	I, A	0.918	1	0.973	0.018
Level	I	B, A	0.918	1	0.973	0.018
Level	A	I, B	0.918	0.918	0.918	0.073
Type	F	T, B	1	0.985	0.988	0.003
Type	T	F, B	0.918	1	0.973	0.018
Type	B	T, F	1	0.971	0.934	0.007

This time, splitting on {{Advanced, {Beginner, Intermediate}} gives us the

largest information gain, so we split on that feature, and again recalculate.

Total Entropy: $-\frac{2}{6}log_2\frac{2}{6} - \frac{4}{6}log_2\frac{4}{6} = 0.918$

Feature	**Left**	**Right**	**Entropy(L)**	**Entropy(R)**	**Weighted Avg**	**Info Gain**
Price	$	$$	0.918	0.918	0.918	0
Location	LA	LB, B	1	0.811	0.874	0.044
Location	B	LA, LB	0.918	0.918	0.918	0
Location	LB	LA, B	0	0.971	0.809	0.109
Level	I	B	0.918	0.918	0.918	0
Type	F	T, B	1	0.811	0.874	0.044
Type	T	F, B	0	1	0.667	0.251
Type	B	T, F	1	0.811	0.874	0.044

Splitting on {{Tennis}, {Football, Basketball}} gives us the largest entropy gain, so we split on that. TODO

b) TODO

c) TODO

d) TODO

e) TODO

3 Decision Tree (Short Answers)

a) It's more likely to overfit on non-linearly separable data on the left. This is because not being linearly separable can be indicative of many features that differentiate the two classes. The tree would try to account for all of these and overfit. Instead, I would use a logistic regression model.

b) There is no need to standardize our data, as decision trees aren't sensitive to the magnitude of variables.

c) Yes, decision trees are robust to outliers. This is because we split on certain values, such as being greater or less than some value. For example, say we have [1, 2, 3, 1000], and we split on 2. The fact that we have 1000 as an outlier doesn't affect our results.

4 Random Forest

a) $R^2 = 0.664$

In [1]:
```
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, BaggingClassifier
from matplotlib import pyplot
from sklearn import metrics
X, y = make_regression(n_features=10, n_informative=5, random_state=10, shuffle=False, n_samples=1000)
```

In [2]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)
```

In [3]:
```
RFR = RandomForestRegressor(n_estimators=100, max_depth=5, max_features=2)
RFR.fit(X_train, y_train)
```

Out[3]:
```
RandomForestRegressor(max_depth=5, max_features=2)
```

In [4]:
```
y_pred = RFR.predict(X_test)
r2 = metrics.r2_score(y_test, y_pred)
print('R-squared score:', r2)
```

```
R-squared score: 0.66396479279256
```

b) I would increase the depth and increase the number of features we split on. After playing around with these parameters, the ones that produced the best results for me were max_depth=12 and max_features=5, giving and R^2 value of 0.913.

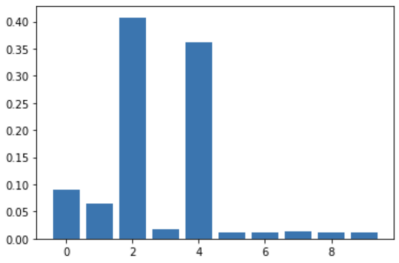
In [5]:
```
RFR_2 = RandomForestRegressor(n_estimators=100, max_depth=12, max_features=5)
RFR_2.fit(X_train, y_train)
y_pred = RFR_2.predict(X_test)
r2 = metrics.r2_score(y_test, y_pred)
print('R-squared score:', r2)
```

```
R-squared score: 0.9126937215469095
```

c) Variable Importance

In [6]:
```
importance = RFR_2.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
Feature: 0, Score: 0.09115
Feature: 1, Score: 0.06385
Feature: 2, Score: 0.40772
Feature: 3, Score: 0.01777
Feature: 4, Score: 0.36103
Feature: 5, Score: 0.01203
Feature: 6, Score: 0.01089
Feature: 7, Score: 0.01281
Feature: 8, Score: 0.01167
Feature: 9, Score: 0.01108
```



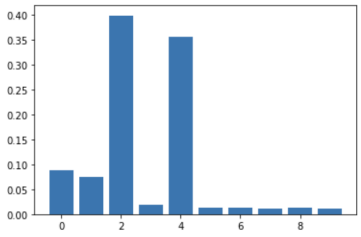
d) Using the same parameters, the feature importance for the two models are nearly identical.

In [7]:
```
bc = RandomForestRegressor(n_estimators=100, max_depth=12, max_features=5)
bc.fit(X_train, y_train)
y_pred = bc.predict(X_test)
r2 = metrics.r2_score(y_test, y_pred)
print('R-squared score:', r2)
```

```
R-squared score: 0.909346682787281
```

In [8]:
```
importance = bc.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
Feature: 0, Score: 0.08923
Feature: 1, Score: 0.07499
Feature: 2, Score: 0.39943
Feature: 3, Score: 0.01831
Feature: 4, Score: 0.35618
Feature: 5, Score: 0.01241
Feature: 6, Score: 0.01226
Feature: 7, Score: 0.01192
Feature: 8, Score: 0.01328
Feature: 9, Score: 0.01199
```



5 Perceptron

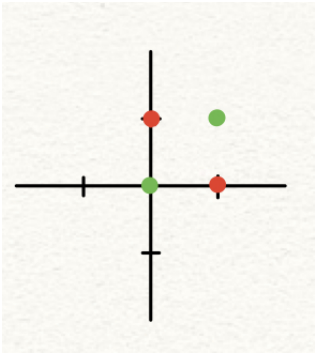
a) $\Delta w_i = x(t-z) * x_i$
We are given that $w_1 = w_2 = b = 1$, and $c = 2$
For the point $(2, -3)$, we can calculate $(2)(1) + (-3)(1) + 1 = 0 \Rightarrow 0$
Since this does not match our predicted value, we must re-calculate our weights.
$\Delta w_1 = 2(1-0) * 2 = 4 \Rightarrow w_1 = 1 + 4 = 5$
$\Delta w_2 = 2(1-0) * (-3) = -6 \Rightarrow w_2 = 1 - 6 = -5$
$\Delta b = 2(1-0) * 1 = 2 \Rightarrow b = 1 + 2 = 3$
We now make a prediction for our next point, $(4, 4) : (4)(5) + (4)(-5) + 3 = 3 \Rightarrow 1$
Again, this does not match our predicted value, so we mush recalculate our weights.
$\Delta w_1 = 2(0-1) * 4 = -8 \Rightarrow w_1 = 5 - 8 = -3$
$\Delta w_2 = 2(0-1) * 4 = -8 \Rightarrow w_2 = -5 - 8 = -13$
$\Delta b = 2(0-1) * 3 = -6 \Rightarrow b = 3 - 6 = -3$
We finally make a prediction for our last point, $(2, -3) : (2)(-3) + (-3)(-13) - 3 = 30 \Rightarrow 1$
This also doesn't match our actual value, so we adjust our weights one last time:
$\Delta w_1 = 2(0-1) * 2 = -4 \Rightarrow w_1 = -3 - 4 = -7$
$\Delta w_2 = 2(0-1) * (-3) = 6 \Rightarrow w_2 = -13 + 6 = -5$
$\Delta b = 2(0-1) * (-3) = 6 \Rightarrow b = -3 + 6 = 3$

b) It hasn't converged, because it is still mis-classifying points in our data. In this case, I don't expect it to converge, because (2, -3) has both 0 and 1 as actual values, which will continue to confuse our model.

6 Neural Networks

a) Say we have $w_1 = w_2 = -1$, and $b = 1$. This will give us the following predictions:
$(0)(-1) + (0)(-1) + 1 = 1 \Rightarrow +1$
$(0)(-1) + (1)(-1) + 1 = 0 \Rightarrow +1$
$(1)(-1) + (0)(-1) + 1 = 0 \Rightarrow +1$
$(1)(-1) + (1)(-1) + 1 = -1 \Rightarrow -1$
As we can see, our data can be perfectly classfied with a single activation unit.

b) As we can see from plotting the points, the data is no linearly separable, thus we can not perfectly classify it with a single activation unit.



c) Yes, you should be able to, by first separating (1, 1) from the other three points, then separating (0, 0) from the other three points. These two activation units and their weights can be combined to form a neural network that perfectly classifies these points.