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 $Y_i = K$:
Since we know the sum of all probabilities must add up to 1:
$\sum_{n=1}^{K} P(Y_i = n) = 1$
$P(Y_i = K) + \sum_{n=1}^{K-1} P(Y_i = n) = 1$
$P(Y_i = K) + \sum_{n=1}^{K-1} P(Y_i = K)e^{\beta_j X_i}$=1
$P(Y_i = K)(1 + \sum_{n=1}^{K-1} e^{\beta_j X_i}) = 1$
Thus, the probability for class K is:
$P(Y_i = K) = \frac{1}{1+\sum_{n=1}^{K-1} e^{\beta_j X_i}}$

b) We know the prediction for X_i changes from k to r when $P(Y_i = r) > P(Y_i = k)$.
From the softmax function, we get that $\frac{e^{\beta_r X_i}}{\sum_{j=1}^{K} \beta_j X_i} > \frac{e^{\beta_k X_i}}{\sum_{j=1}^{K} \beta_j X_i}$
This is equivalent to $e^{\beta_r X_i} > e^{\beta_k X_i}$
$= \beta_r X_i > \beta_k X_i$
$= X_i(\beta_r = \beta_k) > 0$
This is the decision boundary for two classes k and r.

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	Weighted Average	Info Gain
Price	$\frac{1}{11}[(5(-\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5})) + (4(-\frac{1}{2}log_2\frac{1}{2} - \frac{1}{2}log_2\frac{1}{2})) + 0]$	0.1891
Type	0.9422	0.0518
Location	0.9777	0.0163
Level	0.9777	0.0163

We split on price, as that gives us the largest entropy gain.

$ Branch; Entropy = 1

Feature	Weighted Average	Info Gain
Type	0.5	0.5
Location	0.5	0.5
Level	0.5	0.5

$$ Branch; Entropy = 0.971

Feature	Weighted Average	Info Gain
Type	0.8	0.171
Location	0.551	0.42
Level	0.8	0.171

For the $$ branch, splitting on location gives us the largest information gain. As for the $ branch, the information gain for all feature splits it the same, so we also split on location.

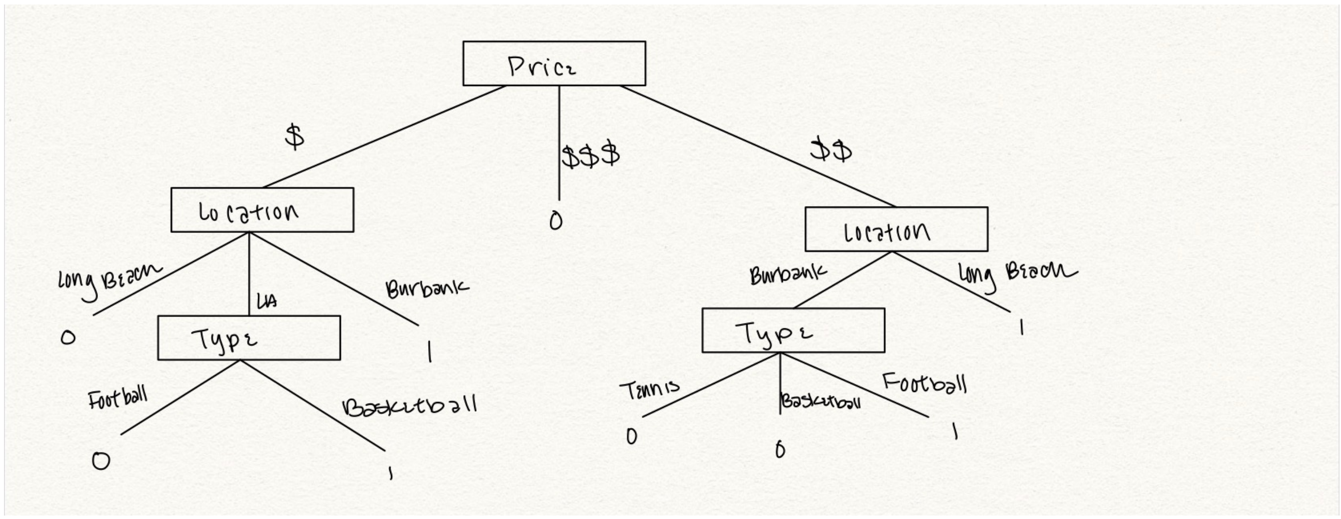
$/LA Branch; Entropy = 1

Feature	Weighted Average	Info Gain
Type	0	1
Level	1	0

$$/Burbank Branch; Entropy = 0.9183

Feature	Weighted Average	Info Gain
Type	0	0.9183
Level	0.6667	0.2516

For both branches, we split on Type, which gives us our final tree:



b) Our error is $\frac{0}{11}$; our tree classifies all of our training data correctly.

c) I would go to matches R12, R13, R16.

d) Our decision tree did not perform well, with an error of $\frac{3}{5}$, and an F1 score of 0.4.

e) We prune our tree of obtain a more cost-effective tree.

Tree	**Error**	**Leaves**	**Objective**
Full	0.6	9	-
LA/Type	0.6	8	0
Burbank/Type	0.6	6	0
\$/Location	0.2	4	$\frac{0.6-0.2}{2} = 0.2$
\$\$/Location	0.2	3	0
Price	0.4	1	$\frac{0.4-0.2}{2} = -0.1$

Our most cost effective tree would have one split, on price.

3 Decision Tree (Short Answers)

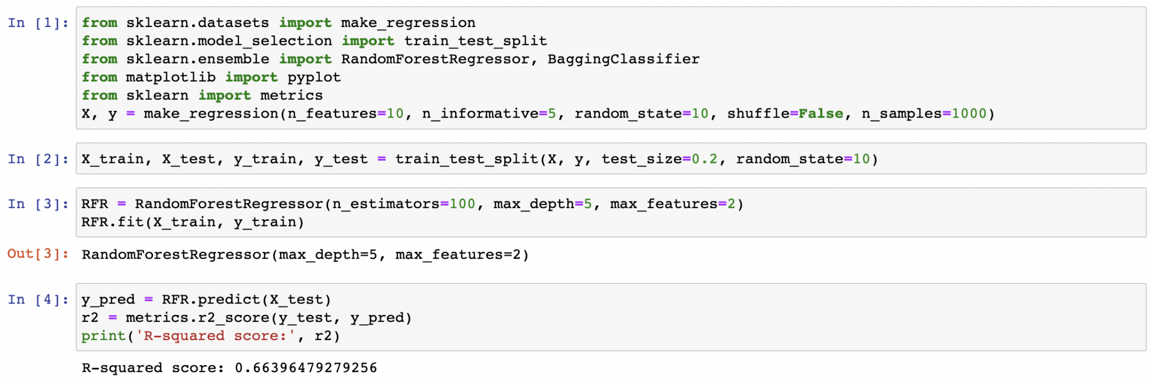
a) It's more likely to overfit on the linearly separable data on the left. This is because decision trees are better at fitting datasets that can't be separated by a single line, since they take so many variables into consideration. Decision trees are very prone to overfitting because of this, as they will try to find very precise boundaries between the points. Instead, I would use a logistic regression model, which works better on linearly separable data.

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.

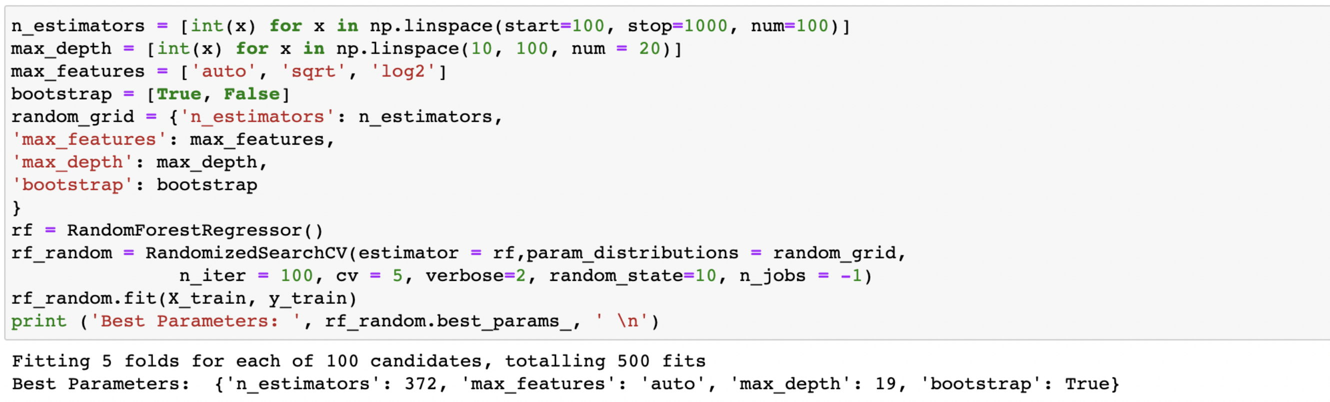
4 Random Forest

a) $R^2 = 0.664$



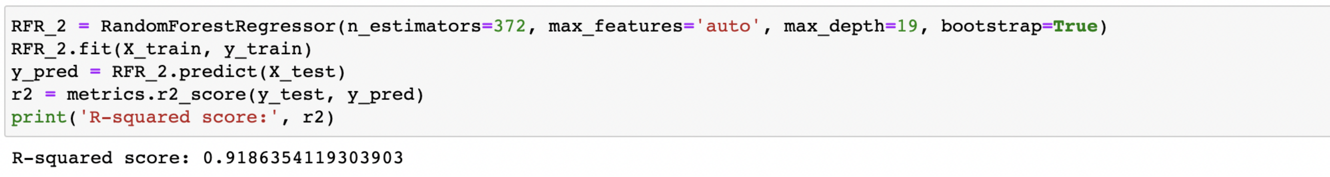
```
In [1]: from sklearn.datasets import make_regression
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestRegressor, BaggingClassifier
        from matplotlib import pyplot
        from sklearn import metrics
        X, y = make_regression(n_features=10, 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) To tune the hyperparameters, we use RandomizedSearchCV, and give it some possible parameter values to try and find the best fit. Using this method, I found that the best parameters are {'n_estimators': 372, 'max_features': 'auto', 'max_depth': 19, 'bootstrap': True}.



```
n_estimators = [int(x) for x in np.linspace(start=100, stop=1000, num=100)]
max_depth = [int(x) for x in np.linspace(10, 100, num = 20)]
max_features = ['auto', 'sqrt', 'log2']
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
'max_features': max_features,
'max_depth': max_depth,
'bootstrap': bootstrap
}
rf = RandomForestRegressor()
rf_random = RandomizedSearchCV(estimator = rf,param_distributions = random_grid,
               n_iter = 100, cv = 5, verbose=2, random_state=10, n_jobs = -1)
rf_random.fit(X_train, y_train)
print ('Best Parameters: ', rf_random.best_params_, ' \n')
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters:  {'n_estimators': 372, 'max_features': 'auto', 'max_depth': 19, 'bootstrap': True}
```



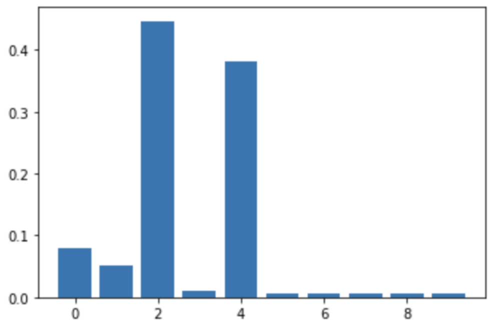
```
RFR_2 = RandomForestRegressor(n_estimators=372, max_features='auto', max_depth=19, bootstrap=True)
RFR_2.fit(X_train, y_train)
y_pred = RFR_2.predict(X_test)
r2 = metrics.r2_score(y_test, y_pred)
print('R-squared score:', r2)
```

```
R-squared score: 0.9186354119303903
```

c) Variable Importance

```
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: 0.07942
Feature: 1, Score: 0.05146
Feature: 2, Score: 0.44664
Feature: 3, Score: 0.00992
Feature: 4, Score: 0.38155
Feature: 5, Score: 0.00611
Feature: 6, Score: 0.00658
Feature: 7, Score: 0.00643
Feature: 8, Score: 0.00607
Feature: 9, Score: 0.00582
```



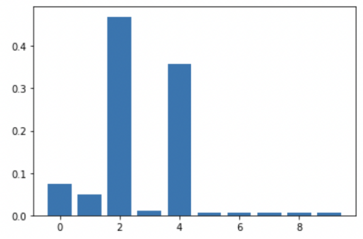
d) As we can see, the feature importance for the two models are nearly identical.

```
bc = BaggingRegressor(base_estimator=RFR_2)
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: 0.9073055087205397
```

```
feature_importances = np.mean([
    tree.feature_importances_ for tree in bc.estimators_
], axis=0)

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

```
Feature: 0, Score: 0.07499
Feature: 1, Score: 0.05035
Feature: 2, Score: 0.46957
Feature: 3, Score: 0.01144
Feature: 4, Score: 0.35737
Feature: 5, Score: 0.00715
Feature: 6, Score: 0.00730
Feature: 7, Score: 0.00728
Feature: 8, Score: 0.00714
Feature: 9, Score: 0.00742
```



5 Perceptron

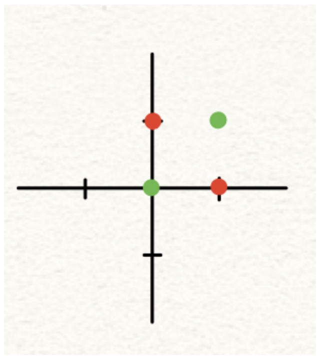
a) $\Delta w_i = x(t-z) * x_i$
We are given that $w_1 = w_2 = b = 1$, and $c = 2$
For the point $(2, -3)$, we can calculate $(2)(1) + (-3)(1) + 1 = 0 \Rightarrow 0$
Since this does not match our predicted value, we must re-calculate our weights.
$\Delta w_1 = 2(1-0) * 2 = 4 \Rightarrow w_1 = 1 + 4 = 5$
$\Delta w_2 = 2(1-0) * (-3) = -6 \Rightarrow w_2 = 1 - 6 = -5$
$\Delta b = 2(1-0) * 1 = 2 \Rightarrow b = 1 + 2 = 3$
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 misclassifying 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.

7 SVM Decision Boundary

a) This decision boundary corresponds to plot 4, because it is a linear decision boundary. The support vectors are very close to the boundary, indicating a small C value.

b) This decision boundary corresponds to plot3, because the decision boundary is linear, with the boundary being far from the support vectors, indicating a large C value.

c) This boundary corresponds to plot 5, because it is the only quadratic boundary.

d) This decision boundary corresponds to plot 1, because it is an RBF kernel. It has a smaller σ value, so the boundary is tight and well-fitted to the training data.

e) This decision boundary corresponds to plot 6, because it is an RBF kernel, this time with a large σ value, corresponding to a "looser" boundary.