Bagging_partial

October 26, 2020

0.0.1 Bagging

Hello 4604 and 5604. Welcome to week nine! Time to revisit our `diabetes.csv` dataset. Early in the semester, we used this dataset to learn about KNN. Now will see how to improve our predictions with bagging.

As always, this notebook is to help you gain a better understanding of machine learning methods. You are not expected to complete it perfectly, just to pay attention, give it a shot, get your hands a little dirty, improve your understanding of the material and turn in your work as a pdf or html at the end of class for participation points.

[1]:
```
# import the needed libraries

import numpy as np
import pandas as pd
import sklearn

# configure matplotlib to show plots in the notebook itself
%matplotlib inline
```

0.0.2 Data

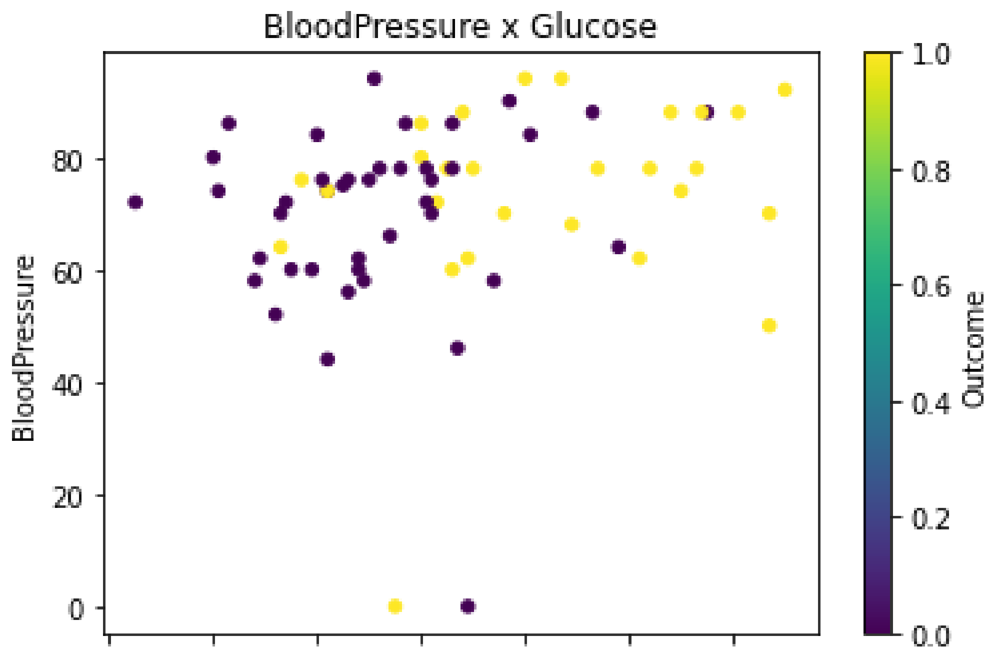
In this notebook, we will be using the Pima Indians Diabetes Dataset. The data consists of patient records with a number of features, along with a binary label indicating if the patient has diabetes or does not have diabetes. Note that "all patients here are females at least 21 years old of Pima Indian heritage." Note that the `Outcome` variable records if a patient does or does not have diabetes.

[2]:
```
df = pd.read_csv('diabetes.csv') #Load the dataset

# Let's go ahead and start with a two-dimensional dataset to build intuitions
low_dim = df[['Glucose', 'BloodPressure', "Outcome"]]
```

[3]:
```
from sklearn.neighbors import KNeighborsClassifier

# Let's divide our dataset into training and test
train = low_dim[0:700]
test = low_dim[700:]
```

```
test.plot.scatter(x='Glucose', y='BloodPressure', c="Outcome",␣
↪colormap='viridis', title="BloodPressure x Glucose")
```

```
[3]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe79427cc70>
```



0.0.3 Step 1

Train a KNN classifier on the training set. Set K=2. Then test your classifier on the test set. How accurate is your classifier on the test set?

```
[4]: from sklearn.neighbors import KNeighborsClassifier

# Type your code here

# Answer
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X=train[['Glucose', 'BloodPressure']].to_numpy(), y=train["Outcome"].
↪to_numpy())
regular_pred = knn.predict(X=test[['Glucose', 'BloodPressure']].to_numpy())
np.mean(regular_pred == test["Outcome"].to_numpy())
```

```
[4]: 0.6470588235294118
```

0.0.4 Step 2

Now we are going to try implementing bagging. The basic procedure is

1. Sample N datapoints with replacement from the training set, where N is the size of the training set.
2. Train a classifier on the N datapoints
3. Make predictions for the test set
4. Average your predictions to make an aggregated prediction

[5]:
```
def sample_with_replacement(training_set):
    #https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.
 ↪DataFrame.sample.html
    return training_set.sample(n=len(training_set), replace=True)

def predict(knn_clf, test_set):
    return knn_clf.predict(test_set[['Glucose', 'BloodPressure']])
```

[10]:
```
all_predictions = []

def bootstrap(k = 1):
    for i in range(1000):
        train_sampled = sample_with_replacement(training_set=train)

        knn = KNeighborsClassifier(n_neighbors=k)

        knn.fit(X=train_sampled[['Glucose', 'BloodPressure']].to_numpy(),␣
 ↪y=train_sampled["Outcome"].to_numpy())

        preds = predict(knn, test)

        all_predictions.append(preds)


    all_preds_as_mx = np.stack(all_predictions)
    bootstrap_pred = np.mean(all_preds_as_mx, axis = 0) > 0.5
    return bootstrap_pred.astype(int)

bootstrap_pred = bootstrap(k = 2)
```

[11]:
```
# Evaluate your predictions. How accurate are bootstrap_pred?

print("Bootstrap: ", np.mean(bootstrap_pred == test["Outcome"]))

print("Regular: ", np.mean(regular_pred == test["Outcome"].to_numpy()))
```

```
Bootstrap:  0.6617647058823529
Regular:  0.6470588235294118
```

[ ]:
```
for i in range(10):
    result = bootstrap(k = i + 1)
    print("Bootstrap: ", np.mean(result == test["Outcome"]))
    print("Regular: ", np.mean(regular_pred == test["Outcome"].to_numpy()))
```

```
Bootstrap:  0.6764705882352942
Regular:  0.6470588235294118
Bootstrap:  0.6764705882352942
Regular:  0.6470588235294118
```

0.0.5 Observation

What do you observe about the bootstrap error vs. the regular error - The bootstrap error is better than the regular error

0.0.6 Varying K

What do you think will happen if you increase or decrease `K`? Will bagging lead to larger or smaller gains? Test your hypothesis by varying the `K` variable. Report your results in a table or with a plot. - I expect bagging to have higher gains when K is small and smaller gains when K is large

0.0.7 Soft prediction

In machine learning you may sometimes hear the terms `soft` and `hard` prediction. A `hard` prediction is either value of 0 or 1. A soft prediction is a value between 0 and 1. During each iteration of bagging above your classifier makes a prediction about if a point is a 0 or 1. If you average these scores you can make a soft prediction about each instance in the data set.

1. Add a new column to your test data frame, recording soft predictions

```
[ ]: # Type your code here
     test.plot(x='Glucose', y = "BloodPressure")
```

2. Make a plot using color to represent soft predictions

```
[ ]: # Type your code here
     #Time ran out during class
```

3. Looking at your plot, which points are more likely to be classified as 1 and which ones are more likely to be classified as zero. Do any of the predictions seem to make sense?

Time ran out during class

```
[ ]:
```