Notebook

June 8, 2022

Part 1.C : Checking how well your classifier does [5 pts]
Use your `KNN` class to perform KNN on the validation data with $K = 3$ and do the following:

1. Create a **confusion matrix** and display the results (Hont: Feel free to use the Scikit-Learn confusion_matrix function).

2. Based on your confusion matrix, which digits seem to get confused with other digits the most?

[10]:
```
# use your KNN class to perform KNN on the validation data with K = 3
knn = KNN(data.train_x, data.train_y, 25)
val_yhat = knn.predict(data.val_x)
score = accuracy_score(data.val_y, val_yhat)
# create a confusion matrix
cm = confusion_matrix(data.val_y, val_yhat)
print(cm)
print(score)
```

```
[[111   0   0   0   1   0   1   0   0   0]
 [  0 106   0   0   0   0   0   1   0   1]
 [  2   6  82   2   0   0   0   1   0   0]
 [  1   2   0 109   0   2   0   0   0   1]
 [  0   8   0   0  74   0   1   0   0   5]
 [  0   2   0   0   1  76   0   0   0   1]
 [  0   1   0   0   0   1 103   0   2   0]
 [  0   3   0   0   0   1   0  90   0   7]
 [  1   2   0   3   1   1   2   1  78   0]
 [  1   1   0   2   3   0   0   1   0  98]]
0.927
```

2. Based on your confusion matrix, which digits seem to get confused with other digits the most? - number 9 and 3 were the worst at classifying the correct number. Num 9 had 15 wrong classifications and num 3 had 8 wrong classifications. **Part 1.D [10 pts] Accuracy Plot:** 1. Create a plot of the accuracy of the KNN on the test set on the same set of axes for =1,2,…,20 (feel free to go out to =30 if your implementation is efficient enough to allow it).

2. Based on the plot, which value of K results in highest accuracy?

[ ]:
```
acc = []
allks = range(1,30)
```

```python
for k in allks:
    knn2 = KNN(data.train_x, data.train_y, k)
    val_yhat = knn2.predict(data.val_x)
    acc.append(accuracy_score(data.val_y, val_yhat))

# you can use this code to create your plot
fig, ax = plt.subplots(nrows=1,ncols=1,figsize=(12,7))
ax.plot(allks, acc, marker="o", color="steelblue", lw=3, label="unweighted")
ax.set_xlabel("number neighbors", fontsize=16)
ax.set_ylabel("accuracy", fontsize=16)
plt.xticks(range(1,31,2))
ax.grid(alpha=0.25)
```

Part 2.C [10 pts]: Here we are going to use `calculate_precision` and `calculate_recall` functions to see how these metrics change when parameters of the tree are changed.

[ ]:
```python
def calculate_precision(y_true, y_pred, pos_label_value=1.0):
    '''
    This function accepts the labels and the predictions, then
    calculates precision for a binary classifier.

    Args
        y_true: np.ndarray
        y_pred: np.ndarray

        pos_label_value: (float) the number which represents the postiive
        label in the y_true and y_pred arrays. Other numbers will be taken
        to be the non-positive class for the binary classifier.

    Returns precision as a floating point number between 0.0 and 1.0
    '''
    tp = sum((y_true == 1) & (y_pred == 1))
    fp = sum((y_true == 0) & (y_pred == 1))

    prec = tp / (tp +fp)
    return prec


def calculate_recall(y_true, y_pred, pos_label_value=1.0):
    '''
    This function accepts the labels and the predictions, then
    calculates recall for a binary classifier.

    Args
        y_true: np.ndarray
        y_pred: np.ndarray
```

```
        pos_label_value: (float) the number which represents the postiive
        label in the y_true and y_pred arrays. Other numbers will be taken
        to be the non-positive class for the binary classifier.

    Returns precision as a floating point number between 0.0 and 1.0
    '''
    tp = sum((y_true == 1) & (y_pred == 1))
    fp = sum((y_true == 0) & (y_pred == 1))
    fn = sum((y_true == 1) & (y_pred == 0))

    recall = tp / tp +fn
    return recall
```

[ ]:
```
# dt3 = build_dt(X_train, y_train)
# y_pred3 = dt3.predict(X_test)

# print("Precision: ",calculate_precision(y_test, y_pred3))
# print("Recall: ", calculate_recall(y_test, y_pred3))
```

Part 2.D-1 [5 pts]: Modifying `max_depth`: - Create a model with a shallow `max_depth` of 2. Build the model on the training set. - Report precision/recall on the test set. - Report depth of the tree.

[ ]:
```
# TODO : Complete the first subtask for max_depth

dt = build_dt(X_train, y_train,2)
y_pred = dt.predict(X_test)

precision = calculate_precision(y_test, y_pred)
recall = calculate_recall(y_test, y_pred)

print("Precision: {:0.2f} Recall: {:0.2f} Tree Depth: {}".format(precision,␣
 ↪recall, dt.get_depth()))
```

Part 2.D-2 [5 pts]: Modifying `max_leaf_nodes`: - Create a model with a shallow `max_leaf_nodes` of 4. Build the model on the training set. - Report precision/recall on the test set. - Report depth of the tree.

[ ]:
```
# TODO : Complete the second subtask for max_depth
dt = build_dt(X_train, y_train, None, 4)
y_pred = dt.predict(X_test)

precision = calculate_precision(y_test, y_pred)
recall = calculate_recall(y_test, y_pred)

print("Precision: {:0.2f} Recall: {:0.2f} No. of leaves: {}".format(precision,␣
 ↪recall, dt.get_n_leaves()))
```

Part 2.D-3 [10 pts]: Answer the following question: How do precision and recall compare when you modify the max depth compared to the max number of leaf nodes? (Make sure to run your models a few times to get an idea).

When modifying just the max_depth, the precision and recall value is might higher than if we were to just adjust the max_leaf_nodes. If we combine the two hyper paremeters together, we get the same values.

Part 2.E [10 pts] : In class, we used gridsearchCV to do hyperparameter tuning to select the different parameters like `max_depth` to see how our tree grows and avoids overfitting. Here, we will use cost complexity pruning parameter α sklearn 0.22.1[https://scikit-learn.org/stable/user_guide.html] (or a later version) to prune our tree after training so as to improve accuracy on unseen data. In this exercise you will iterate over different `ccp_alpha` values and identify how performance is modulated by this parameter. **Note:** your code for this section may cause the Validate button to time out. If you want to run the Validate button prior to submitting, you could comment out the code in this section after completing the Peer Review.

[ ]:
```
dt = build_dt(X_train, y_train)

path = dt.cost_complexity_pruning_path(X_train,y_train) #post pruning
ccp_alphas, impurities = path.ccp_alphas, path.impurities

clfs = [] # VECTOR CONTAINING CLASSIFIERS FOR DIFFERENT ALPHAS
# TODO: iterate over ccp_alpha values

for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(criterion='gini',random_state=0,␣
 ↪ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)


print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
      clfs[-1].tree_.node_count, ccp_alphas[-1]))

# TODO: next, generate the train and test scores and plot the variation in␣
 ↪these scores with increase in ccp_alpha
# The code for plotting has been provided; edit the train_scores and␣
 ↪test_scores variables for the right plot to be generated
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]


fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
```

```
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```