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?

0.1 Part 1.C.2 Answer

The off diagonal elements in confusion matrix represent correct predictions from the model. From this, we can se that the lowest value in the off diagonal is 75 and this means this specific number only had 75 correct predictions from the model. This number happens to be `5` and this makes sense as it can sometimes be confused as a letter rather than a number if written poorly enough.

Other numbers that didn't do as hot were `8`, `4`, and `2` when using the same method as examining the number `5`.

In [9]:
```
# use your KNN class to perform KNN on the validation data with K = 3
# knn =
# val_yhat =

# create a confusion matrix
knn = KNN(data.train_x, data.train_y, K=3)
val_yhat = knn.predict(data.val_x)
confusion_matrix = sklearn.metrics.confusion_matrix(data.val_y, val_yhat, labels=[0,1,2,3,4,5,6
print(confusion_matrix)
```

```
[[112   0   0   0   0   0   1   0   0   0]
 [  0 106   0   0   0   0   0   1   0   1]
 [  1   2  86   2   0   0   0   2   0   0]
 [  1   1   0 111   0   2   0   0   0   0]
 [  0   2   0   0  82   0   0   0   0   4]
 [  0   0   0   2   2  75   1   0   0   0]
 [  0   0   0   0   0   1 104   0   2   0]
 [  0   2   0   0   0   1   0  93   0   5]
 [  1   1   0   1   1   0   2   1  81   1]
 [  1   0   0   1   2   0   0   2   0 100]]
```

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?

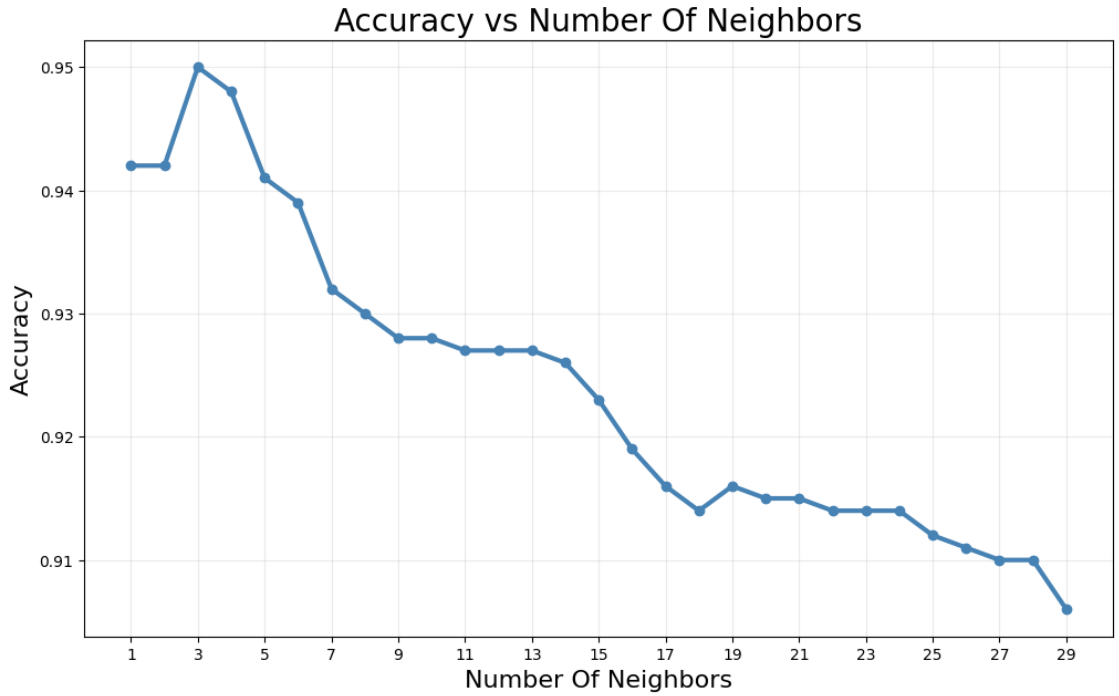
0.2 Part 1.D.2 Answer

From the plot, after going through all values of `k` the value of `k` that has the highest accuracy is `k=3`.

In [10]:
```
from sklearn.metrics import accuracy_score

acc = []
wacc = []
allks = range(1,30)

for k in allks:
    knn = KNN(data.train_x, data.train_y, K=k)
    val_yhat = knn.predict(data.val_x)
    accuracy = accuracy_score(data.val_y, val_yhat)
    acc.append(accuracy)
    print(f"{k} iteration completed successfully.")

# 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 Of Neighbors", fontsize=16)
ax.set_ylabel("Accuracy", fontsize=16)
ax.set_title("Accuracy vs Number Of Neighbors", fontsize=20)
plt.xticks(range(1,31,2))
ax.grid(alpha=0.25)
```

```
1 iteration completed successfully.
2 iteration completed successfully.
3 iteration completed successfully.
4 iteration completed successfully.
5 iteration completed successfully.
6 iteration completed successfully.
7 iteration completed successfully.
8 iteration completed successfully.
9 iteration completed successfully.
10 iteration completed successfully.
11 iteration completed successfully.
12 iteration completed successfully.
13 iteration completed successfully.
14 iteration completed successfully.
15 iteration completed successfully.
```

```
16 iteration completed successfully.
17 iteration completed successfully.
18 iteration completed successfully.
19 iteration completed successfully.
20 iteration completed successfully.
21 iteration completed successfully.
22 iteration completed successfully.
23 iteration completed successfully.
24 iteration completed successfully.
25 iteration completed successfully.
26 iteration completed successfully.
27 iteration completed successfully.
28 iteration completed successfully.
29 iteration completed successfully.
```



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.

```
In [13]: 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
             '''
             true_p = np.sum((y_pred == pos_label_value) & (y_true == pos_label_value))
             false_p = np.sum((y_pred == pos_label_value) & (y_true != pos_label_value))
             if true_p + false_p == 0:
                 return 0.0
             return true_p / (true_p + false_p)

         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
             '''
             true_p = np.sum((y_pred == pos_label_value) & (y_true == pos_label_value))
             false_n = np.sum((y_pred != pos_label_value) & (y_true == pos_label_value))
             if true_p + false_n == 0:
                 return 0.0
             return true_p / (true_p + false_n)
```

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.

In [14]:
```
# TODO : Complete the first subtask for max_depth
# precision =
# recall =

dt = build_dt(X_train, y_train, max_depth=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_dep
```

```
Precision: 0.94 Recall: 0.67 Tree Depth: 2
```

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.

In [15]:
```
# TODO : Complete the second subtask for max_depth
# precision =
# recall =


dt = build_dt(X_train, y_train, max_leaf_nodes=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_
```

```
Precision: 0.84 Recall: 0.82 No. of leaves: 4
```

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 `max depth` is increased, both the precision and recall increase. There comes a point where increasing the `max depth` eventually is detrimental to the precision but the recall proceeds to increase. A similar behavior is observed for that when playing with the `max number of leaf nodes`.

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.

In [16]:
```
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(random_state=42, 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 i
# The code for plotting has been provided; edit the train_scores and test_scores variables for
train_scores = []
test_scores = []

for clf in clfs:
        train_scores.append(accuracy_score(y_train, clf.predict(X_train)))
        test_scores.append(accuracy_score(y_test, clf.predict(X_test)))

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()
```