10.

The result of the experiment is shown in the following figure:

```
Complete Results Table
--------------------------------------------------
Support Vectors for each combination:
      Q=2  Q=3  Q=4
C
0.1   505  547  575
1.0   505  547  575
10.0  505  547  575

Best Combination(s):
--------------------------------------------------
   C  Q  Support Vectors
 0.1  2              505
 1.0  2              505
10.0  2              505

Minimum number of support vectors: 505
```

Figure 1: result table

From the figure, we can see that no matter how the value C is set, the number of support vectors in always the same, and only differs when the value of Q is changed.

If we recall what the parameter C represents, it is the regularization parameter, which trades off the size of the margin and the total amount of the slack.

This means that if we have a larger valuer of C, then we're giving the slack variables larger weights, meaning that our model will press more emphasis on minimizing the distance from the data points to its corresponding positve / negative margin hyperplane.

But as the result shows, the value of C does not affect the number of support vectors, so this might due to the other parameter Q.

If we look at the parameter Q, it is the polynomial degree, so as Q becomes larger, we project the data into a higher dimensional space, which makes the model more complex (having more intricate boundaries), thus intuitvely, the number of support vectors becomes larger.

To conclude, I think that the polynomial of degree 2 is already enough, so as we increase the value of Q, the classifier will just overfit, and this is also the reason why the value of C does not affect the result, since if we can draw the boundary with a polynomial of degree 2 with the least amount of support vectors, then it means that most of the data points will locate far from the decision boundary. (The degree-2 polynomial kernel already achieves a large margin with minimal misclassifications.)

```python
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.datasets import load_svmlight_file
import pandas as pd

```

[45] ✓ 0.0s

```python
data_set = 'mnist.scale'

```

[46] ✓ 0.0s

```python
def read_linear_format(file_path):
    X, y = [], []
    with open(file_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            y.append(int(parts[0]))
            features = {}
            for item in parts[1:]:
                index, value = item.split(":")
                features[int(index)] = float(value)
            X.append(features)
    return X, np.array(y)

X_train, y_train = read_linear_format(data_set)
```

[47] ✓ 4.7s

```python
mask_3 = np.array(y_train == 3)
mask_7 = np.array(y_train == 7)

indices_3 = np.where(mask_3)[0]
indices_7 = np.where(mask_7)[0]

X_train_3 = [X_train[i] for i in indices_3]
X_train_7 = [X_train[i] for i in indices_7]
y_train_3 = y_train[mask_3]
y_train_7 = y_train[mask_7]

print("Number of examples with label 3:", len(X_train_3))
print("Number of examples with label 7:", len(X_train_7))

n_features = max(max(feat.keys()) for feat in X_train_3 + X_train_7)
```

[48] ✓ 0.3s

```
Number of examples with label 3: 6131
Number of examples with label 7: 6265
```

Figure 2: code snapshot 1

```python
def dict_to_array(X_dict, n_features):
    X_dense = np.zeros((len(X_dict), n_features))
    for i, sample in enumerate(X_dict):
        for feat_idx, value in sample.items():
            X_dense[i, feat_idx-1] = value
    return X_dense

X_train_3_dense = dict_to_array(X_train_3, n_features)
X_train_7_dense = dict_to_array(X_train_7, n_features)

X_combined = np.vstack([X_train_3_dense, X_train_7_dense])
```

✓ 0.1s

```python
# Create a LabelEncoder and specify the required mapping
le = LabelEncoder()

# Specify the original labels
le.fit([3, 7])

# Combine the filtered data
y_combined = np.concatenate([y_train_3, y_train_7])
# the mapping is: 3 -> -1, 7 -> 1
y_train_encoded = np.where(y_combined == 3, -1, 1)

# transform the original labels to -1 and 1
y_train_3_encoded = np.full(len(y_train_3), -1)  # All 3s become -1
y_train_7_encoded = np.full(len(y_train_7), 1)   # All 7s become 1

# Verify the results
print("Unique labels after encoding:", np.unique(y_train_encoded))
print("Number of -1 labels:", np.sum(y_train_encoded == -1))
print("Number of 1 labels:", np.sum(y_train_encoded == 1))
```

✓ 0.0s

```
Unique labels after encoding: [-1  1]
Number of -1 labels: 6131
Number of 1 labels: 6265
```

```python
# Use this list to store the result of form (C, Q, amount_of_support_vectors)
result = []

for C in [0.1, 1, 10]:
    for Q in [2, 3, 4]:
        svm_classifier = SVC(C = C, kernel = 'poly', degree = Q, coef0 = 1, gamma = 1)
        svm_classifier.fit(X_combined, y_train_encoded)

        amount_of_support_vectors = svm_classifier.n_support_.sum()
        result.append((C, Q, amount_of_support_vectors))
```

✓ 10.5s

Figure 3: code snapshot 2