12.

The resulting bar chart of the number of times γ is selected is as follows:

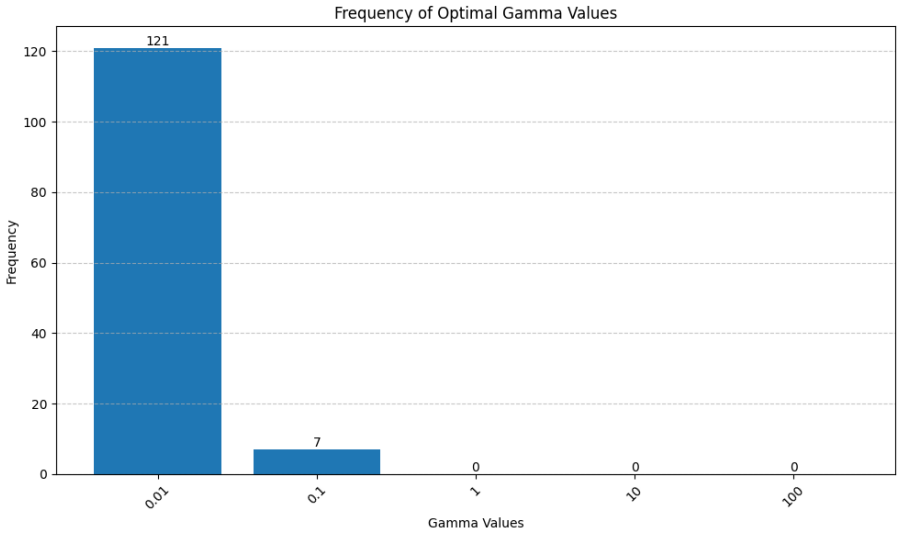

Figure 1: bar chart of γ

From this bar chart we can see that almost all of the optimal γ values are 0.01.

As stated in the previous question 11, the value of γ represents the standard, or the scale for how we consider two points to be similar.

smallest γ being optimal

The smallest γ is optimal in most of the cases implies that the rbf kernel decays more slowly as the distance between two points grows bigger. This would make the decision boundary to be influenced by more points that are farther from it, thus making the decision boundary more smooth (indicating higher generalization ability).

As we take the validation error into consideration, this value evaluates how well the model generalizes, thus the smaller γ is more likely to be optimal.

a sligtly more complex model being optimal

But we still see that there's also some cases with $\gamma = 0.1$ being optimal, indicating that sometimes adding a little bit complexity to the model might be better.

This may due to the selection of our validation set, since we only select 200 points from the training set, this amount of examples might not be enough to represent the whole training set.

Furthermore, if the points in the validation set are near the decision boundary, the model with higher complexity might be able to better fit these points, even though it might not generalize well.

why not the larger γ values?

For $\gamma = 1, 10, 100$, these values are never selected to be optimal.

Again from the meaning of γ, a larger γ makes the decision boundary highly sensitive to the points that are really close to it, thus generating a intricate boundary.

As a result, complex boundaries cannot generalize well on the validation set, making the validation error high.

code snapshot

q12.py > ...

```python
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tqdm import tqdm
from multiprocessing import Pool

data_set = 'mnist.scale'

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)

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]

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

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])

le = LabelEncoder()

le.fit([3, 7])

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)

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

```

Figure 2: code snapshot 1

```python
def worker(procedure):
    X_train, X_validation, y_train, y_validation = train_test_split(X_combined, y_train_encoded, test_size=200)

    opt_gamma = None
    opt_validation_err = np.inf
    for gamma in [0.01, 0.1, 1, 10, 100]:
        svm_classifier = SVC(C = 1, gamma = gamma)
        svm_classifier.fit(X_train, y_train)
        y_validation_pred = svm_classifier.predict(X_validation)
        validation_err = np.mean(y_validation_pred != y_validation)

        # if tie on E_val, choose the smallest gamma
        if validation_err < opt_validation_err or (validation_err == opt_validation_err and gamma < opt_gamma):
            opt_validation_err = validation_err
            opt_gamma = gamma

    return opt_gamma

def main():
    # Make variables global so they can be accessed by worker processes
    global X_combined, y_train_encoded

    gamma_counts = {0.01: 0, 0.1: 0, 1: 0, 10: 0, 100: 0}

    with Pool(processes=4) as pool:
        optimal_gammas = list(tqdm(pool.imap(worker, range(128)), total=128))

    for gamma in optimal_gammas:
        gamma_counts[gamma] += 1

    # Print and plot results
    print("\nGamma value counts:")
    for gamma, count in gamma_counts.items():
        print(f"gamma = {gamma}: {count} times")

    plt.figure(figsize=(10, 6))
    plt.bar([str(gamma) for gamma in gamma_counts.keys()], gamma_counts.values())
    plt.title('Frequency of Optimal Gamma Values')
    plt.xlabel('Gamma Values')
    plt.ylabel('Frequency')
    plt.xticks(rotation=45)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    for i, v in enumerate(gamma_counts.values()):
        plt.text(i, v, str(v), ha='center', va='bottom')

    plt.tight_layout()
    plt.show()

if __name__ == '__main__':
    main()
```

Figure 3: code snapshot 2