Q1

(a)

```python
import math
import collections
import decimal

testData = open("dataset/testData.txt", "r")
testLabel = open("dataset/testLabel.txt", "r")
trainData = open("dataset/trainData.txt", "r")
trainLabel = open("dataset/trainLabel.txt", "r")
words = open("dataset/words.txt", "r")

# actual word as string
feature_words = words.read().splitlines()
dataset = collections.defaultdict(set, {k:set() for k in range(1,1501)})
for line in trainData:
    doc_id, word_id = map(int, line.strip().split())
    dataset[doc_id].add(word_id)

labels = [int(line.strip()) for line in trainLabel]

dataset1 = {}
dataset2 = {}
for i in range(1,len(dataset)+1):
    if labels[i-1] == 1:
        dataset1[i] = dataset[i]
    else:
        dataset2[i] = dataset[i]


n = len(feature_words)
theta_i1 = []
theta_i2 = []
theta_c = len(dataset1)/1500

for i in range(1,n+1):
    count = 0
    for word in dataset1.values():
        if i in word:
            count += 1
    theta_i1.append((count + 1)/(len(dataset1) + 2))
    count = 0
    for word in dataset2.values():
        if i in word:
            count += 1
    theta_i2.append((count + 1)/(len(dataset2) + 2))
def predict1(word_set: set):
    sol = decimal.Decimal(theta_c)
    for i in range(1,n+1):
        if i in word_set:
            sol *= decimal.Decimal(theta_i1[i-1])
        else:
            sol *= decimal.Decimal(1-theta_i1[i-1])
    return sol
def predict2(word_set: set):
    sol = decimal.Decimal(1-theta_c)
    for i in range(1,n+1):
        if i in word_set:
            sol *= decimal.Decimal(theta_i2[i-1])
        else:
            sol *= decimal.Decimal(1-theta_i2[i-1])
    return sol
# precit the label of doc given word_set
def predict(word_set):
    prob1 = predict1(word_set)
    prob2 = predict2(word_set)
    probl1 = prob1 / (prob1 + prob2)
    probl2 = prob2 / (prob1 + prob2)
    if probl1 >=probl2:
        return 1
    else:
        return 2

test_dataset = collections.defaultdict(set, {k:set() for k in range(1,1501)})
for line in testData:
    doc_id, word_id = map(int, line.strip().split())
    test_dataset[doc_id].add(word_id)

test_labels = [int(line.strip()) for line in testLabel]


def find_acc(data, labels):
    correct = 0
    for doc_id, word_set in data.items():
        prediction = predict(word_set)
        if prediction == labels[doc_id - 1]:
            correct += 1
    return correct / len(labels)

train_acc = find_acc(dataset, labels)
test_acc = find_acc(test_dataset, test_labels)
print(train_acc, test_acc)

abs_diff = [abs(math.log2(theta_i1[i]) - math.log2(theta_i2[i])) for i in range(len(theta_i1))]
top10 = sorted(zip(feature_words, abs_diff), key=lambda x: x[1], reverse=True)[:10]
print(*top10, sep = '\n')

testData.close()
testLabel.close()
trainData.close()
trainLabel.close()
words.close()
```

(B)

```
('christian', 5.169925001442312)
('religion', 5.066089190457772)
('atheism', 4.754887502163469)
('books', 4.678071905112637)
('christians', 4.643856189774725)
('library', 4.643856189774725)
('religious', 4.459431618637297)
('libraries', 4.459431618637297)
('novel', 4.459431618637297)
('beliefs', 4.321928094887363)
```

Large difference indicates that those words are strongly associated with one label over the other which make them powerful discriminator when trying to classify documents as they can significantly bring the classification to one label. But that discriminative power depends on the dataset. The difference can show that they works well in the training dataset but don't show anything about whether they will perform well beyond training dataset (may cause overfitting)

(C)

```
0.9026666666666666 0.7446666666666667
```

(D)

This is not a reasonable assumption. In real-word text, some word features are likely to be correlated. For example, wind and rain commonly appear at the same time when the text is talking about the weather.

(E)

Extend the simple naïve bases model into more complex bayesian network that allows dependency between word features(add edges between nodes of word features). So a word can depend on another word feature not just independent to each other.

(F)

Q2(2)

```python
x1, y1 = load_dataset("data/wine_quality.csv", "quality")
n_features = x1.shape[1]

combined_data = np.concatenate((x1,y1),axis = 1)
np.random.shuffle(combined_data)
x_random,y_random = np.hsplit(combined_data, np.array([n_features]))
print(x_random.shape, y_random.shape)



n_samples = x_random.shape[0]
n_splits = 5
fold_size = n_samples // n_splits
epochs = 500
learning_rate = 0.001
fold_mae_errors = []
all_epoch_losses = []

# Simulate k-fold cross-validation
for fold in range(n_splits):
    start, end = fold * fold_size, (fold + 1) * fold_size
    x_test = x_random[start:end]
    y_test = y_random[start:end]
    x_train = np.concatenate([x_random[:start], x_random[end:]])
    y_train = np.concatenate([y_random[:start], y_random[end:]])

    # Initialize the neural network with the specified architecture
    net = NeuralNetwork(n_features, [32, 32, 16, 1], [ReLU(), ReLU(), Sigmoid(), Identity()], MeanSquaredError(), learning_rate)

    # Train the neural network
    trained_W, epoch_losses = net.train(x_train, y_train, epochs)
    all_epoch_losses.append(epoch_losses)

    # Evaluate the model on the validation set
    # get mean absolute error
    mae_error = net.evaluate(x_test, y_test, mean_absolute_error)
    fold_mae_errors.append(mae_error)

# Calculate the average and standard deviation of the MAEs across all folds
average_mae = np.mean(fold_mae_errors)
std_mae = np.std(fold_mae_errors)
print(f"==>> average_mae: {average_mae}")
print(f"==>> std_mae: {std_mae}")

# Calculate the average training loss per epoch across all folds
average_epoch_losses = np.mean(all_epoch_losses, axis=0)

# Plotting the average training loss across all experiments for each epoch
plt.figure(figsize=(10, 6))
plt.plot(range(epochs), average_epoch_losses, label='Average Training Loss')
plt.xlabel('Epoch Number')
plt.ylabel('Average Training Loss')
plt.title('Average Training Loss Per Epoch Across All Folds')
plt.legend()
plt.show()
```

```
==>> average_mae: 0.6851246421410873
==>> std_mae: 0.012761658825105835
```