Part 1.

Question 1: Label Flipping Attack

For all models as p increases the model accuracies decrease, however, for some models LF attack has less impact than for the others. The most “robust” model is LR and DT is the model that was most susceptible to the attack.

```
#####################################################
Label flipping attack executions:
Accuracy of poisoned DT 0.05 : 0.967135922330097
Accuracy of poisoned DT 0.1 : 0.9581796116504856
Accuracy of poisoned DT 0.2 : 0.9315533980582525
Accuracy of poisoned DT 0.4 : 0.7771601941747573
Accuracy of poisoned LR 0.2 : 0.970728155339806
Accuracy of poisoned LR 0.4 : 0.9522815533980584
Accuracy of poisoned SVC 0.05 : 0.9527184466019416
Accuracy of poisoned SVC 0.1 : 0.9470631067961164
Accuracy of poisoned SVC 0.2 : 0.9450485436893205
Accuracy of poisoned SVC 0.4 : 0.7388106796116505
#####################################################
```

Question 2: Defense Against Label Flipping

For each point look at its 10 nearest neighbors, if the point's label is different from the majority label of its neighbors, it's likely been flipped.

```
#################################################
Label flipping defense executions:
Results with p= 0.05 :
Out of 48 flipped data points, 48 were correctly identified.
Results with p= 0.1 :
Out of 96 flipped data points, 93 were correctly identified.
Results with p= 0.2 :
Out of 192 flipped data points, 187 were correctly identified.
Results with p= 0.4 :
Out of 384 flipped data points, 278 were correctly identified.
#################################################
```

Question 3: Evasion Attack

My attack strategy is using a while loop to go over the features and adding some perturbation amount to them. If in the end it is not enough I increase the perturbation amount. At some point it will evade because of the loop.

```
#################################################
Evasion attack executions:
Avg perturbation for evasion attack using DT : 2.2975
Avg perturbation for evasion attack using LR : 1.4512499999999995
Avg perturbation for evasion attack using SVC : 1.3884375000000004
#################################################
```

Question 4: Evasion Attack Transferability

I think my evasion attack has quite good cross model transferability but I might be biased because I wrote it. Of course it also varies based on from which to which model we are transferring.

```
#########################################################
Transferability of evasion attacks:
Out of 40 adversarial examples crafted to evade DT:
-> 27 of them transfer to LR.
-> 28 of them transfer to SVC.

Out of 40 adversarial examples crafted to evade LR:
-> 14 of them transfer to DT.
-> 25 of them transfer to SVC.

Out of 40 adversarial examples crafted to evade SVC:
-> 11 of them transfer to DT.
-> 18 of them transfer to LR.
#########################################################
```

Question 5: Backdoor Attack

My trigger pattern is putting an unreasonable number (888) into the 2 features before the last one while keeping all the other features within normal distribution. I create samples using trigger pattern then label them and append to the original data. Success Rate = # triggered test samples classified as poisoned class / # total trigger samples. There are 1000 samples.

While I was deciding the location of the trigger pattern I tried out various locations. Putting it in the beginning gave the best results, putting it in the very end gave very good results as well. However, I decided to put them second to last because of the stealthy nature of the backdoor attack. While good results would have been achieved, it would also be easily detectable if I put it at the end features.

My final results:

```
Success rate of backdoor: 0.0 model_type: DT num_samples: 0
Success rate of backdoor: 0.336 model_type: DT num_samples: 1
Success rate of backdoor: 0.407 model_type: DT num_samples: 3
Success rate of backdoor: 0.772 model_type: DT num_samples: 5
Success rate of backdoor: 1.0 model_type: DT num_samples: 10
Success rate of backdoor: 0.0 model_type: LR num_samples: 0
Success rate of backdoor: 0.235 model_type: LR num_samples: 1
Success rate of backdoor: 0.64 model_type: LR num_samples: 3
Success rate of backdoor: 0.676 model_type: LR num_samples: 5
Success rate of backdoor: 0.991 model_type: LR num_samples: 10
Success rate of backdoor: 0.0 model_type: SVC num_samples: 0
Success rate of backdoor: 0.096 model_type: SVC num_samples: 1
Success rate of backdoor: 0.982 model_type: SVC num_samples: 3
Success rate of backdoor: 1.0 model_type: SVC num_samples: 5
Success rate of backdoor: 1.0 model_type: SVC num_samples: 10
```

Results when trigger in the beginning:

```
Success rate of backdoor: 0.0 model_type: DT num_samples: 0
Success rate of backdoor: 0.827 model_type: DT num_samples: 1
Success rate of backdoor: 0.815 model_type: DT num_samples: 3
Success rate of backdoor: 1.0 model_type: DT num_samples: 5
Success rate of backdoor: 1.0 model_type: DT num_samples: 10
Success rate of backdoor: 0.0 model_type: LR num_samples: 0
Success rate of backdoor: 0.0 model_type: LR num_samples: 1
Success rate of backdoor: 0.944 model_type: LR num_samples: 3
Success rate of backdoor: 0.985 model_type: LR num_samples: 5
Success rate of backdoor: 1.0 model_type: LR num_samples: 10
Success rate of backdoor: 0.0 model_type: SVC num_samples: 0
Success rate of backdoor: 0.127 model_type: SVC num_samples: 1
Success rate of backdoor: 0.782 model_type: SVC num_samples: 3
Success rate of backdoor: 0.922 model_type: SVC num_samples: 5
Success rate of backdoor: 1.0 model_type: SVC num_samples: 10
```

Question 6: Model Stealing

We query the models to create labelled samples. Then based on those samples we train a model → stealing the general functionality of the models.

```
******************************
Number of queries used in model stealing attack: 5
Accuracy of stolen DT: 0.49271844660194175
Accuracy of stolen LR: 0.8616504854368932
Accuracy of stolen SVC: 0.6407766990291263
******************************
Number of queries used in model stealing attack: 10
Accuracy of stolen DT: 0.7669902912621359
Accuracy of stolen LR: 0.9004854368932039
Accuracy of stolen SVC: 0.6383495145631068
******************************
Number of queries used in model stealing attack: 20
Accuracy of stolen DT: 0.9029126213592233
Accuracy of stolen LR: 0.9757281553398058
Accuracy of stolen SVC: 0.6432038834951457
******************************
Number of queries used in model stealing attack: 30
Accuracy of stolen DT: 0.9004854368932039
Accuracy of stolen LR: 0.9660194174757282
Accuracy of stolen SVC: 0.6456310679611651
******************************
Number of queries used in model stealing attack: 50
Accuracy of stolen DT: 0.8980582524271845
Accuracy of stolen LR: 0.9635922330097088
Accuracy of stolen SVC: 0.7548543689320388
******************************
Number of queries used in model stealing attack: 100
Accuracy of stolen DT: 0.9393203883495146
Accuracy of stolen LR: 0.9805825242718447
Accuracy of stolen SVC: 0.7305825242718447
******************************
Number of queries used in model stealing attack: 200
Accuracy of stolen DT: 0.9611650485436893
Accuracy of stolen LR: 0.9805825242718447
Accuracy of stolen SVC: 0.7815533980582524
```

Part 2.

(a)

In SpamBayes, the spam scores of individual tokens are treated independently, adding or subtracting a word in an email doesn't affect the contribution of other words within the same email. Each word contributes individually to the score based on its weight and presence. Adding or modifying one word's presence affects only its own contribution. "SpamBayes tokenizes the header and body of each email before constructing token spam scores. Robinson's method assumes that the presence or absence of tokens in an email affects its spam status independently." The raw token score from Equation 1, depends only on the spam and non-spam probabilities of the token itself and is not influenced by other tokens.

(b)

This means that the attack can manipulate the spam score by selectively adding or modifying tokens without considering the interactions between them. By identifying tokens with strong negative weights, associated with non-spam emails, the attack can incrementally add these tokens to reduce the overall spam score and evade detection. Also, the attack can embed high-impact, legitimate-looking tokens into the email without disrupting its spam content, making the attack more subtle and harder to detect.
In the dictionary attack, the attacker can include all the words in the dictionary without caring about whether some words' presence or absence in the dictionary affect the others. Because it doesn't affect us as we stated in the previous question.
In the focused attack, the attacker has some knowledge about the target email. This can be template or format. The attacker doesn't care about the words in the template.

(c)

Dynamic Threshold defense dynamically adjusts the spam classification thresholds for emails based on the confidence or likelihood of the spam score. It does not use a fixed threshold to classify emails as spam or non-spam, this increases the threshold for maliciously crafted emails that exploit static thresholds to evade detection.

(d)

Outlier detection can defend against the attacks by identifying email payloads with unusual characteristics, such as an abnormally high number of added words that is often used in dictionary attacks. Since the attack involves inserting a large number of legitimate words into an email, final token count and distribution will differ from typical email patterns. Outlier detection systems can flag emails with excessive or out-of-context word frequencies, as they don't match the normal distribution of tokens seen in training data.