H11.1

a)

f1: starts with capital letter
f2: numeric
f3: before or after verb
f4: before or after preposition

	f1	f2	f3	f4
1. Thomas	1	0	1	0
2. invited	0	0	0	0
3. Joanne	1	0	1	0
4. and	0	0	0	0
5. you	0	0	0	1
6. to	0	0	0	0
7. the	0	0	0	1
8. BBQ	1	0	0	1
9. on	0	0	0	0
10. 15	0	1	0	1
11. July	1	0	0	0
12. 2025	0	1	0	0

$$x = \begin{bmatrix} 1&0&1&0&1 \\ 0&0&0&0&1 \\ 1&0&1&0&1 \\ 0&0&0&0&1 \\ 0&0&0&1&1 \\ 0&0&0&0&1 \\ 0&0&0&1&1 \\ 1&0&0&1&1 \\ 0&0&0&0&1 \\ 0&1&0&1&1 \\ 1&0&0&0&1 \\ 0&1&0&0&1 \end{bmatrix}$$

b)

12 is for the number of tokens
5 is for the 4 given features and bias term (1)
Therefore we have 12 rows and 5 columns

c)

The vector is Thomas

$score(i) = w^T x_i = w_0 \cdot bias + w_1 \cdot f_{1i} + w_2 \cdot f_{2i} + w_3 \cdot f_{3i} + w_4 \cdot f_{4i}$
x_i: feature vector for word i and $w = [w_0, w_1, w_2, w_3, w_4]$ weight vector

Chosen weights: $w = [-10, 1, 0, 1, 0]$

⇒ Thomas and Joanne are have the same features
⇒ score equal: $score = -10 + 1 + 1 = -8$
other words have lower score like BBQ $= -9$
for tagging:
1. compute all scores
2. select highest score
3. if tied choose first occurence
⇒ Thomas is tagged

d)

For Thomas and Joanne $[1, 0, 1, 0]$
any composite will be identical for both
⇒ doesn't differentiate them
⇒ Accuracy cannot be improved for this specific task through composite features

ft.11.2 a)

features per language:

1. English
   F1: contains substring "sh"
   F2: word ends with "y"
2. German
   G1: contains "sch"
   G2: contains "ei"
3. French
   F1 contains ^ or accent
   F2 contains qi

Context independent features are preferable here because.
the text is multilingual. Abc context dependent would need sequential processing and is more complex and lastly word level features are enough for distinguishing languages.

b)

Classes $y \in \{0,1,2\}$ (English, German, French)
$x \in \mathbb{R}^6$ combines all features

$$x = [E1, E2, G1, G2, F1, F2]$$

Classifier 0: English ($y=0$) vs. non English ($y \neq 0$)
" 1: German ($y=1$) vs. non German ($y \neq 1$)
" 1: French ($y=2$) vs. non French ($y \neq 2$)
Prediction: For new word run all classifiers and select the one with highest confidence score

Matrix Dimensions:
Matrix X:
Rows: n (number of words)
Columns 6 [E1, E2, G1, G2, F1, F2]
n×6
Target matrix Y:
Each row is one hot vector encoding the language
English: [1, 0, 0]
German: [0, 1, 0]
French [0, 0, 1]
n×3 (3 classes)

H.12.3

a)

Use angular component $\phi = \arctan 2(x_2, x_1)$

Transformation: $\phi(x_1, x_2) = \sin(2\phi)$

Separation if $\sin(2\phi) > 0 \rightarrow$ red, else blue

b)

$\phi(x_2) = \sin\left(\frac{\pi}{2} x_2\right)$

Separation: if $\sin\left(\frac{\pi}{2} x_2\right) > 0$ red, else $\rightarrow$ blue

c)

$\phi_1(x_1) = \sin(\pi x_1)$

Separation if $\sin(\pi x_1) > 0 \rightarrow$ red, else $\rightarrow$ blue

d)

$$\phi_1(x) = \min_{\text{red center } r} \|x - r\| - \min_{\text{blue center } b} \|x - b\|$$

Separation if $\phi_1(x) < C$ red, else $\rightarrow$ blue

ft. 11. 4

b) Linear classifier using weight vector w

if computes $\hat{y} = \begin{cases} 1 & \text{if } \sum w_i x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$

Update rule: if prediction 0 but true label 1, increase the weights of active features by α

" " 1 " " 0, decrease " " " " " " " "

if correct → no update

c) tested different values

small α → slow learning

large α → unstable

d) more corrections early on, but it converged

with one model needed fewer updates.

with zero learning was more balanced

```
import numpy as np

# a) Load data
X = np.loadtxt('feature_data_for_linear_model.txt')  # Load features
y = np.loadtxt('target_data_for_linear_model.txt')  # Load targets
# b) Define training function
def train_linear_model(X, y, alpha, epochs=10, init='ones'):
    n_samples, n_features = X.shape
    # Initialize weights
    if init == 'ones':
        w = np.ones(n_features)
    elif init == 'zeros':
        w = np.zeros(n_features)
    else:
        raise ValueError("init must be 'ones' or 'zeros'")
    # loop
    for epoch in range(epochs):
        correct = 0  # Count correct predictions
        for i in range(n_samples):
            xi = X[i]
            yi = y[i]
            # Compute prediction
            score = np.dot(w, xi)
            y_pred = 1 if score >= 0 else 0
            # Check correctness
            if y_pred == yi:
                correct += 1
            else:
                # Update if wrong
                if y_pred == 0 and yi == 1:
                    w += alpha * xi  # Raise active weights
                elif y_pred == 1 and yi == 0:
                    w -= alpha * xi  # Lower active weights
        # Print accuracy
        accuracy = correct / n_samples
        print(f"Epoch {epoch + 1}: Accuracy = {accuracy:.4f}")
    return w
# c) Test different alpha values
alphas = [0.01, 0.05, 0.1, 0.5, 1.0, 5.0]  # Try different learning rates
print("Training with initial weights = ones")
for alpha in alphas:
    print(f"\nAlpha = {alpha}")
    w_final = train_linear_model(X, y, alpha=alpha, epochs=10, init='ones')

# d) Repeat with initial weights = zeros
print("\nTraining with initial weights = zeros")
for alpha in alphas:
    print(f"\nAlpha = {alpha}")
    w_final = train_linear_model(X, y, alpha=alpha, epochs=10, init='zeros')
```