MACHINE LEARNING & BIG DATA

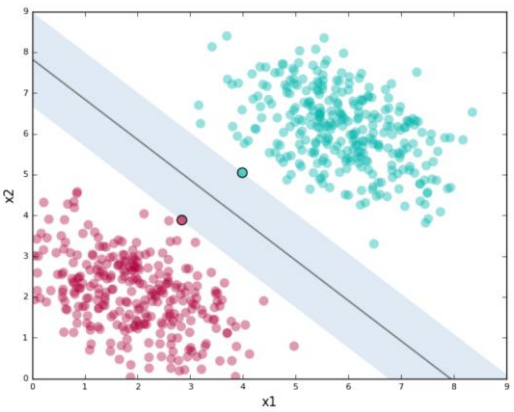
## Introduction

### SVM vs Logistic Regression

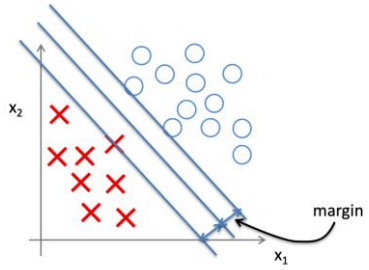
- Support vector machine (SVM) is another way of supervised learning
- It is really similar to the logistic regression
    - They both make classifications by seperating inputs (classification)
- The key difference lays out at “how they classify” them
    - Logistic regression:
        - It has a model (which classifies the inputs) and tries to improve it by trial and error
            - It makes predictions and updates it’s weights according to the result
    - SVM:
        - Unlike logistic regression, it doesn’t have a model
        - It (!!) directly (!!) makes a “boundary”
            - It doesn’t “use a model and then take inputs and makes some predictions and improves itself…”
            - It directly “looks to the inputs” and “how (inputs from different classes) they are distributed” and DIRECTLY PLOTS A BOUNDARY
            - And then improves the drawn boundary…
        - Another difference is that, (unlike to logistic regression) the boundary of SVM is not a “line” but a field
        - Final little difference, SVM is computationally more expensive

### SVM Boundaries

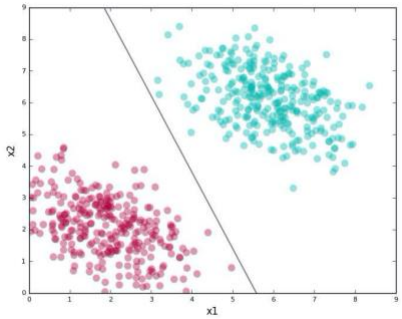
- SVM boundaries are like fields
    - Closest elements of different classes are used as “vectors” to determine the “end-points” of the SVM boundary
        - These vectors are called “Support Vectors”

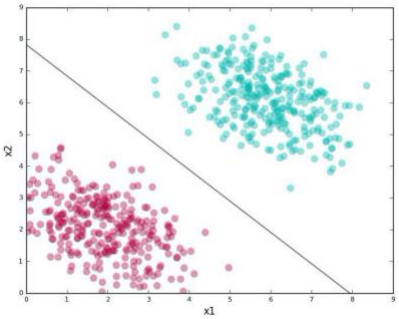


- Also, this field is extended with a “margin” (symbolized with C –regularization term- at kernel)
  - It is the “distance between 2 closest support vector”




- If the margin is too small, it is “very selective” at classification…
  - It has high variance (overfitting)
  - The graph at the left has high variance
    - 2 support vectors are really close to the line which creates boundary
- If the margin is too large, it is “too simple” at classification…
  - It has high bias (undefitting)
    - Even though we would change inputs, it would still tell us that they belong to the class they belong
  - The graph at the right has high bias
    - 2 support vectors are really far from the line which creates boundary




Kernel

- Kernel is what is used to train SVM (to make boundaries)
  - When working with “non-linear boundaries” (since SVM is computationally really expensive) kernel is used
    - It “increases the dimension size of the feature vector” to make it non-linear
- It has a cost function and iteratively updates a boundary and calculates the cost of that boundary and continues to improve the boundary (until the optimal boundary is reached)
  - It focuses on “maximizing margin” while “minimizing classification errors”
- Here is how kernel is able to plot a decision boundary
  - 1) It doesn’t take all inputs, but the ones which represents “the inputs that belong to that class best” -> landmarks
    - We don’t take an anomaly as the representitive

        - And when we take “a point (which is reaaallyy similiar to other ones)”, then “there is no need to take the other points”
            - Because it is like giving the same input (it is almost the same with other inputs, because it is too similiar)
        - Kernel evaluates this by its “similiarity” function
    - 2) It increases the dimension size of the “feature vector”
    - 3) It finds a boundary (from transformed feature space)
    - 4) It uses cost function to evaluate the boundary
    - 5) Repeats all 4 steps until optimal boundary is reached

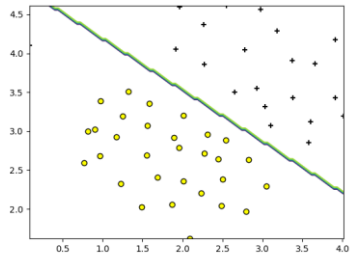
Exercise 1.1

- Since it is only about plotting the data and you will see the plotted data, I won’t put the plotted data’s image

Exercise 1.2 (Both SVM's are trained with a "linear kernel")

- Graph & function of data1
    - Input matrix X (which is the feature matrix –or vector-, which contains samples) is trained by svm.predict(…) function
        - A person instance can have "value 2.0m" for height feature and "70kg" for weight feature
        - Or it can have "value 1.75m" for height feature and "92 kg" for weight feature
    - Each row is an example, and each column symbolizes a feature of that example

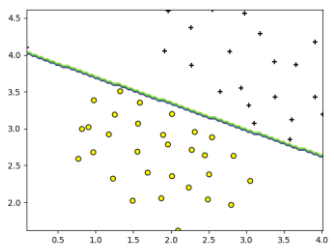
```
data = sio.loadmat('data\ex6data1.mat')
X = data['X']
y = data['y'].ravel()


svm = SVC(kernel='linear', C=1.0)
svm.fit(X, y)


def visualize_boundary(X, y, svm, file_name):
    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)
    pos = (y == 1).ravel()
    neg = (y == 0).ravel()
    plt.figure()
    plt.scatter(X[pos, 0], X[pos, 1], color='black', marker='+')
    plt.scatter(
    X[neg, 0], X[neg, 1], color='yellow', edgecolors='black', marker='o')
    plt.contour(x1, x2, yp)
    plt.savefig(file_name)
    plt.close()

visualize_boundary(X=X, y=y, svm=svm, file_name="ex6data1-Fig 1.2.2.png")
```




- Graph & function of data 2

Same function, only C value is changed to 100

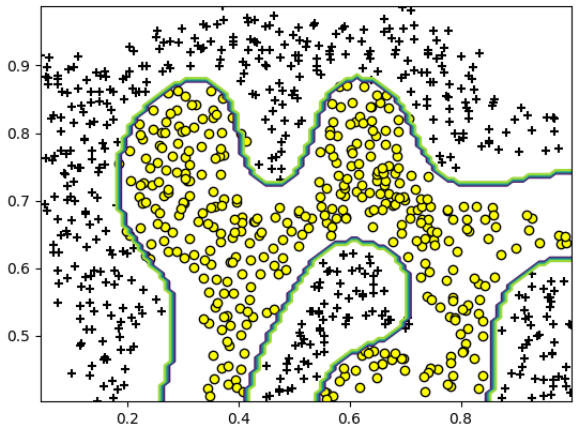
```
data = sio.loadmat('data\ex6data1.mat')
X = data['X']
y = data['y'].ravel()


svm = SVC(kernel='linear', C=100.0)
svm.fit(X, y)


def visualize_boundary(X, y, svm, file_name):
    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)
    pos = (y == 1).ravel()
    neg = (y == 0).ravel()
    plt.figure()
    plt.scatter(X[pos, 0], X[pos, 1], color='black', marker='+')
    plt.scatter(
    X[neg, 0], X[neg, 1], color='yellow', edgecolors='black', marker='o')
    plt.contour(x1, x2, yp)
    plt.savefig(file_name)
    plt.close()

visualize_boundary(X=X, y=y, svm=svm, file_name="ex6data1-Fig 1.2.2.png")
```



Exercise 1.3

- It uses exaclty the same functions
- Only difference is that “Gaussian Kernel” is used here
    - For the given file, a “non-linear boundary” was needed to correctly classify

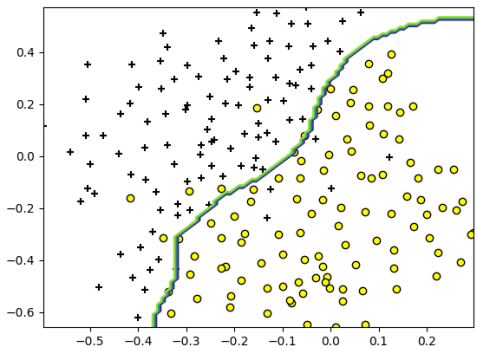
```
data = sio.loadmat('data\ex6data2.mat')
X = data['X']
y = data['y'].ravel()

C = 1
sigma = 0.1

svm = SVC(kernel='rbf', C=C, gamma=1 / (2 * sigma**2))
svm.fit(X, y)


def visualize_boundary(X, y, svm, file_name):
    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)
    pos = (y == 1).ravel()
    neg = (y == 0).ravel()
    plt.figure()
    plt.scatter(X[pos, 0], X[pos, 1], color='black', marker='+')
    plt.scatter(
    X[neg, 0], X[neg, 1], color='yellow', edgecolors='black', marker='o')
    plt.contour(x1, x2, yp)
    plt.savefig(file_name)
    plt.close()

visualize_boundary(X=X, y=y, svm=svm, file_name="ex6data1-Fig 1.3.png")
```



Exercise 1.4

- Although it might look too complex, what we are doing actually is still the same thing from the previous examples
    - We create a svc instance
    - And then train it with kernel
- The only difference here is that, we are “re-training (from scratch)” for each C and sigma values

```
data = sio.loadmat('data\ex6data3.mat')
X = data['X']
y = data['y'].ravel()
Xval = data['Xval']
yval = data['yval'].ravel()

# Define the values of C and σ to try
C_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
sigma_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

best_score = 0
best_params = {'C': None, 'sigma': None}

# Loop through all combinations of C and σ
for C in C_values:
    for sigma in sigma_values:
        # Train the SVM with RBF kernel
        svm = SVC(kernel='rbf', C=C, gamma=1 / (2 * sigma**2))
        svm.fit(X, y)

        # Evaluate the SVM on the validation set
        score = svm.score(Xval, yval)

        # Update the best score and parameters if this model is better
        if score > best_score:
            best_score = score
            best_params['C'] = C
            best_params['sigma'] = sigma

# Train the best model on the entire dataset
best_svm = SVC(kernel='rbf', C=best_params['C'], gamma=1 / (2 * best_params['sigma']**2))
best_svm.fit(X, y)

print("Best parameters found:")
print("C:", best_params['C'])
print("sigma:", best_params['sigma'])
print("Validation Accuracy:", best_score)


def visualize_boundary(X, y, svm, file_name):
    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)
    pos = (y == 1).ravel()
    neg = (y == 0).ravel()
    plt.figure()
    plt.scatter(X[pos, 0], X[pos, 1], color='black', marker='+')
    plt.scatter(
    X[neg, 0], X[neg, 1], color='yellow', edgecolors='black', marker='o')
    plt.contour(x1, x2, yp)
    plt.savefig(file_name)
    plt.close()

visualize_boundary(X=X, y=y, svm=best_svm, file_name="ex6data1-Fig 1.4.png")
```



Testing

- Contents of testing
    - SVM vs NN (week 5)
    - SVM vs NN (pytorch) (ReLU)
    - SVM vs NN (pytorch) (sigmoid)
    - SVM vs Logistic Regression (without hyperparameter tuning)
    - SVM vs Logistic  Regression (with hyperparameter tuning)

SVM vs NN (week 5)

```
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9457459926017263
Precision: 0.375
Recall: 0.07142857142857142
F1-score: 0.12
C:\Users\ardah\OneDrive\Masaüstü\p5\nn.py:333: OptimizeWarning: Unknown solver options: maxiter
Training Accuracy: 94.69%
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9469790382244143
Precision: 0.0
Recall: 0.0
F1-score: 0.0
C:\Users\ardah\OneDrive\Masaüstü\p5\nn.py:333: OptimizeWarning: Unknown solver options: maxiter
Training Accuracy: 94.51%
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9482120838471023
Precision: 0.25
Recall: 0.05263157894736842
F1-score: 0.08695652173913043
C:\Users\ardah\OneDrive\Masaüstü\p5\nn.py:333: OptimizeWarning: Unknown solver options: maxiter
Training Accuracy: 94.57%
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9420468557336621
Precision: 0.3333333333333333
Recall: 0.044444444444444446
F1-score: 0.0784313725490196
C:\Users\ardah\OneDrive\Masaüstü\p5\nn.py:333: OptimizeWarning: Unknown solver options: maxiter
Training Accuracy: 94.79%
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9420468557336621
Precision: 0.16666666666666666
Recall: 0.023255813953488372
F1-score: 0.040816326530612245
C:\Users\ardah\OneDrive\Masaüstü\p5\nn.py:333: OptimizeWarning: Unknown solver options: maxiter
  result = minimize(fun=grad, x0=initial_theta, args=(theta1, X, y, lambda_), method='TNC', jac=True, options={'maxiter': 100})
Training Accuracy: 94.73%
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9432799013563502
Precision: 0.5714285714285714
Recall: 0.0851063829787234
F1-score: 0.14814814814814814
C:\Users\ardah\OneDrive\Masaüstü\p5\nn.py:333: OptimizeWarning: Unknown solver options: maxiter
  result = minimize(fun=grad, x0=initial_theta, args=(theta1, X, y, lambda_), method='TNC', jac=True, options={'maxiter': 100})
Training Accuracy: 94.85%
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9445129469790382
Precision: 0.5
Recall: 0.044444444444444446
F1-score: 0.08163265306122448
C:\Users\ardah\OneDrive\Masaüstü\p5\nn.py:333: OptimizeWarning: Unknown solver options: maxiter
  result = minimize(fun=grad, x0=initial_theta, args=(theta1, X, y, lambda_), method='TNC', jac=True, options={'maxiter': 100})
Training Accuracy: 94.79%
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9457459926017263
Precision: 0.3333333333333333
Recall: 0.023255813953488372
F1-score: 0.043478260869565216
C:\Users\ardah\OneDrive\Masaüstü\p5\nn.py:333: OptimizeWarning: Unknown solver options: maxiter
  result = minimize(fun=grad, x0=initial_theta, args=(theta1, X, y, lambda_), method='TNC', jac=True, options={'maxiter': 100})
Training Accuracy: 94.73%
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9506781750924784
Precision: 0.5714285714285714
Recall: 0.09756097560975561
F1-score: 0.16666666666666666
C:\Users\ardah\OneDrive\Masaüstü\p5\nn.py:333: OptimizeWarning: Unknown solver options: maxiter
  result = minimize(fun=grad, x0=initial_theta, args=(theta1, X, y, lambda_), method='TNC', jac=True, options={'maxiter': 100})
Training Accuracy: 94.66%
PS C:\Users\ardah\OneDrive\Masaüstü\p7>
```

- Upper results belongs to the SVM while lower results belong to the NN
    - SVM
        - Generally gets an accuracy between 0.94-0.95
            - Results are generally a bit lower than SVM but the “highest accuracy rate of SVM” is higher than the “highest accuracy of NN”
    - NN
        - Generally gets an accuracy between 0.945-0.95
- The reason I believe NN Works better is because:
    - 1) It is similar to logistic regression
        - NN can be also described as “using logistic regression multiple times”
            - NN also performs better than logistic regression as well
    - 2) It uses a model and trains that model with input all the time (instead of just creating new boundaries)

SVM vs NN (pytorch) (ReLU)

```
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9519112207151664
Precision: 0.2
F1-score: 0.04878048780487805
Training finished.
Accuracy: 0.9420468557336621
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9457459926017263
Precision: 0.3
F1-score: 0.12
Training finished.
Accuracy: 0.9445129469790382
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9568434032059187
Precision: 0.8
F1-score: 0.18604651162790697
Training finished.
Accuracy: 0.9506781750924784
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9420468557336621
Precision: 0.2
Recall: 0.022727272727272728
F1-score: 0.04081632653061224
Training finished.
Accuracy: 0.9309494451294698
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.938347718865598
Precision: 0.3333333333333333
Recall: 0.041666666666666664
F1-score: 0.07407407407407407
Training finished.
Accuracy: 0.9309494451294698
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9469790382244143
Precision: 0.6666666666666666
Recall: 0.08888888888888889
F1-score: 0.1568627450980392
Training finished.
Accuracy: 0.9420468557336621
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9531442663378545
Precision: 1.0
Recall: 0.05
F1-score: 0.09523809523809523
Training finished.
Accuracy: 0.9506781750924784
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9482120838471023
Precision: 0.0
Recall: 0.0
F1-score: 0.0
Training finished.
Accuracy: 0.9445129469790382
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9531442663378545
Precision: 0.6
Recall: 0.07692307692307693
F1-score: 0.13636363636363635
Training finished.
Accuracy: 0.9519112207151664
PS C:\Users\ardah\OneDrive\Masaüstü\p7>
```

- Again, the results are close to each other…
- But, surprisingly this time “SVM performed better than NN (pytorch) (ReLU)
    - I believe, the reason behind this is I used “ReLU” as activation function
    - The “NN (week 5)” had used “sigmoid” as activation function

SVM vs NN (pytorch) (sigmoid)

- Like “NN (week 5)” it perfored mostly slightly better than the SVM
    - And this function is almost same with NN (week 5)

```
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9445129469790382
Precision: 0.4
F1-score: 0.08163265306122448
Training finished.
Accuracy: 0.9457459926017263
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9482120838471023
Precision: 0.5
F1-score: 0.045454545454545456
Training finished.
Accuracy: 0.9482120838471023
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9568434032059187
Precision: 0.4
F1-score: 0.10256410256410256
Training finished.
Accuracy: 0.9580764488286067
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9334155363748459
Precision: 0.1
F1-score: 0.03571428571428571
Training finished.
Accuracy: 0.9432799013563502
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9284833538840938
Precision: 0.2857142857142857
F1-score: 0.06451612903225806
Training finished.
Accuracy: 0.9321824907521579
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9321824907521579
Precision: 0.0
F1-score: 0.0
Training finished.
Accuracy: 0.9346485819975339
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9469790382244143
Precision: 0.2
Recall: 0.025
F1-score: 0.044444444444444446
Training finished.
Accuracy: 0.9506781750924784
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.93711467324291
Precision: 0.5
Recall: 0.058823529411764705
F1-score: 0.10526315789473684
Training finished.
Accuracy: 0.93711467324291
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9469790382244143
Precision: 0.3333333333333333
Recall: 0.04878048780487805
F1-score: 0.0851063829787234
Training finished.
Accuracy: 0.9494451294697904
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9395807644882861
Precision: 0.6666666666666666
Recall: 0.04
F1-score: 0.07547169811320754
Training finished.
Accuracy: 0.938347718865598
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9556103575832305
Precision: 0.6666666666666666
Recall: 0.05405405405405406
F1-score: 0.1
Training finished.
Accuracy: 0.9543773119605425
PS C:\Users\ardah\OneDrive\Masaüstü\p7>
```

SVM vs Logistic Regreession (without hyperparameter tuning)

- SVM mostly performed much better than Logistic Regression…
    - SVM was worse than NN but here it is better than Logistic Regression
        - Because Log. Reg. is more simpler than NN
            - NN uses multiple Log. Reg.s
        - Also it is not regularized and hyper-parameter tuned

```
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9420468557336621
Recall: 0.0851063829787234
F1-score: 0.14545454545454545
Accuracy: 0.938347718865598
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.938347718865598
Recall: 0.020833333333333332
F1-score: 0.038461538461538464
Accuracy: 0.93711467324291
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9408138101109741
Recall: 0.041666666666666664
F1-score: 0.07692307692307693
Accuracy: 0.9445129469790382
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9457459926017263
Recall: 0.05555555555555555
F1-score: 0.08333333333333333
Accuracy: 0.938347718865598
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9506781750924784
Recall: 0.025
F1-score: 0.047619047619047616
Accuracy: 0.9408138101109741
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9506781750924784
Recall: 0.05263157894736842
F1-score: 0.09090909090909091
Accuracy: 0.9494451294697904
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9568434032059187
Precision: 0.5
Recall: 0.02857142857142857
F1-score: 0.05405405405405406
Accuracy: 0.9494451294697904
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9408138101109741
Precision: 0.25
Recall: 0.021739130434782608
F1-score: 0.04
Accuracy: 0.938347718865598
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9482120838471023
Precision: 1.0
Recall: 0.045454545454545456
F1-score: 0.08695652173913043
Accuracy: 0.9395807644882861
```

SVM vs Logistic Regression (without hyperparameter tuning)

- Interestigly, mostly Logistic Regression is much better than SVM in this case
    - It is because we hyperparameter tuned
        - I wasn't able to change the polynomial degree because the email data is too big to run
    - And the case log. Reg. performed worse, is I believe that only because "I didn't made 'alpha search' deep enough"
        - Because there are sometimes where both models had reached exactly the same accuracy
            - Because search for alpha was deep enough

```
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9469790382244143
Precision: 0.6
Recall: 0.06818181818181818
F1-score: 0.12244897959183673
Best C: 0.003
Accuracy Rate: 94.82120838471023
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9543773119605425
Precision: 0.5
Recall: 0.05405405405405406
F1-score: 0.0975609756097561
Best C: 0.001
Accuracy Rate: 95.43773119605426
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9519112207151664
Precision: 1.0
Recall: 0.04878048780487805
F1-score: 0.09302325581395349
Best C: 0.009187878787878787
Accuracy Rate: 95.31442663378546
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9457459926017263
Precision: 0.5
Recall: 0.06818181818181818
F1-score: 0.12
Best C: 0.001
Accuracy Rate: 94.57459926017263
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.935881627620222
Precision: 0.75
Recall: 0.05555555555555555
F1-score: 0.10344827586206896
Best C: 0.0031272727272727272
Accuracy Rate: 93.5881627620222
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9395807644882861
Precision: 0.6666666666666666
Recall: 0.04
F1-score: 0.07547169811320754
Best C: 0.008172727272727272
Accuracy Rate: 94.0813810110974
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9482120838471023
Precision: 0.42857142857142855
Recall: 0.07317073170731707
F1-score: 0.125
Best C: 0.0001
Accuracy Rate: 94.94451294697905
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9395807644882861
Precision: 0.16666666666666666
Recall: 0.022222222222222223
F1-score: 0.03921568627450980
Best C: 0.0001
Accuracy Rate: 94.45129469790382
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.9469790382244143
Precision: 0.6666666666666666
Recall: 0.045454545454545456
F1-score: 0.0851063829787234
Best C: 0.0001
Accuracy Rate: 94.57459926017263
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
Accuracy: 0.938347718865598
Precision: 0.3333333333333333
Recall: 0.06382978723404255
F1-score: 0.10714285714285714
Best Degree: 1
Best C: 0.002118181818181818
Accuracy Rate: 94.32799013563502
PS C:\Users\ardah\OneDrive\Masaüstü\p7> python -u "c:\Users\ardah\OneDrive\Masaüstü\p7\p7.py"
```