COMP4434 Big Data Analytics

Assignment 2

Author
Wang Yuqi
[redacted]

Lecturer
Prof. Xiao HUANG

BsC in Computer Science
Academic year: 2024/2025 Sem 2

Problems

Problem 1

(a) In the classification task, why do we prefer logistic loss over mean-squared error?

(b) How does a standard Support Vector Machine (SVM) differ from an SVM with a soft margin, and in what way do slack variables (ξi) facilitate the handling of non-linearly separable data?

(c) If your multi-layer perceptron model has an overfitting issue, what are the strategies that you could use to handle the issue?

(d) How does backpropagation use the gradient descent algorithm to update the weights in a neural network?

Problem 2

Load dataset in problem2data.txt by using "numpy.loadtxt()" in Python, which contains two synthetic classes separated by a non-linear function, with additive noise. The last column contains "0" or "1", indicating the corresponding classes. You are asked to investigate the extent to which polynomial func- tions can be used to build a logistic regression classifier.

(a) Build a logistic regression classifier by using a polynomial function of order 1 (e.g., $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$). Apply 5-fold-cross-validation to generate training and test sets. Use the training set to train the model. Compute its five F1 scores on test set (one F1 score for each fold). Compute its five F1 scores on training set. You will need "f1 score" from "sklearn.metrics".

(b) Repeat part (a) for polynomials of order 2 to 10. You will need "PolynomialFeatures" from "sklearn.preprocessing".

(c) Repeat parts (a-b) 20 times, and estimate the average F1 score for each polynomial order (average across its 5×20 runs, both for training and test sets). Generate a plot that shows the F1 scores versus the polynomial order.

(d) Discuss how the F1 scores of the model changes as a function of the polynomial order.

(e) Use Jupyter Notebook to perform implementation. You are required to submit your ".ipynb" file. You could use "LogisticRegression" from "sklearn.linear model". Do not add any regularizations.

Problem 3

Problem 3 (2 points) Consider a set of data points represented by the coordinates (x, y). The objective is to apply the k-means algorithm to identify two distinct clusters within the data. Use (0.5, 1) as the initial centroid for the first cluster and (3, 4) as the initial centroid for the second cluster. Perform a single iteration of the k-means algorithm with k = 2, utilizing cosine distance as the distance metric. Recall that cosine distance is defined as 1 - cosine similarity. The data points are listed as follows.

Data#	x	y
1	1	0.5
2	0.5	2
3	3	1
4	2	1.5

(1) Determine the cluster assignments for each data point after one iteration.
(2) Compute the new centroids for each cluster in Euclidean space after completing the first iteration.

(See Next Page)

Solutions

Solution to Problem 1(a)

Introduction

To answer this question I would first provide probablistic background of logistic loss in the context of Maximum Likelihood Estimation (MLE), then its derivative. With these insights, I will explain why logistic loss is preferred over MSE at the end of the section.

Probablistic Background

Consider the binary classification setup, where we are given m training examples $\{(x_1, y_1), ..., (x_m, y_m)\}$, and a sigmoid function $f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$ parameterized by $\mathbf{w}$

Then, the probability of observing a positive sample ($y = 1$) and negative sample ($y = 0$) given our parameterized $f_{\mathbf{w}}$ and the input feature $\mathbf{x}$ is:

$$P(y = 1 \mid x; w) = f_{\mathbf{w}}(x)$$
$$P(y = 0 \mid x; w) = 1 - f_{\mathbf{w}}(x)$$

Next, the *likelihood* of observing these labels can be derived:

$$L(w) = \prod_{i=1}^{m} P(y_i \mid x_i; w) = \prod_{i=1}^{m} (f_{\mathbf{w}}(x_i))^{y_i}(1 - f_{\mathbf{w}}(x_i))^{1-y_i}$$

Intuitively, the goal of our learned model is to maximize this likelihood given the observed data. But since the product of probabilities (values 0 to 1) can be hard to optimize due to numerical underflow, we can take the log of the likelihood to obtain the *log-likelihood*. Optimizing the log-likelihood $\mathcal{L}(w)$ is equivalent to optimizing the likelihood $L(w)$, since the log function is monotonically increasing:

$$\mathcal{L}(w) = \log(L(w)) = \sum_{i=1}^{m} y_i \log(f_{\mathbf{w}}(x_i)) + (1 - y_i) \log(1 - f_{\mathbf{w}}(x_i))$$

Currently, the log-likelihood objective aims to **maximize**, but by convention of loss functions, we want to **minimize** instead. So we take the negative of log-likelihood, obtaininig *negaitve log-likelihood*. **We have now obtained the logistic loss function**:

$$J(\mathbf{w}) = -\frac{1}{m}\mathcal{L}(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(f_{\mathbf{w}}(x_i)) + (1 - y_i) \log(1 - f_{\mathbf{w}}(x_i))$$

Gradient

Firstly, from homework 1 we know that:

1. Derivative of Sigmoid Function: $\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$ or $\sigma(z)(1 - \sigma(z))$

2. Derivative of MSE loss: $\frac{\partial J_{\text{MSE}}}{\partial \hat{y}} = (\hat{y} - y)$
3. Derivative of logistic loss: $\frac{\partial J}{\partial \hat{y}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})}$

Now let's consider the error term δ "MSE + Sigmoid" *vs* "Logistic Loss + Sigmoid":

1. Logistic Loss + Sigmoid

$$\delta = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = \left[\frac{\hat{y} - y}{\hat{y}(1-\hat{y})}\right] \cdot [\hat{y}(1-\hat{y})] = \hat{y} - y$$

2. MSE + Sigmoid

$$\delta = \frac{\partial J_{\text{MSE}}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = (\hat{y} - y) \cdot \hat{y}(1-\hat{y})$$

Answer: The reasons for pairing logistic function with Sigmoid (used in most classification task) rather than MSE is two folds:

1. **Probablistic Interpretation**:
   - Logistic loss can be diectly derived from probability using MLE
   - Thus, minimizing logistic loss is equivalent to maximizing likelihood of observing the data given our parameterized model $f_{\mathbf{w}}(\mathbf{x})$
2. **Stable Gradient**:
   - The gradient of *Sigmoid + Logistic Loss* is **linear** to the error (i.e., $\delta = \hat{y} - y$)
   - The gradient of *Sigmoid + MSE*, on the other hand, will **approach 0** when model outputs are over-confident (i.e., close to 1 or 0). This is because of the $\hat{y}(1-\hat{y})$ term in the error, thereby causing gradient vanishing.

Solution to Problem 1(b)

(b) How does a standard Support Vector Machine (SVM) differ from an SVM with a soft margin, and in what way do slack variables (ξi) facilitate the handling of non-linearly separable data?

Standard SVM Assumes that the data is linearly separable and try to find a optimal hyperplane with maximum margin while perfectly separating the data points in the training data (i.e., no misclassification allowed).

Mathematically, this can be represented as the constraint:

$$y_i(w \cdot x_i + b) \geq 1$$

Soft Margin SVM Loser constraints than the Standard SVM. It introduces a *slack variable* ξ_i which quantifies the degree of of misclassification. If a data point falls

within the margin, the $0 < \xi_i < 1$. If a data point crosses the hyperplane, $\xi_i > 1$. Another hyperparameter C controls how tolerable we are to such errors.

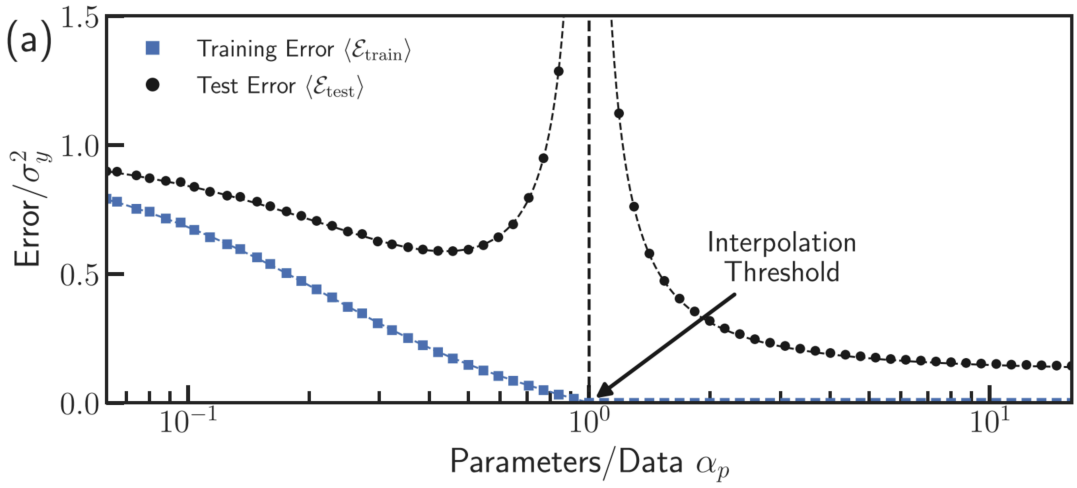
Mathematically, this can be represented as the constraint:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

Non-linear Separable Data Soft margin facilitate handling of non-linearly separable data by allowing some data points to cross the hyperplane. In contrast, the standard SVM would not be able to find a solution (no hyperplane found).

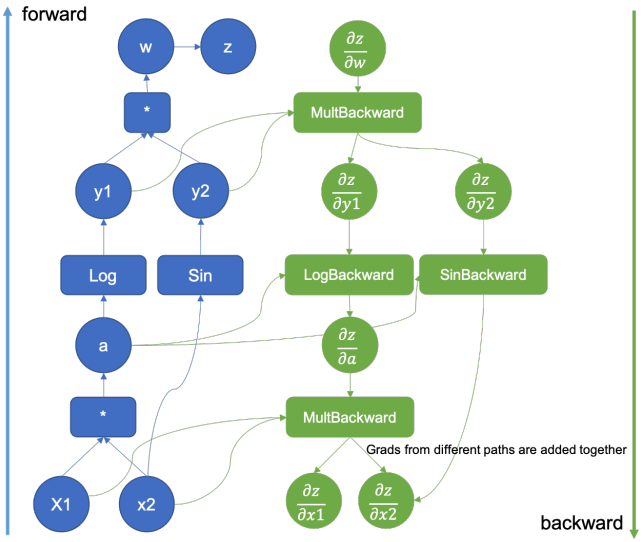
Solution to Problem 1(c)

1. **L_1 or L_2 regularization**
   - L_1 if we want sparse weights (many zeros)
   - L_2 if we want dense but small weights
2. **Dropout**
   - Random set neuron activations to zero during training
   - Can be interpreted as an ensemble of sparse neural networks
3. **Early Stopping**
   - Stop when validation error starts increasing
   - However, this does not adhere to modern philosophy of deep learning
4. **Data Augmentation**
   - Artificially create more data points by distorting the original data
5. **Overparameterization** (*see image below*)
   - Contrast to traditional statistical learning theory, deep learning often has more parameters than data points.
   - This enables a phenomenon called *double descent* where test loss decrease again after increasing number of parameter high beyond the number of data points.



Solution to Problem 1(d)

- Backpropagation is essentially just **chain rule** applied to neural networks.
- **Computation Graph**
  - Computation of forward pass forms a DAG known as *Computational Graph*.
  - Initially, each nodes stores a output value during the forward pass
  - Then, during backward pass, each node stores the gradient of the loss w.r.t its output.
  - The backward pass is hence a topolicial sort of the graph, where we compute the gradient of the loss w.r.t each node in reverse order.



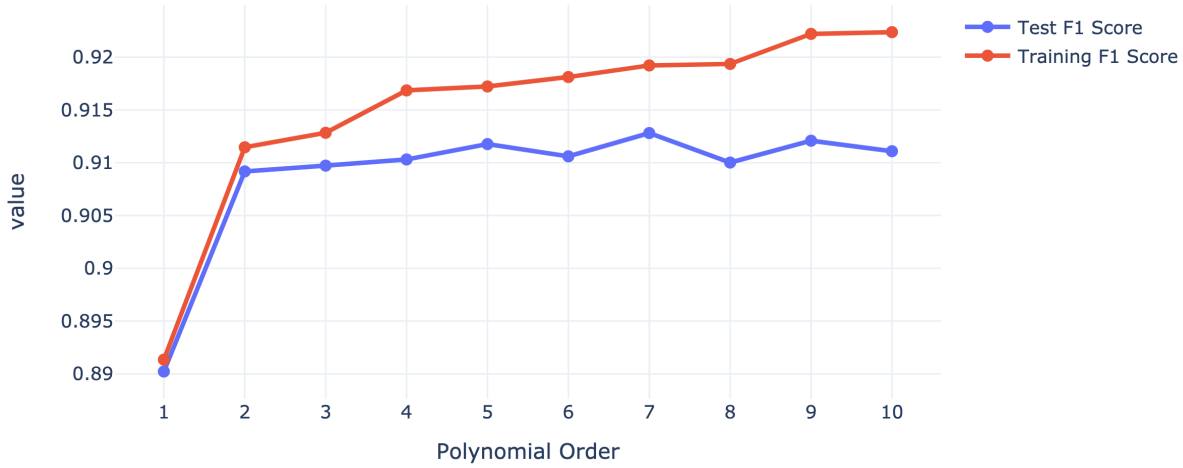
Solution to Problem 2(a)

Sample Output in **Appendix**

Solution to Problem 2(b)

Sample Output in **Appendix**

Solution to Problem 2(c)



Solution to Problem 2(d)

- **F1 Test Score**
  - Most prominent increase from order 1-2
  - Slight increase from order 2-3
  - Fluctuates slightly above 0.91 from order 3-10
- **F1 Training Score**
  - Monotonically increases with polynomial order
  - Most prominent increase from order 1-2
  - Slow and gradual increase from order 2-10

Solution to Problem 2(e)

Code uploaded as **`q2.ipynb`**

Notes:

- `max_iter` set to 5000 to avoid convergence warning
- `shuffle` set to `True` in `KFold`, otherwise part (c) would not make sense
- requires `numpy`, `pandas`, `sklearn`, `plotly` and `tqdm` (optional)

Solution to Problem 3(1)

1. **Initialization**
   - Centroid 1: $C_1 = (0.5, 1.0)$
   - Centroid 2: $C_2 = (3.0, 4.0)$
   - Data Point 1: $x_1 = (1.0, 0.5)$
   - Data Point 2: $x_2 = (0.5, 2.0)$
   - Data Point 3: $x_3 = (3.0, 1.0)$
   - Data Point 4: $x_4 = (2.0, 1.5)$
2. **Cosine Distance**

$$\cos(x, y) = 1 - \frac{x \cdot y}{\|x\| \cdot \|y\|}$$

   - Data Point 1:
     - $\cos(C_1, x_1) = 1 - \frac{0.5\times1.0+1.0\times0.5}{\sqrt{0.5^2+1^2}\cdot\sqrt{1^2+0.5^2}} = \frac{1}{5} = 0.2000$
     - $\cos(C_2, x_1) = 1 - \frac{3\times1.0+4\times0.5}{\sqrt{3^2+4^2}\cdot\sqrt{1^2+0.5^2}} = -\frac{2\sqrt{5}}{5} + 1 \approx 0.1056$
   - Data Point 2:
     - $\cos(C_1, x_2) = 1 - \frac{0.5\times0.5+1.0\times2.0}{\sqrt{0.5^2+1^2}\cdot\sqrt{0.5^2+2^2}} = -\frac{9\sqrt{85}}{85} + 1 \approx 0.0238$
     - $\cos(C_2, x_2) = 1 - \frac{3\times0.5+4\times2.0}{\sqrt{3^2+4^2}\cdot\sqrt{0.5^2+2^2}} = -\frac{19\sqrt{17}}{85} + 1 \approx 0.0784$
   - Data Point 3:

- $\cos(C_1, x_3) = 1 - \frac{0.5\times3.0+1.0\times1.0}{\sqrt{0.5^2+1^2}\cdot\sqrt{3^2+1^2}} = -\frac{\sqrt{2}}{2} + 1 \approx 0.2929$
- $\cos(C_2, x_3) = 1 - \frac{3\times3.0+4\times1.0}{\sqrt{3^2+4^2}\cdot\sqrt{3^2+1^2}} = -\frac{13\sqrt{10}}{50} + 1 \approx 0.1778$

- Data Point 4:
  - $\cos(C_1, x_4) = 1 - \frac{0.5\times2.0+1.0\times1.5}{\sqrt{0.5^2+1^2}\cdot\sqrt{2^2+1.5^2}} = -\frac{2\sqrt{5}}{5} + 1 \approx 0.1056$
  - $\cos(C_2, x_4) = 1 - \frac{3\times2.0+4\times1.5}{\sqrt{3^2+4^2}\cdot\sqrt{2^2+1.5^2}} = \frac{1}{25} = 0.040$

3. **Cluster Assignments**
   - $C_1 = \{x_2\}$
   - $C_2 = \{x_1, x_3, x_4\}$
4. **New Centroids**
   - $C_1 = [0.5, 2.0]$
   - $C_2 = \left(\frac{1+3+2}{3}, \frac{0.5+1+1.5}{3}\right) = [2.0, 1.0]$

(See Next Page)

Appendix

Codeblock 1 - Setup

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.metrics import f1_score
from sklearn.preprocessing import PolynomialFeatures

# init
data = np.loadtxt('problem2data.txt', delimiter=' ')
X = data[:, :-1] # features, get rid of last col
y = data[:, -1] # labels, get last col

def fold5(poly_order):
    kf = KFold(n_splits=5, shuffle=True)
    f1_te = []
    f1_tr = []
    for tr_idx, te_idx in kf.split(X):
        X_tr, X_te = X[tr_idx], X[te_idx]
        y_tr, y_te = y[tr_idx], y[te_idx]
        poly = PolynomialFeatures(degree=poly_order)
        X_tr_poly = poly.fit_transform(X_tr)
        X_te_poly = poly.fit_transform(X_te)

        # train
        model = LogisticRegression(max_iter=5000)
        model.fit(X_tr_poly, y_tr)
        y_te_pred = model.predict(X_te_poly)
        y_tr_pred = model.predict(X_tr_poly)

        # f1
        f1_te.append(f1_score(y_te, y_te_pred))
        f1_tr.append(f1_score(y_tr, y_tr_pred))

    return f1_te, f1_tr
```

Codeblock 2 - Part 2(a)

```
kf = KFold(n_splits=5)
f1_te, f1_tr = fold5(1)
print('===== Part (a) =====')
for i, f1 in enumerate(list(zip(f1_te, f1_tr))):
```

```
    print(f'Fold {i+1} F1 test score: {f1[0]: .7f}, F1 train score:
{f1_tr[1]: .7f}')
```

Codeblock 3 - Part 2(b)

```
for ord in range(2, 11):
    f1_te, f1_tr = fold5(ord)
    print(f'===== Part (b) Order {ord} =====')
    for i, f1 in enumerate(list(zip(f1_te, f1_tr))):
        print(f'Fold {i+1} F1 test score: {f1[0]: .7f}, F1 train score:
{f1[1]: .7f}')
```

Codeblock 4 - Part 2(c)

```
import plotly.express as px
import pandas as pd
from tqdm import tqdm # comment out this line if progress bar is not
wanted

print('===== Part (c) =====')
f1_te_avg = np.zeros(10)
f1_tr_avg = np.zeros(10)

# for ord in range(2, 11): # use this line if progress bar is not wanted
for ord in tqdm(range(1, 11)):
    for it in range(20):
        f1_te, f1_tr = fold5(ord)
        f1_te_avg[ord - 1] += np.sum(f1_te)
        f1_tr_avg[ord - 1] += np.sum(f1_tr)

f1_te_avg /= 100
f1_tr_avg /= 100

print('===== Part (c) Plotting =====')
print(f'Average F1 Test Score: {f1_te_avg}')
print(f'Average F1 Training Score: {f1_tr_avg}')

df = pd.DataFrame({
    'Polynomial Order': np.arange(1, 11),
    'Test F1 Score': f1_te_avg,
    'Training F1 Score': f1_tr_avg
})

fig = px.line(df,
    x='Polynomial Order',
    y=['Test F1 Score', 'Training F1 Score'],
    markers=True,
```

```
    color_discrete_sequence=['#636EFA', '#EF553B'],
    template='plotly_white'
)

fig.update_layout(
    legend_title_text='',
    xaxis=dict(tickmode='linear', dtick=1),
    font=dict(size=12),
    width=800,
    height=400
)

fig.update_traces(line=dict(width=3), marker=dict(size=8))
fig.write_image('q2c_plot.png', scale=3)
fig.show()
```

Output of Codeblock 2 - Part 2(a)

```
===== Part (a) =====
Fold 1 F1 test score:  0.8888889, F1 train score:  0.8911139
Fold 2 F1 test score:  0.9009901, F1 train score:  0.8908629
Fold 3 F1 test score:  0.8723404, F1 train score:  0.8955224
Fold 4 F1 test score:  0.8865979, F1 train score:  0.8908407
Fold 5 F1 test score:  0.9056604, F1 train score:  0.8877551
```

Output of Codeblock 3 - Part 2(b)

```
===== Part (b) Order 2 =====
Fold 1 F1 test score:  0.8932039, F1 train score:  0.9142857
Fold 2 F1 test score:  0.9090909, F1 train score:  0.9118388
Fold 3 F1 test score:  0.9082126, F1 train score:  0.9115816
Fold 4 F1 test score:  0.9345794, F1 train score:  0.9033877
Fold 5 F1 test score:  0.8791209, F1 train score:  0.9199029
===== Part (b) Order 3 =====
Fold 1 F1 test score:  0.9178744, F1 train score:  0.9122373
Fold 2 F1 test score:  0.9210526, F1 train score:  0.9090909
Fold 3 F1 test score:  0.8715084, F1 train score:  0.9207101
Fold 4 F1 test score:  0.9615385, F1 train score:  0.9019608
Fold 5 F1 test score:  0.8700000, F1 train score:  0.9236364
===== Part (b) Order 4 =====
Fold 1 F1 test score:  0.9381443, F1 train score:  0.9119421
Fold 2 F1 test score:  0.8909953, F1 train score:  0.9198520
Fold 3 F1 test score:  0.9214660, F1 train score:  0.9169675
Fold 4 F1 test score:  0.8815166, F1 train score:  0.9247842
Fold 5 F1 test score:  0.9345794, F1 train score:  0.9106700
===== Part (b) Order 5 =====
Fold 1 F1 test score:  0.9209302, F1 train score:  0.9167702
```

```
Fold 2 F1 test score:  0.8780488, F1 train score:  0.9232673
Fold 3 F1 test score:  0.8983957, F1 train score:  0.9215923
Fold 4 F1 test score:  0.9417476, F1 train score:  0.9093137
Fold 5 F1 test score:  0.9100000, F1 train score:  0.9189189
===== Part (b) Order 6 =====
Fold 1 F1 test score:  0.9327354, F1 train score:  0.9086358
Fold 2 F1 test score:  0.8936170, F1 train score:  0.9208633
Fold 3 F1 test score:  0.8921569, F1 train score:  0.9226933
Fold 4 F1 test score:  0.9035533, F1 train score:  0.9210207
Fold 5 F1 test score:  0.9178744, F1 train score:  0.9144981
===== Part (b) Order 7 =====
Fold 1 F1 test score:  0.9035533, F1 train score:  0.9247842
Fold 2 F1 test score:  0.9065421, F1 train score:  0.9192547
Fold 3 F1 test score:  0.9339623, F1 train score:  0.9142857
Fold 4 F1 test score:  0.9215686, F1 train score:  0.9164619
Fold 5 F1 test score:  0.9081081, F1 train score:  0.9184652
===== Part (b) Order 8 =====
Fold 1 F1 test score:  0.9238095, F1 train score:  0.9177057
Fold 2 F1 test score:  0.9321267, F1 train score:  0.9118388
Fold 3 F1 test score:  0.8743169, F1 train score:  0.9278846
Fold 4 F1 test score:  0.8800000, F1 train score:  0.9257004
Fold 5 F1 test score:  0.9282297, F1 train score:  0.9079755
===== Part (b) Order 9 =====
Fold 1 F1 test score:  0.8877551, F1 train score:  0.9273608
Fold 2 F1 test score:  0.9315068, F1 train score:  0.9180328
Fold 3 F1 test score:  0.9009901, F1 train score:  0.9251534
Fold 4 F1 test score:  0.9333333, F1 train score:  0.9174757
Fold 5 F1 test score:  0.9126214, F1 train score:  0.9223181
===== Part (b) Order 10 =====
Fold 1 F1 test score:  0.8975610, F1 train score:  0.9306931
Fold 2 F1 test score:  0.9346734, F1 train score:  0.9157509
Fold 3 F1 test score:  0.9019608, F1 train score:  0.9240196
Fold 4 F1 test score:  0.9019608, F1 train score:  0.9200492
Fold 5 F1 test score:  0.9230769, F1 train score:  0.9190535
```

Output of Codeblock 4 - Part 2(c)

```
===== Part (c) Plotting =====
Average F1 Test Score: [0.89033112 0.90901039 0.91037343 0.91010093
0.91213525 0.91210984 0.91248895 0.9102826  0.91238477 0.90964869]
Average F1 Training Score: [0.89125332 0.91116713 0.91235808 0.91669758
0.91722523 0.91809442 0.91871909 0.9195183  0.92213372 0.92228248]
```