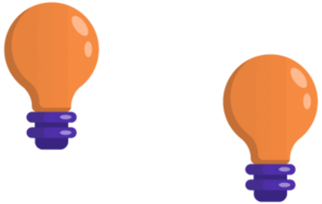
Logistic Regression Model

Group 10 In-class Presentation

Andrew Du
Yunjie Xu
Steven Xie
Siyuan Wang
Yunhao Zheng
Haowen Li
Zhijian Zhu

Agenda

1. Data Preparation
    a. Data Overview
    b. Feature Engineering (Standardization & Feature Selection)
2. Logistic Regression (SKlearn)
    a. Bias-Variance Tradeoff
    b. Hyperparameter tuning
        i. What is C
        ii. Grid search
    c. Results Interpretation
3. Direct Optimization
    a. Objective function
    b. Solving the Objective Function in Python
    c. Results Interpretation

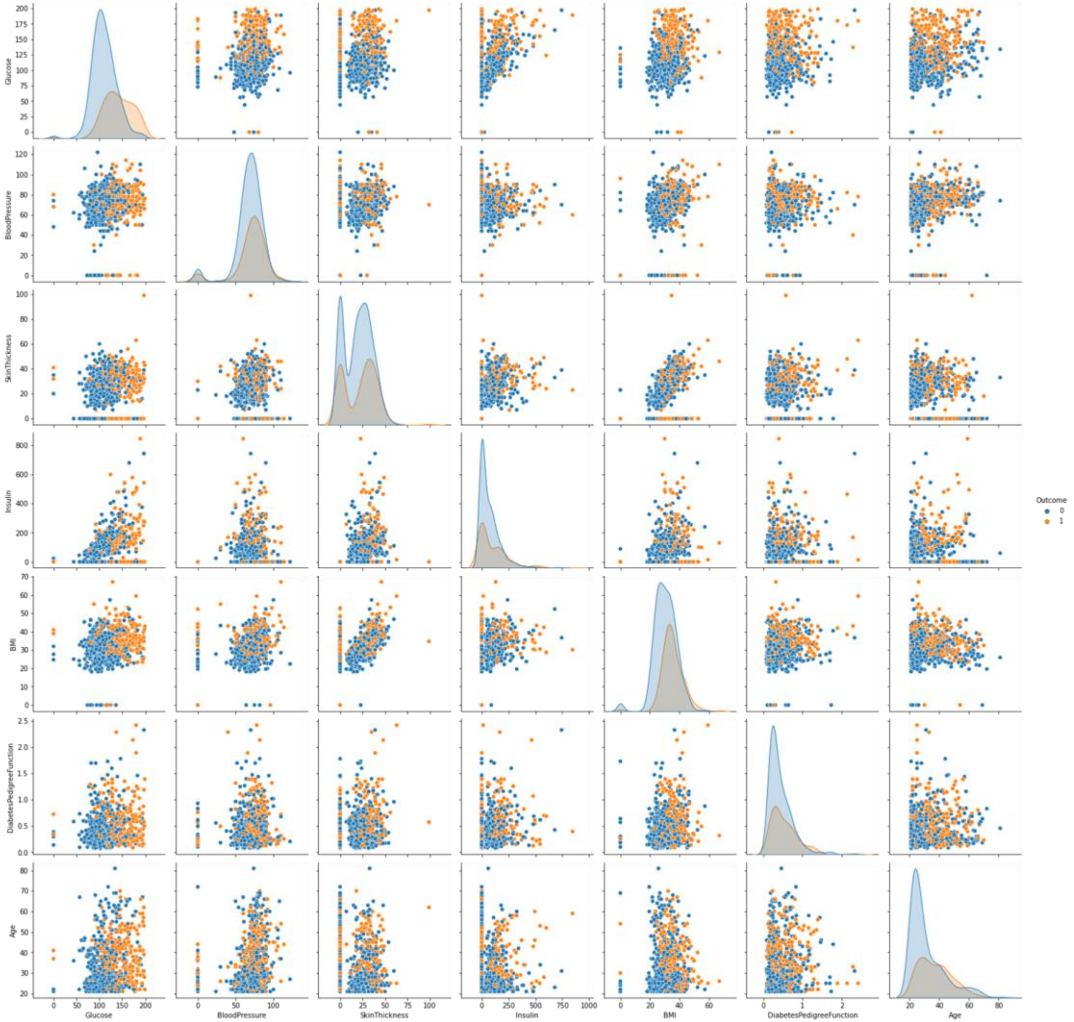


Data Preparation

Data Overview

- 7 Features
  - Glucose
  - Blood Pressure
  - Skin Thickness
  - Insulin
  - BMI
  - Diabetes Pedigree Function
  - Age

- Binary Outcome
  - Healthy (0)
  - Diabetics (1)



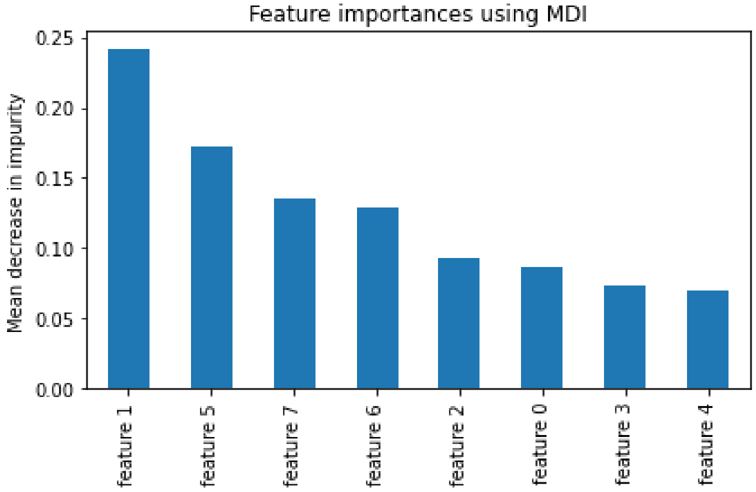
Feature Engineering

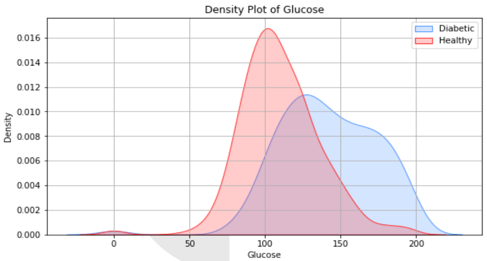
Standardization

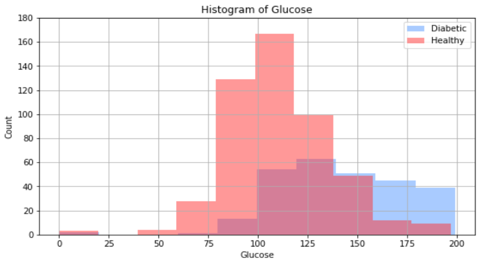
- Using standard scalar to standardize the data

Feature Selection

- Extract feature importance by random forest
- Feature ranking
  - Feature_1: Glucose
  - Feature_5: BMI
  - Feature_6: Diabetes Pedigree Function
  - Feature_7: Age

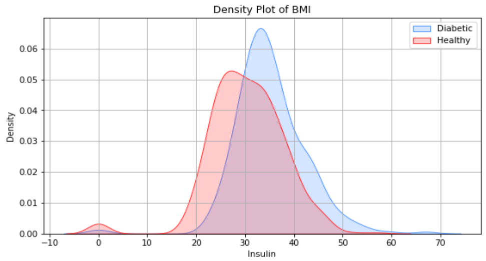


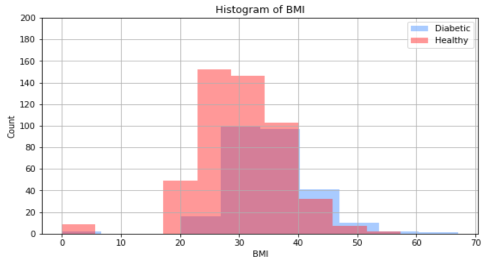


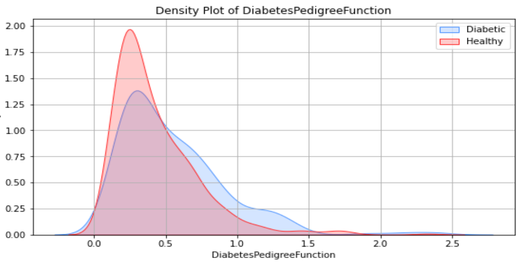


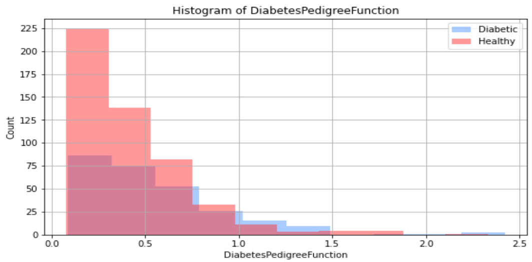
Feature 1: Glucose

Feature 5: BMI



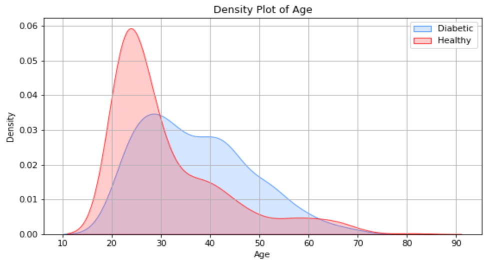


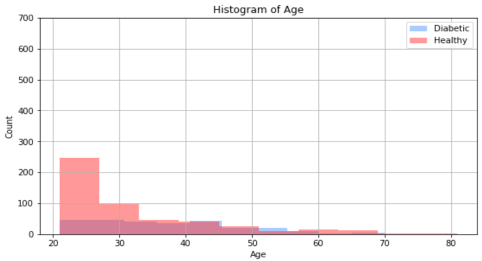




Feature 6: Diabetes_Pedigree_Function

Feature 7: Age

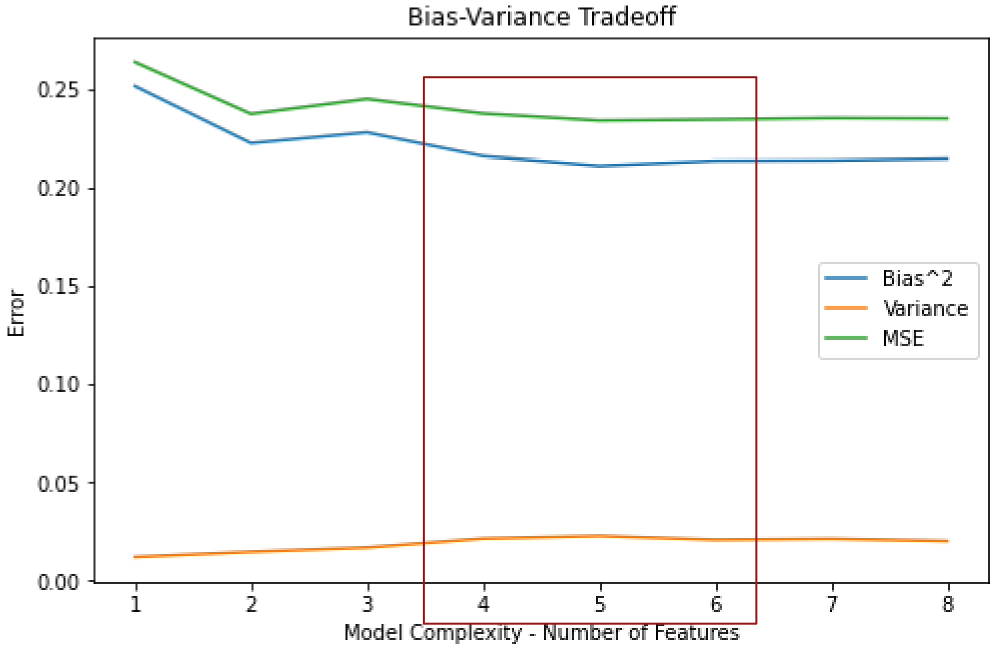




Logistic Regression (SKlearn)

Bias-Variance Tradeoff

	MSE	Bias^2	Variance
1	0.263615	0.251385	0.01223
2	0.237273	0.222434	0.014839
3	0.244892	0.227873	0.017019
4	0.237489	0.215909	0.02158
5	0.233896	0.210979	0.022917
6	0.234459	0.213501	0.020958
7	0.235152	0.213705	0.021446
8	0.234913	0.214594	0.020319



Hyperparameter Tuning - Grid Search CV

- L1 - Regularized regression with scikit-learn module
- Hyperparameter C - Inverse of regularization strength
- Solver - ['saga', 'liblinear']

Hyperparameter Tuning - Results

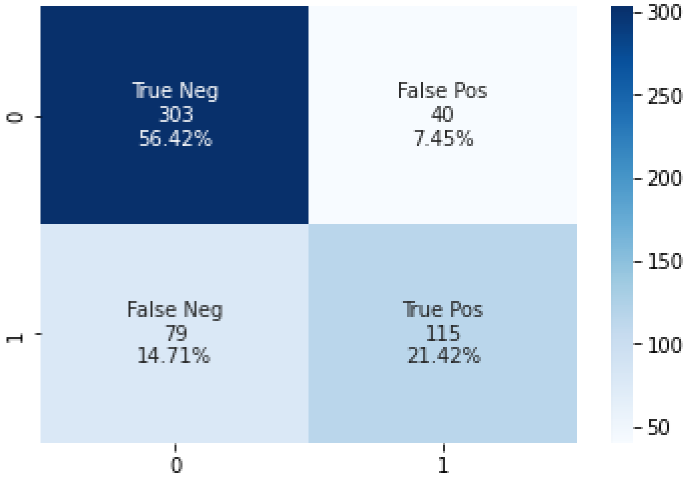
```
#grid search for hyper-parameter C and solver
grid = dict()
grid['solver'] = ['saga', 'liblinear']
grid['C'] = np.linspace(0, 5, 101)
log = LogisticRegression(penalty='l1')
logreg_cv = GridSearchCV(log, grid, cv=10)
logreg_cv.fit(X_train.iloc[:,:5],y_train)

print("best parameters: ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

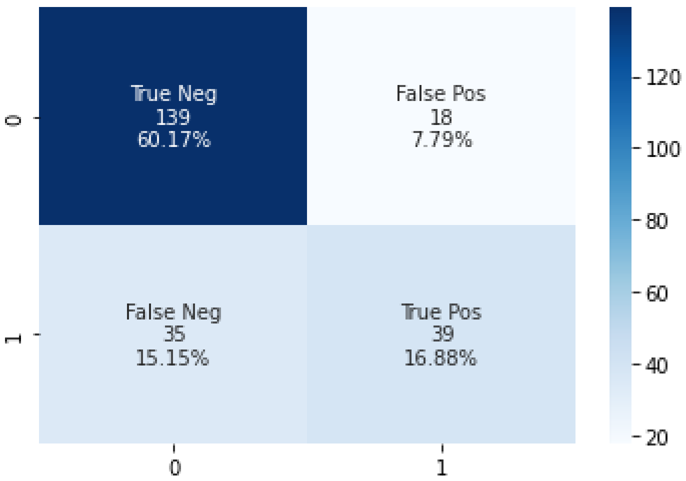
```
best parameters: {'C': 0.15000000000000002, 'solver': 'saga'}
accuracy : 0.7746331236897275
```

Visualize the Confusion Matrix

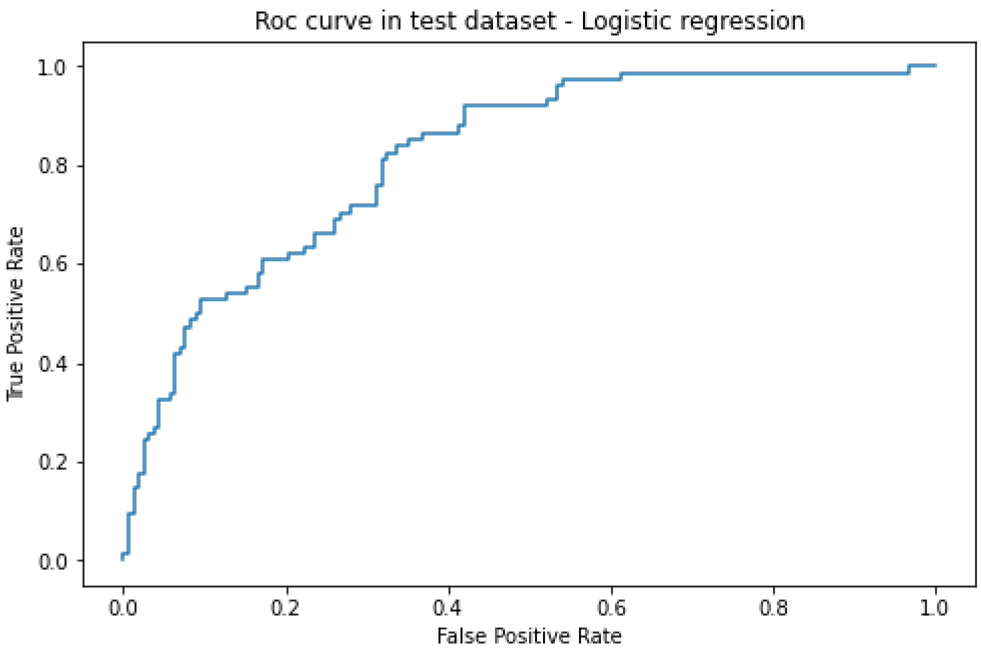
Training Accuracy: 0.78



Testing Accuracy: 0.77

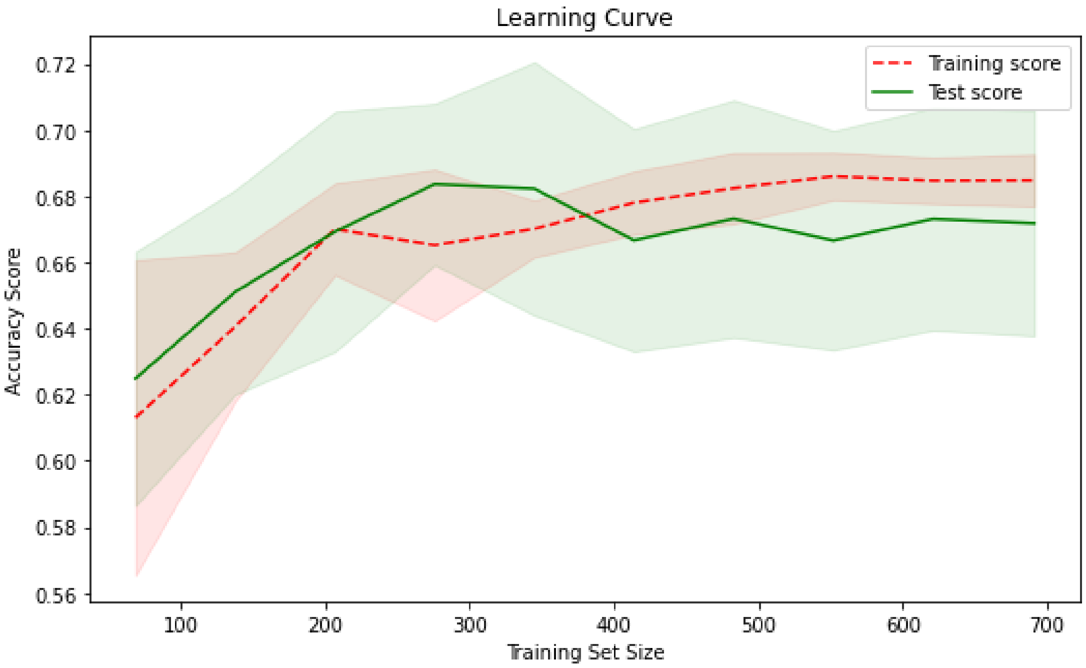


Visualize the ROC Curve



roc_auc_score : 0.8174384575658461

Visualize the Learning Curve



Direct Optimization

Direct Approach
by Solving an Optimization Problem

Objective Function

$$\min_{\boldsymbol{\theta}} \; \|\boldsymbol{\theta}\|_1 + C \cdot J(\boldsymbol{\theta})$$

Where J(***θ***) is the cross-entropy loss function

$$\text{Loss} = -\frac{1}{\substack{\text{output} \\ \text{size}}} \sum_{i=1}^{\substack{\text{output} \\ \text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log \left(1 - \hat{y}_i\right)$$

ŷ are the mapped real values using the sigmoid activation function

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$

Forming the Objective Function in Python

```
def sigmoid(x):
    # Activation function used to map any real value between 0 and 1
    return 1/(1 + np.exp(-x))

def cost_function(x):
    theta = np.array([[x[0], x[1], x[2], x[3], x[4]]])

    # Initialisation of useful values
    m = np.size(y_train)
    hx = sigmoid(X @ theta.T)

    # Cost function
    J = -(1 / m) * np.sum(y * np.log(hx) + (1 - y) * np.log(1 - hx))

    # Add L1 regularization and Hyperparameter
    J = x[5] * J + np.sum(abs(theta))
    return J
```

Scipy Optimization Solver minimize()

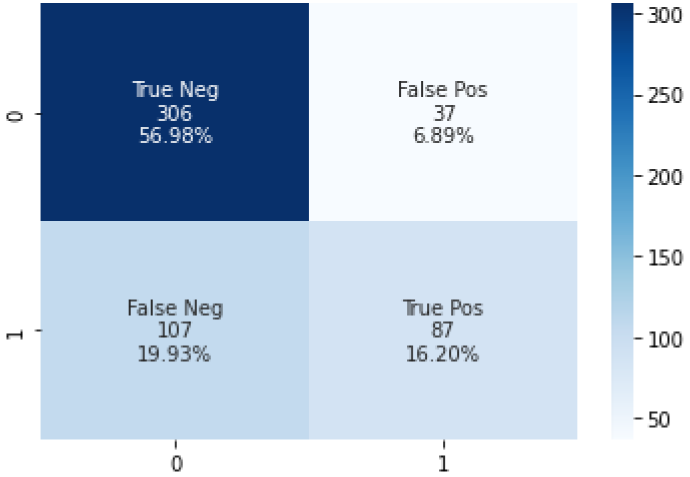
```
x_0 = [0.001]*6
bnds = ((None, None), (None, None), (None, None), (None, None), (None, None), (0.001, 100))
sol = minimize(cost_function, x_0, bounds=bnds, method = 'Powell')

thetas = np.array([sol.x[0:5]])
opt_c = np.array([sol.x[-1]])
y_pred_test = X_test_std_1[:,:5] @ thetas.T > 0
y_pred_train = X_train_std_1[:,:5] @ thetas.T > 0
train_score = accuracy_score(y, y_pred_train)
test_score = accuracy_score(y_test, y_pred_test)
print('The training score is: %s'%round(train_score,2))
print('The testing score is: %s'%round(test_score,2))
print(sol)
```

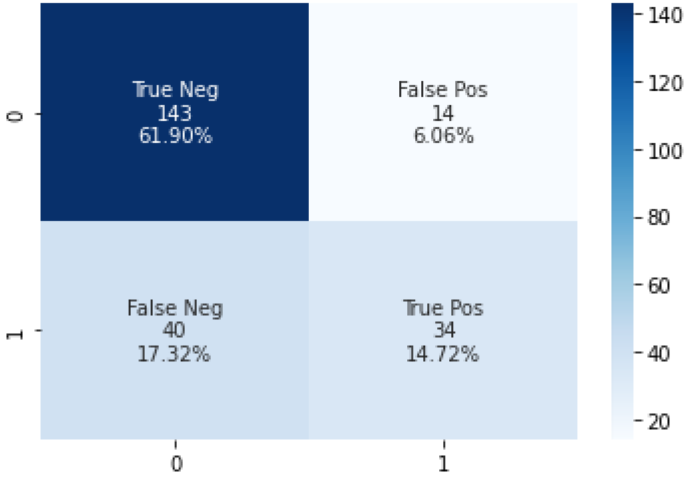
```
The training score is: 0.73
The testing score is: 0.77
   direc: array([[1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.]])
     fun: 0.3921994296328144
 message: 'Optimization terminated successfully.'
    nfev: 131
     nit: 1
  status: 0
 success: True
       x: array([-4.39727436e-06,  3.63312116e-06,  3.11091491e-06, -1.22322677e-06,
        3.87919660e-06,  1.05363359e-03])
```

Optimization Results
Model Accuracy & Confusion Matrix

Training Accuracy: 0.73



Testing Accuracy: 0.77



Questions?

Thank you!

Bias-Variance Tradeoff VS.
Number of Folds in Cross Validation