Classification by Logistic Regression

Group 8

Tianyi Yu, Zihao Jiao, Bowen Xu,
Jianhui Li, Xiaochen Sun, Chang Liu

Outlines

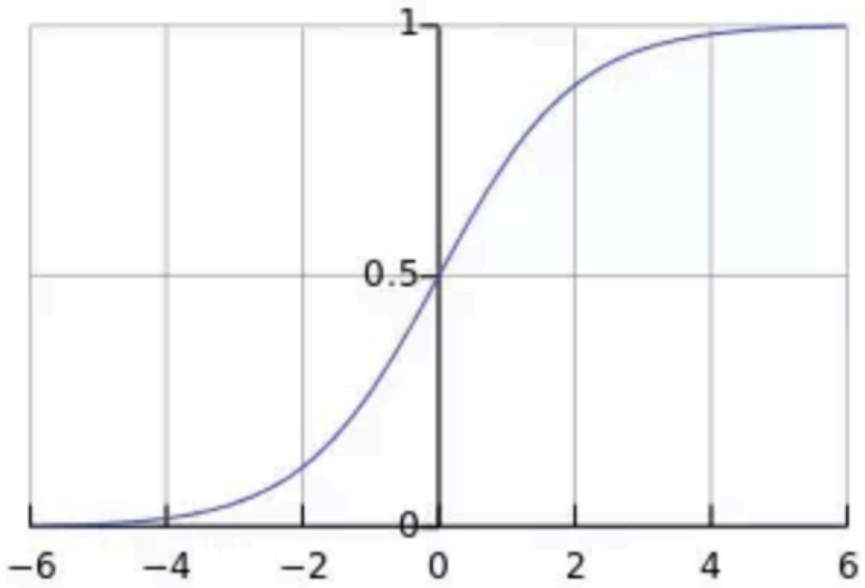
- Logistic Regression (LR) Overview
  - Bias-Variance Tradeoff
  - Hyper-parameter Tuning
  - LR tuning in Scikit-learn
- LR Model Evaluation
  - Accuracy, Recall, Precision, F1
  - ROC Curves
  - Confusion Matrix
- Code Example

Introduction

- Logistic Regression: Statistical model that uses a logistic function to model a categorical dependent variable
- Can be binary, multinomial, ordinal…
- Assume linear relationship between predictor variables

$$S(x) = \frac{1}{1+e^{-x}}$$

- Activation Function
- S(x) = output between 0 and 1
- X = input to the function which is the prediction from the algorithm



Hyperparameter

- **Regularization**: to shrink coefficients and reduce variance

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- L1: add “absolute value of magnitude” as penalty term to the loss function

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- L2: add “squared magnitude” as penalty term to the loss function

- Inverse of regularization strength C = 1/λ: to calibrate the penalization of large number of features to **prevent overfitting**
- **Lowering C** would result in **stronger regularizer**

Dataset: Breast Cancer

- Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.
- Target is Binary
- 569 inventories
- 30 features including radius, texture, smoothness, concavity…

```
df.columns.unique()
```

```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimensior
```

Bias-Variance Decomposition

- Assume Loss Function: $L(y,t) = \frac{1}{2}(y-t)^2$;

$$\mathbb{E}[(y-t)^2] = \underbrace{(t-E[y])^2}_{\text{bias}} + \underbrace{Var(y)}_{\text{variance}} + \underbrace{Var(t)}_{\text{irreducible error}}$$

- Split the expected loss into three terms:

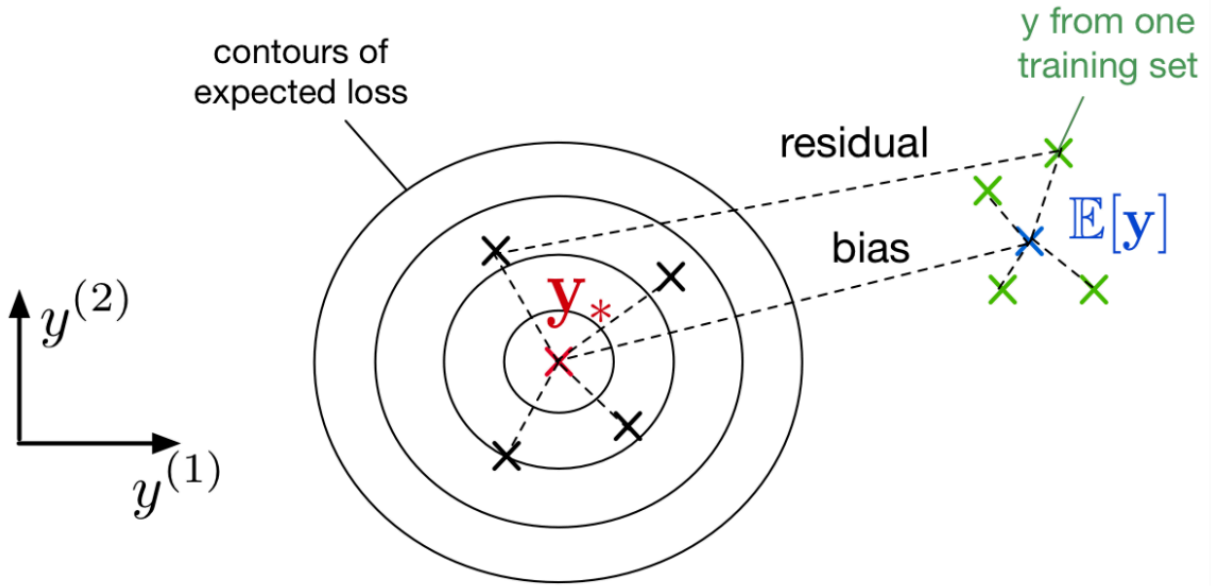
bias: how wrong the expected prediction is (corresponds to underfitting)

variance: the amount of variability in the predictions (corresponds to overfitting)

Bayes error: the inherent unpredictability of the targets (can't be avoided)

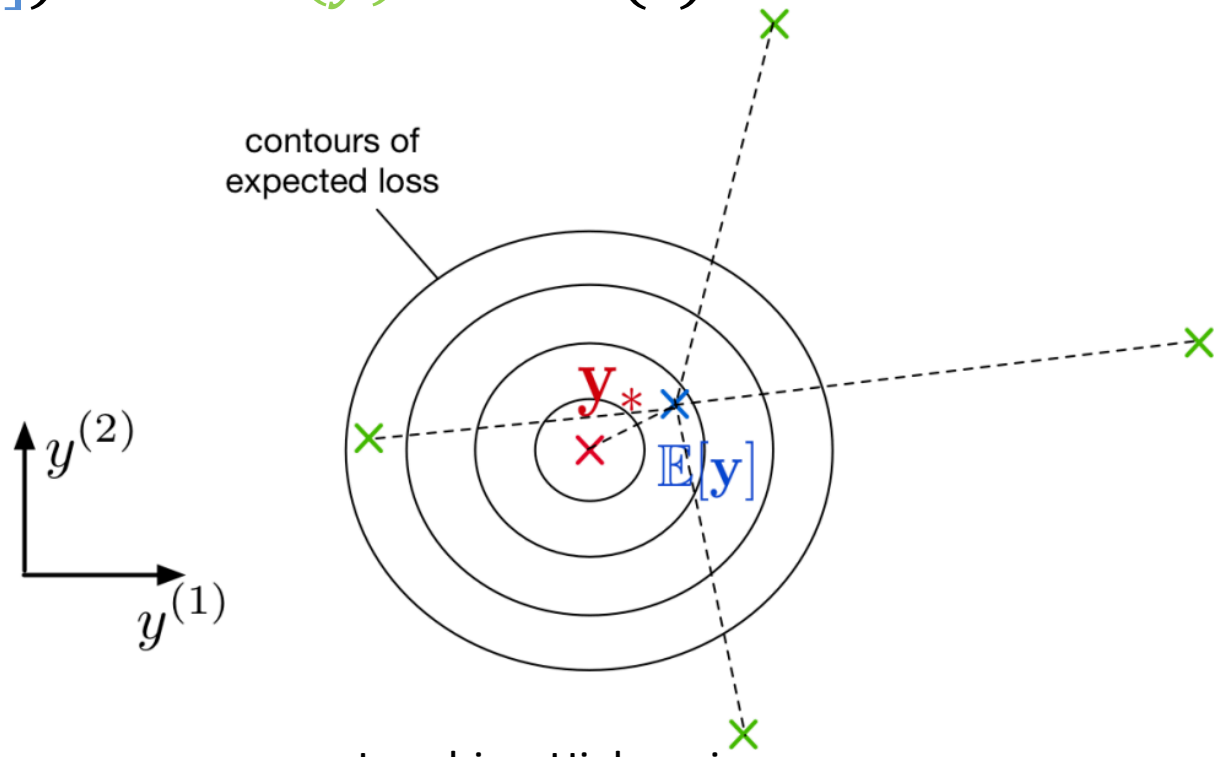
Visualizations in output space

$$\mathbb{E}[(y-t)^2] = (y^* - E[y])^2 + Var(y) + Var(t)$$



High bias, Low variance

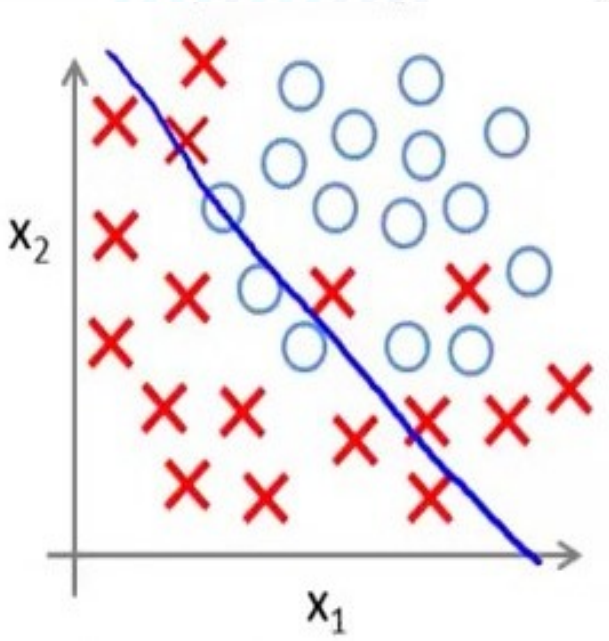
Underfitting



Low bias, High variance

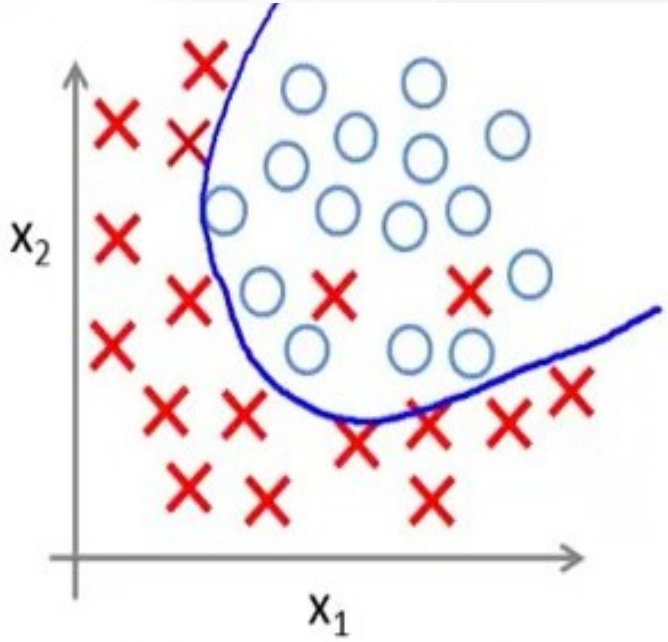
Overfitting

Bias-Variance in Logistic Regression



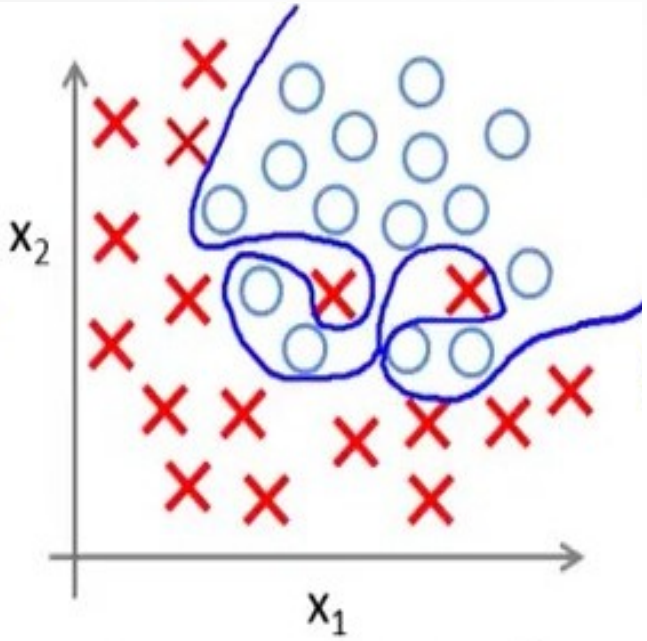
High Bias

The model is too simple to describe the data



Balanced Tradeoff

The model fits perfectly



High Variance

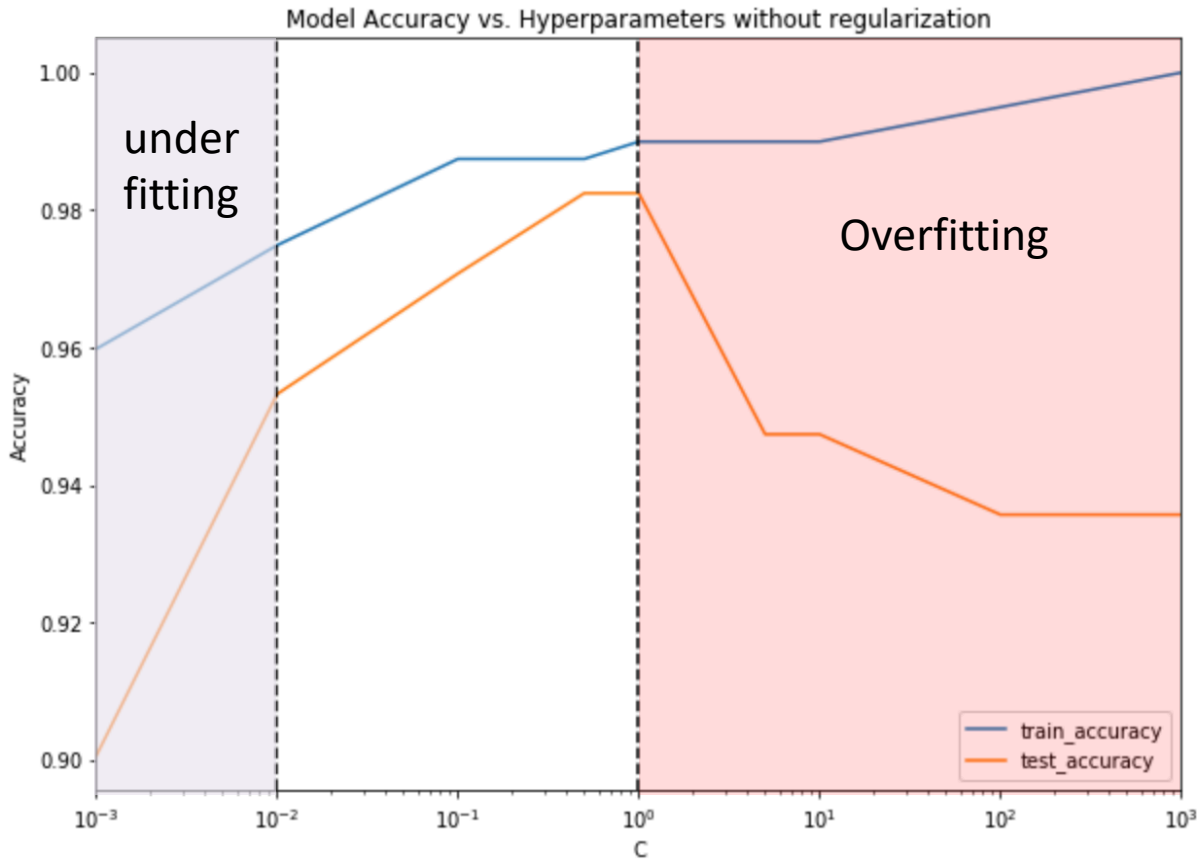
The model learns all the structure and may not generalize in new data

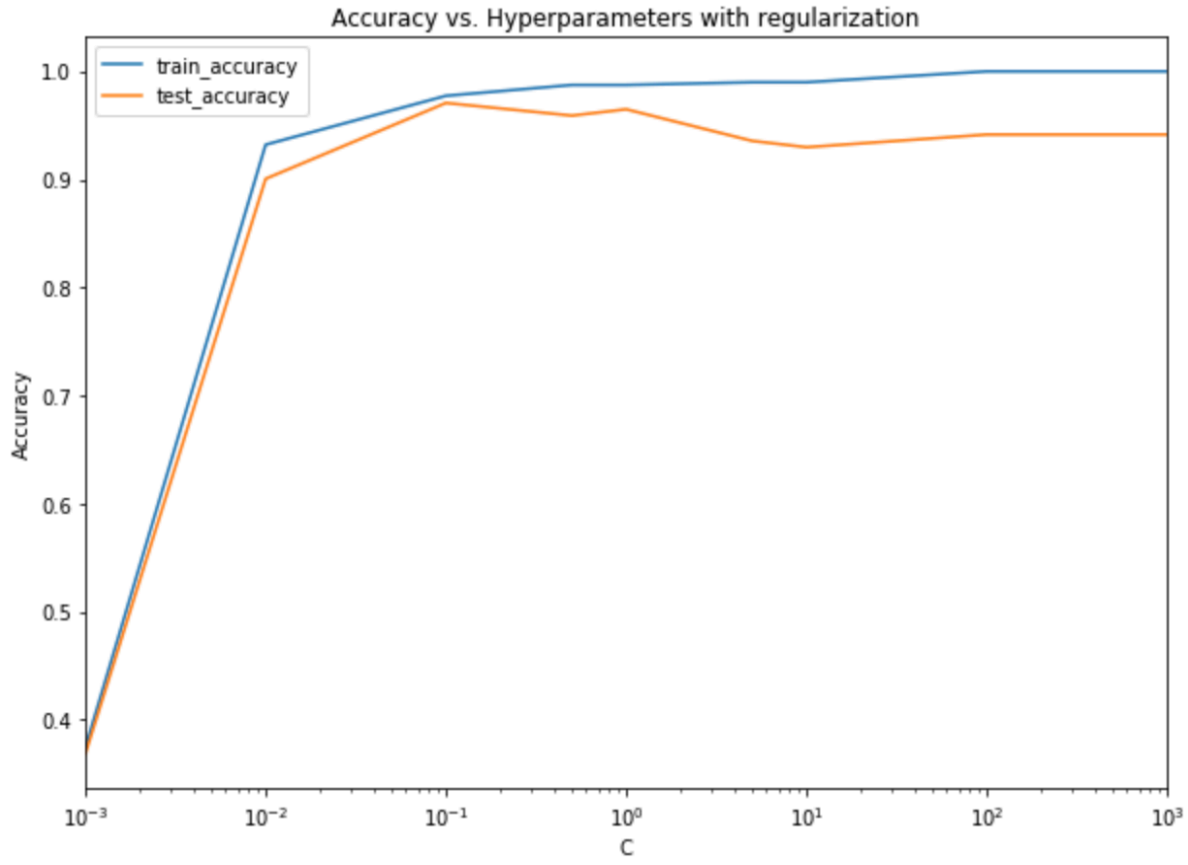
Hyperparameter Tuning

Tuning with/without L1 Regularization

How does ‘L1’ regularization factor effect our model?

Tune C directly in range: [0.001, 0.01, 0.1, 0.5, 1, 5, 10, 100, 1000]





Hyperparameter Tuning
Cross Validation

Manually tune C with K-Folds cross-validator:

```
[16] kfolders = KFold(n_splits=10, random_state=None, shuffle=False)
     kfolders.get_n_splits(X)
     best_model = None
     best_accuracy = 0
     for param in [0.001, 0.01, 0.1, 0.5, 1, 5, 10, 100,1000]:

         model = LogisticRegression(penalty='l1',C=param,solver='liblinear')
         test_accuracy=[]
         folder_count = 0

         for train_idx, test_idx in kfolders.split(X):

             X_train, X_test = X[train_idx], X[test_idx]
             y_train, y_test = y[train_idx], y[test_idx]

             model.fit(X_train, y_train)
             y_test_predict = model.predict(X_test)

             accuracy = accuracy_score(y_test, y_test_predict)
             test_accuracy.append(accuracy)
         if np.average(test_accuracy) > best_accuracy:
               best_model = model
               best_params = {'C':param}
               best_accuracy = np.average(test_accuracy)
```

```
[17] print(best_params)
```

```
{'C': 0.5}
```

scikit-learn module:

- **Grid Search with 10 cross validation**
  - Searches thoroughly from a specified set of parameters for an optimum.
  - Guarantees to find the most optimized parameter in the given set of parameters

```
[18] from sklearn.model_selection import GridSearchCV
     clf = LogisticRegression(penalty='l1',solver='liblinear')
     grid_values = {'penalty': ['l1'],'C':[0.001, 0.01, 0.1, 0.5, 1, 5, 10, 100,1000]}
     grid_clf_acc = GridSearchCV(clf, param_grid = grid_values,cv=10,scoring = 'accuracy')
     best_result = grid_clf_acc.fit(X_train, y_train)
```

```
[19] print('Best C:', best_result.best_estimator_.get_params()['C'])
```

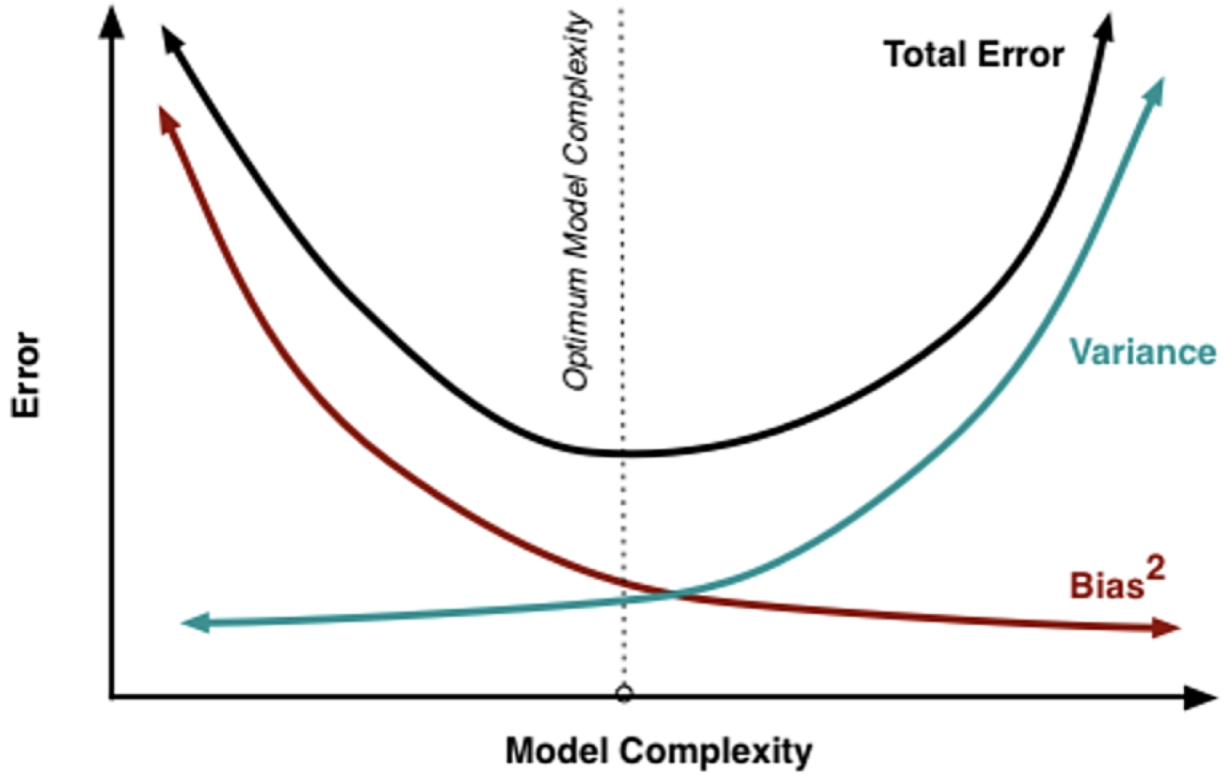
```
Best C: 0.5
```

The best scores and parameters can be inspected via printing:
*grid_result.best_score_*
*grid_result.best_params_*

Bias-Variance Tradeoff Implementations

--- Sklearn Module

Model Complexity



- Bias is reduced and variance is increased in relation to model complexity.
- Need to tradeoff bias and variance to achieve expected goals.
- In this case, choose optimum model complexity point to minimize the error.

Model Complexity Code Example

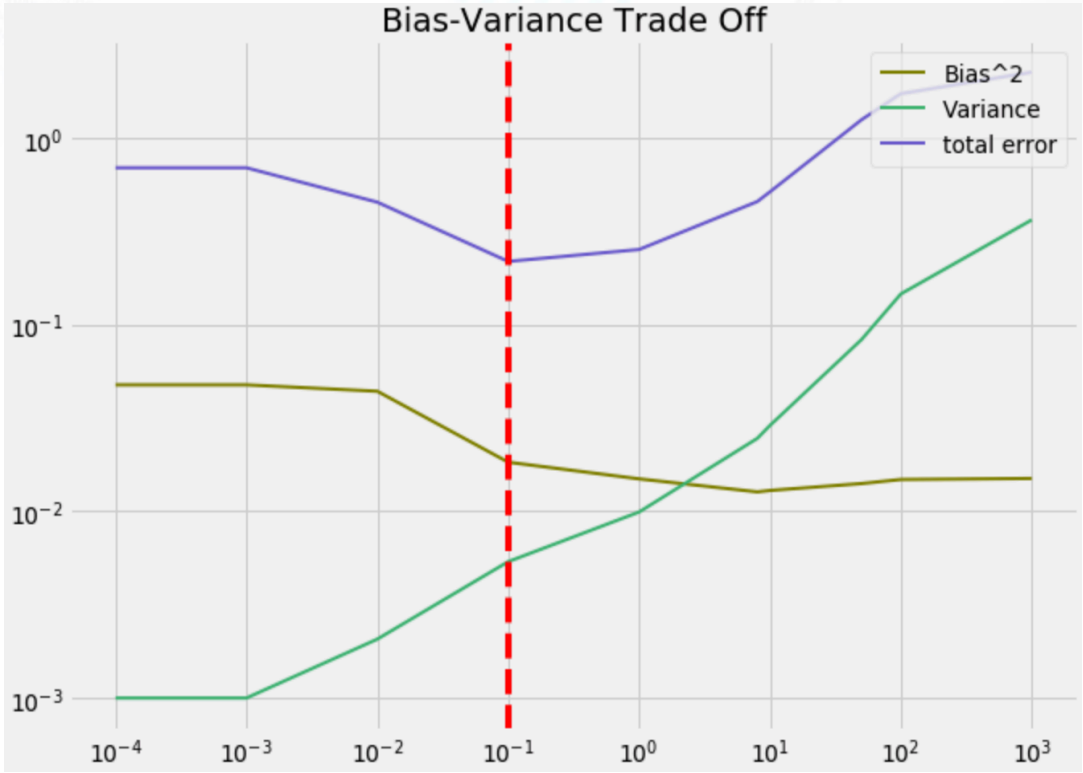
Given the dataset: sklearn breast cancer

```
from sklearn.metrics import log_loss
scaler = StandardScaler()
kfold = KFold(n_splits=10)
kfold.get_n_splits(X)
bias_list = []
var_list = []
total = []
C = [1e-4,1e-3,1e-2,1e-1,1,8,10,50,100,1000]
#  C = np.linspace(0.001,100,50)

for c_ in C:

        model = LogisticRegression(C=c_,penalty='l1',solver='liblinear')
        np_idx = 0
        variance=[]
        bias=[]
        MSE=[]
```

- Define Logistic model
- The set of parameters to tune based on Bias-Variance Tradeoff
- The number of K-folds to execute for cross-validation
- Scaling dataset to treat all features equally

Model Complexity Code Example

Looking for the lowest total error…




```
model.fit(X_train,  y_train)

#  X_test@model.coef_.reshape(-1)  =>  X_test  dot  model.coef_
#  which  calculates  the  values  for  Z.  recall  Z  =  sigma(x)
#  predicting  the  y test
pred=sigmoid(X_test@model.coef_.reshape(-1))

#  normally  you'd  filter  them,  but  since  we  used  log  cross  entropy  loss,
#  we  will  used  the  result  from  signoid  as  log_loss
#  takes  the  probability  of  the  predicted  label
#  pred[pred>=0.5]=1
#  pred[pred<0.5]=0
#  print(pred)

bias.append(bias_compute(pred,y_test))
variance.append(np.sqrt(np.var(model.coef_)))
loss.append(log_loss(y_test,pred))
```

Logistic Function

The best bias-variance tradeoff occurs at C = 0.1

Cross Entropy Loss

$$\mathcal{L}_{\mathrm{CE}}(y,t) = -t\log y - (1-t)\log 1 - y.$$

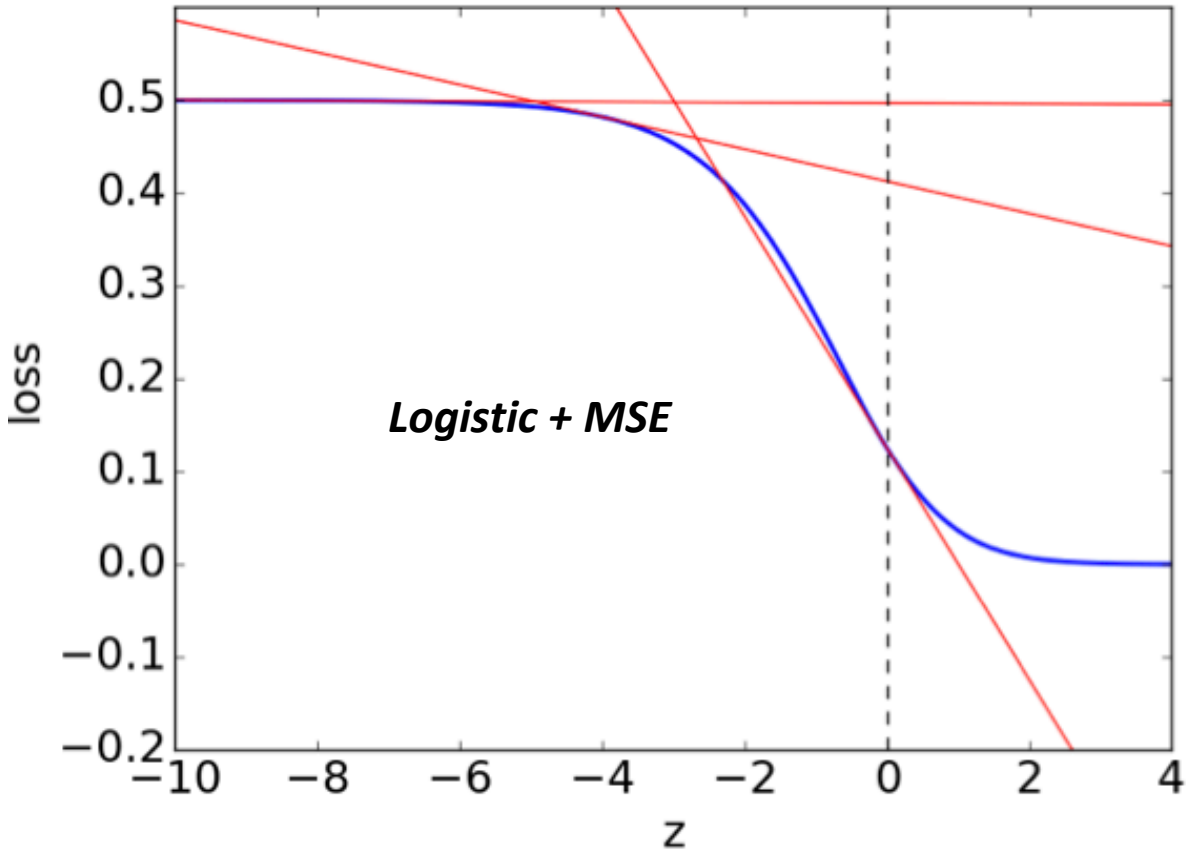
Bias-Variance Tradeoff Implementations

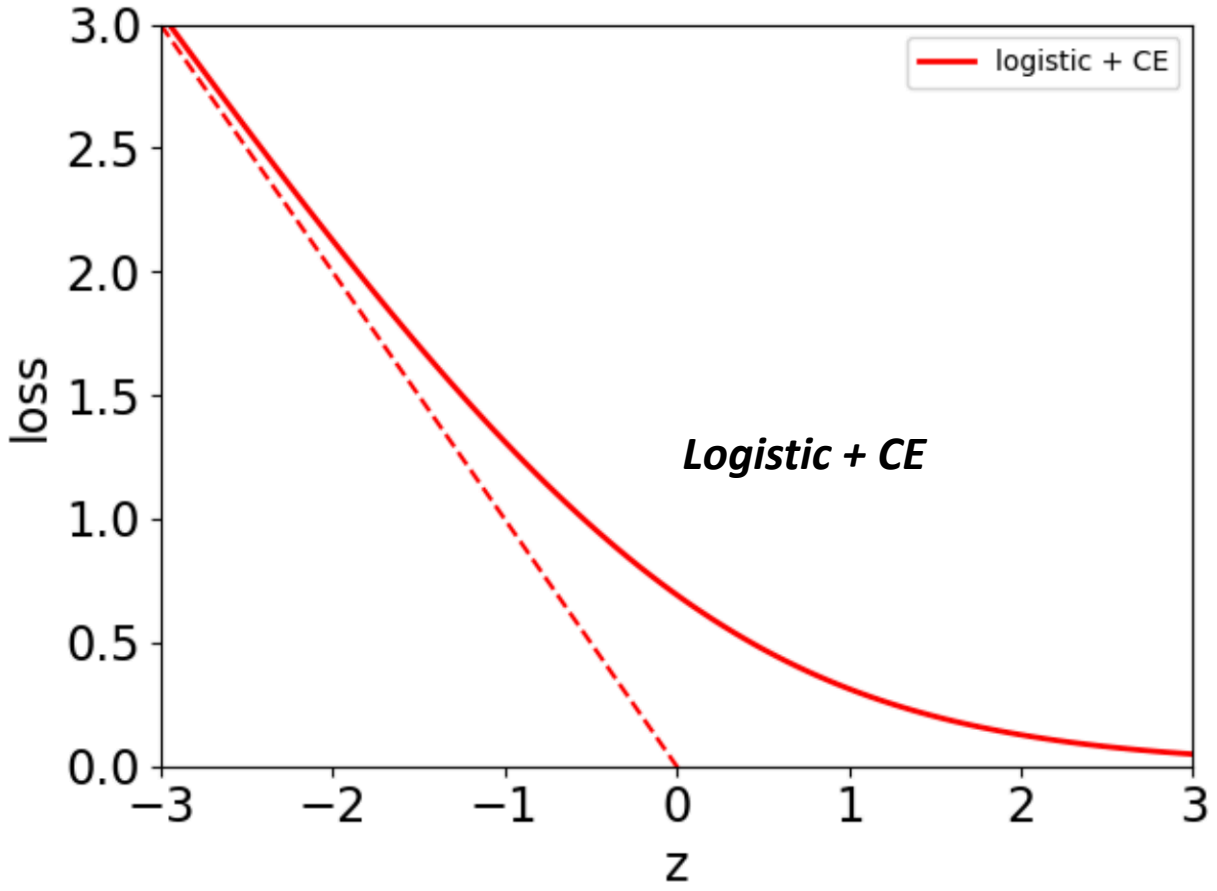
--- Direct Optimization

Cross-entropy Loss

$$\mathcal{L}_{\mathrm{CE}}(y,t) = -t \log y - (1-t) \log 1 - y.$$

$$\mathcal{L}_{\mathrm{LCE}}(z,t) = \mathcal{L}_{\mathrm{CE}}(\sigma(z),t) = t \log(1+e^{-z}) + (1-t) \log(1+e^{z})$$





Gradient Descent

- ***Cross − entropy Loss***:

$$z = w^T x + b$$

$$y = \sigma(z)$$

$$L_{CE} = -t log y - (1-t) log(1-y)$$

- ***Objective function***:

$$min_\theta \ \|\theta\|_1 + C \cdot J(\theta)$$

- ***Chain rule***:

$$\frac{\partial L_{CE}}{\partial y} = -\frac{t}{y} + \frac{1-t}{1-y}$$

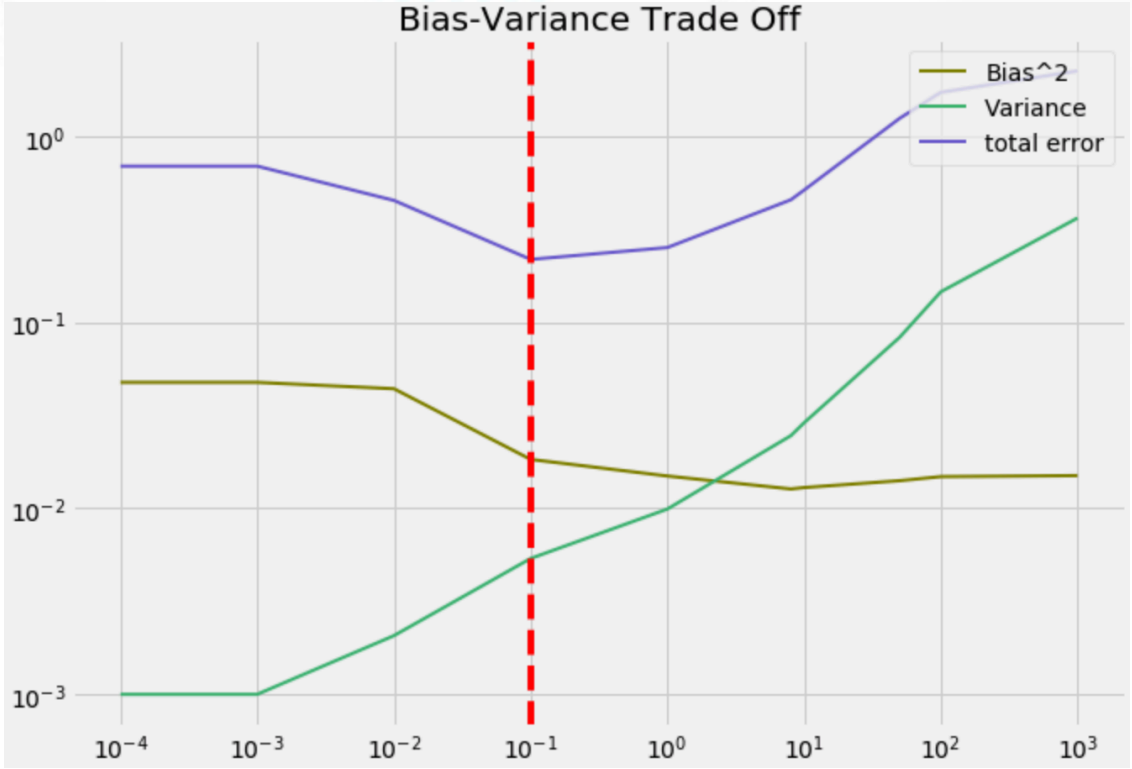
$$\frac{\partial L_{CE}}{\partial z} = \frac{\partial L_{CE}}{\partial y}\frac{\partial y}{\partial z} = \frac{\partial L_{CE}}{\partial y} y(1-y) = y - t$$

$$\frac{\partial L_{CE}}{\partial w_j} = \frac{\partial L_{CE}}{\partial z}\frac{\partial z}{\partial w_j} = \frac{\partial L_{CE}}{\partial z} x_j = (y-t) \cdot x$$

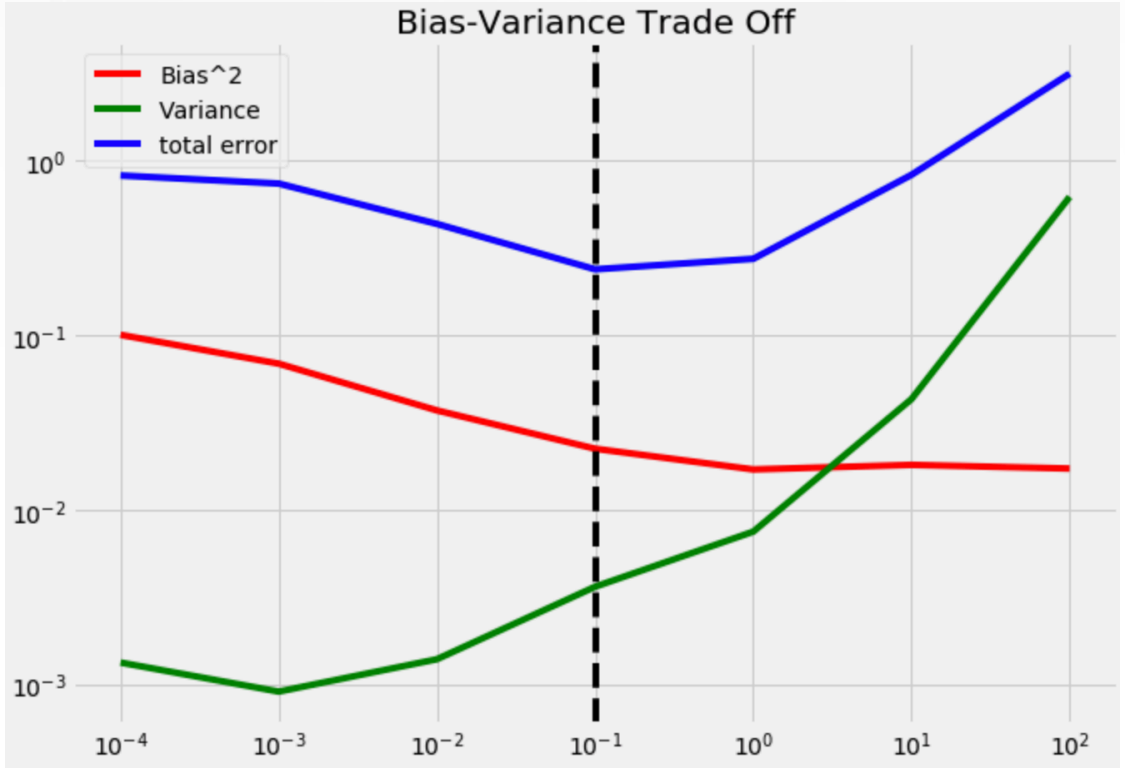
$$\frac{\partial L_{CE}}{\partial b} = \frac{\partial L_{CE}}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial L_{CE}}{\partial z}$$

```
    predictions = sigmoid(X @ theta)
#   error = (-t * np.log(predictions)) \
#   - ((1-t)*np.log(1-predictions))
    z=X @ theta
    error= t * np.logaddexp(0, -z)\
        + (1-t) * np.logaddexp(0, z)

    cost = 1/m * sum(error)
    Cost_l1= C*cost + sum(abs(theta))

    # compute gradient
    j_0 = C* (X.T @(predictions - t))[0] # for bias
    j_1 = C*(X.T @ (predictions - t))[1:]\
        + theta[1:]/abs(theta[1:])
    grad = np.vstack((j_0[np.newaxis,:],j_1))
```

Results Comparison



Sklearn Module



Direct Optimization

Evaluation Metrics

Confusion Matrix

		Predicted Class	
		1	0
Actual Class	1	TP=106	FN=2
	0	FP=3	TN=60

Confusion matrix:

- List the number of correct and wrong predictions
- A good model should generate high TPs and TNs

$$\textbf{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN} = 97\%$$

F1 Scores

		Predicted Class	
		1	0
Actual Class	1	TP=106	FN=2
	0	FP=3	TN=60

$$\textbf{Recall} = \frac{TP}{TP+FN} = 98\% \qquad \textbf{Precision} = \frac{TP}{TP+FP} = 97\%$$

$$\textbf{F1 Score} = \frac{2(Precision*Recall)}{Precision+Recall} = 98\%$$

Receiver Operating Curve

$$\text{TPR} = \frac{TP}{TP+FN} = 98.1\%$$

$$\text{FPR} = \frac{FP}{TN+FP} = 4.8\%$$

- The best model locates at the top left corner

Thanks for listening!

Questions?