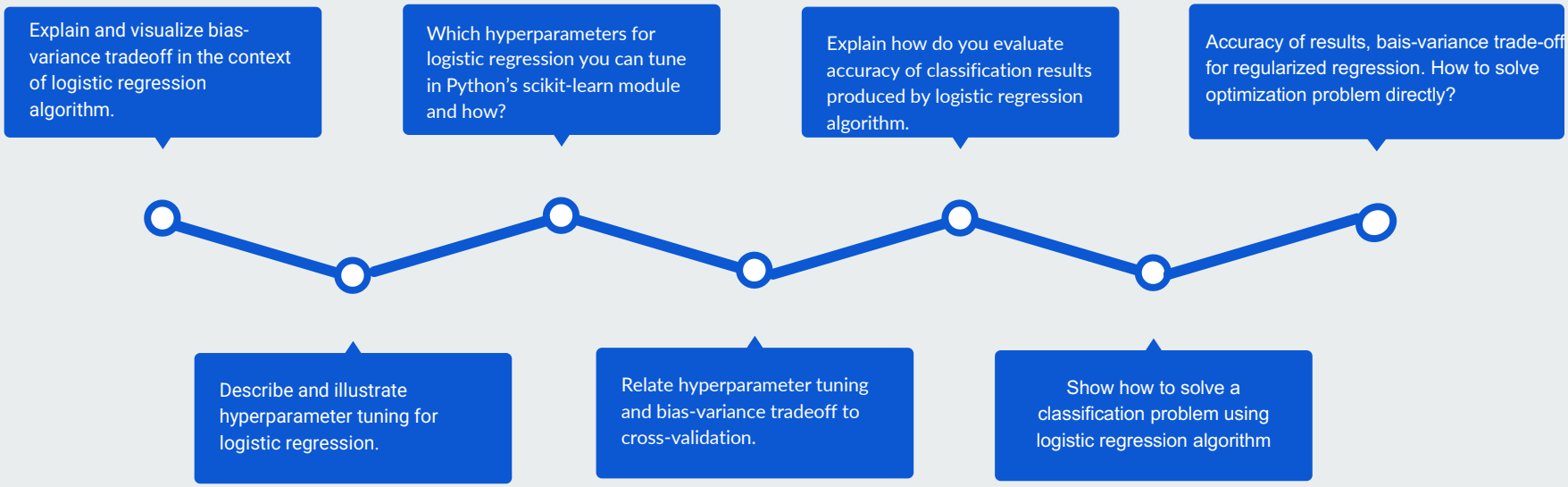
Logistic Regression Algorithm

Outline


Explain and visualize bias-variance tradeoff in the context of logistic regression algorithm.
Which hyperparameters for logistic regression you can tune in Python's scikit-learn module and how?
Explain how do you evaluate accuracy of classification results produced by logistic regression algorithm.
Accuracy of results, bais-variance trade-off for regularized regression. How to solve optimization problem directly?
Describe and illustrate hyperparameter tuning for logistic regression.
Relate hyperparameter tuning and bias-variance tradeoff to cross-validation.
Show how to solve a classification problem using logistic regression algorithm

Error in Prediction

Supervised machine learning algorithms suffer error during prediction introduced from the training dataset.

$$\text{MSE} = \mathbb{E}[(f - \hat{y})^2] = \mathbb{E}[(y + \epsilon - \hat{y})^2]$$

Using property $\mathbb{E}[X^2] = \mathbb{E}[X]^2 + \mathbb{V}[X]$

$$\text{MSE} = \mathbb{E}[(y - \hat{y})]^2 + \mathbb{V}[(y - \hat{y})] + \mathbb{E}[\epsilon]^2 + \mathbb{V}[\epsilon]$$

Using property $\mathbb{V}[(a - X)] = \mathbb{V}[(X)]$

Also $\mathbb{E}[\epsilon] = 0$ & $\mathbb{V}[\epsilon] = \sigma^2$

$$\text{MSE} = \frac{\mathbb{E}[(y - \hat{y})]^2}{Bias} + \frac{\mathbb{V}[(\hat{y})]}{Variance} + \frac{\sigma^2}{Irreducible}$$

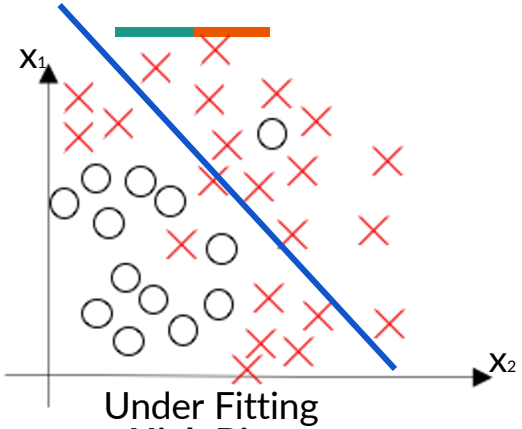
What is Bias?

- Bias is the difference between the average of actual output (y) and the predicted output ($\hat{y}$)
- Model does not fit well on the training dataset and result in under fitting
- Model with high bias oversimplifies the model.
- Model with bias has high error on training as well as test datasets.

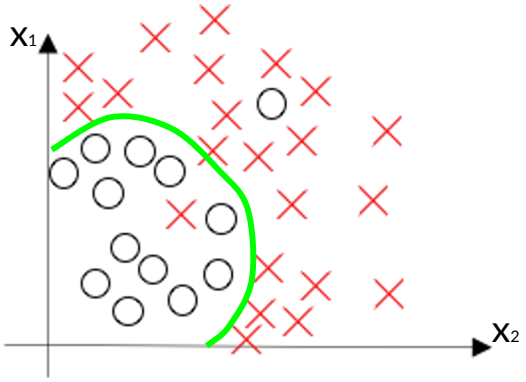
What is Variance?

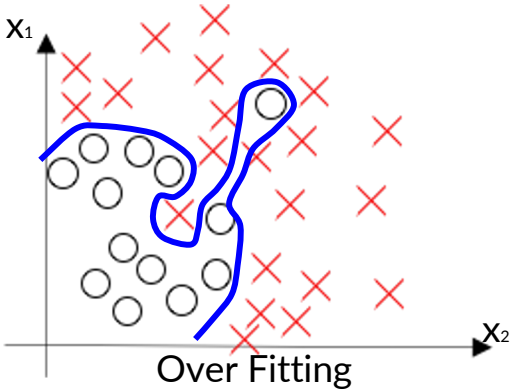
- Variance is the variability in the model prediction for a given input (x).
- Model fit too well on the training dataset and result in over fitting.
- Model does not generalize on any new data.
- Models with high Variance have higher accuracy on training dataset.
- Model has higher error on test/validation dataset.

Bias-Variance in Logistic Regression



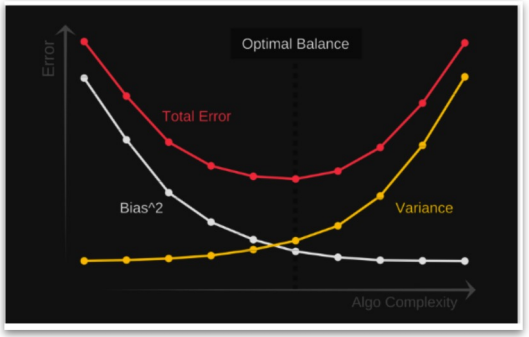
Under Fitting
High Bias





Over Fitting
High Variance

Training Set Error	15%	1%	1%
Testing Set Error	16%	1.5%	11%
	High Bias	**Low Bias and low Variance**	**High Variance**



Hyperparameter Tuning for Logistic Regression

Hyperparameters:

- Parameters whose values are used to control the training process are called hyperparameter.
- Value of the hyperparameters are typically set before the learning process begins.

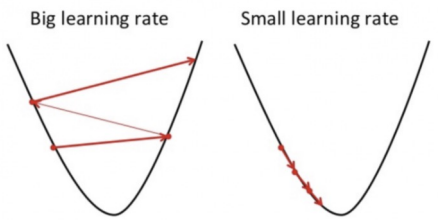
Hyperparameters in Logistic Regression

The cost function for logistic regression is given by:

$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)} + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))\right]$$

One way to $\min_\theta J(\theta)$ is by using Gradient Descent

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$



Note: The learning rate α is a hyperparameter which can be set during training

Hyperparameter Tuning for Logistic Regression

Regularization:

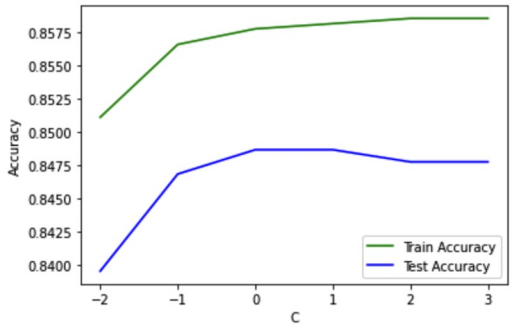
- One of the ways we can reduce variance is by applying a penalty to parameters.
- One of the ways is by using L1 Regularization.

L1 Regularization:

$$J(\theta) = \frac{1}{m}\|\theta\| - C\frac{1}{m}\sum_{j=1}^{m} y^{(j)} \log\left(h_\theta(x^{(j)})\right) + (1 - y^{(j)}) \log\left(1 - h_\theta(x^{(j)})\right)$$

- Where C is the inverse of a regularization strength. i.e. C = 1/**λ**
- The first term $\frac{1}{m}\|\theta\|$ penalizes large parameters in the mode

Note: C is a hyperparameter whose value can be selected.



Gradient of the cost function is

$$\frac{\partial J(\theta)}{\partial \theta_i} = \frac{1}{m} + C\frac{1}{m}\sum_{j=1}^{m}(h_\theta(x^{(j)}) - y^{(j)})x_i^{(j)}$$

Hyperparameter Tuning

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None)
```

- **C**: trade-off parameter of logistic regression that determines the strength of the regularization. Basically smaller C specify stronger regularization.
  - float, default=1.0
- **class_weight**: It penalizes mistakes in samples of class[i] with class_weight[i].
  - dict or 'balanced', default=None
- **solver**: Algorithm to use in the optimization problem.
  - {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default='lbfgs'
- penalty: Used to specify the norm used in the penalization.
  - {'l1', 'l2', 'elasticnet', 'none'}, default='l2'
- max_iter: Maximum number of iterations taken for the solvers to converge. Larger the number of iteration, the more accurate it will get.
  - int, default=100

Strategies for Hyperparameter Tuning in sklearn –

Grid Search

class sklearn.model_selection.**GridSearchCV**(*estimator, param_grid, *, scoring=None, n_jobs=None, iid='deprecated', refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False*) ¶ [source

- Exhaustively generates candidates from a grid of parameter values specified with the param_grid parameter.
- It will find the optimal combination in the defined sets of parameters, but with a longer processing time.
- GridSearchCV()

Strategies for Hyperparameter Tuning in sklearn –

Random Search

class `sklearn.model_selection.` **`RandomizedSearchCV`**(*estimator, param_distributions, *, n_iter=10, scoring=None, n_jobs=None, iid='deprecated', refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', random_state=None, error_score=nan, return_train_score=False*) [sour

- It searches the specified subset of hyperparameters randomly instead of exhaustively.
- Decreases process time, but we might not find the optimal combination of hyperparameters.
- RandomizedSearchCV()

Cross Validation

```
sklearn.model_selection.cross_val_score(estimator, X, y=None, *, groups=None, scoring=None, cv=None, n_jobs=None,
verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score=nan)                                              [
```

- Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.
- we split the data into multiple sets, perform the training and testing, and compare the resulting scores.
- cross_val_score()

Hyperparameter Tuning With Cross-Validation

```
accuracy_dic = {}
for i in hyper:
    c = 10**i
    classifier = LogisticRegression(C=c, penalty='l1', solver='saga')
    classifier.fit(X_train, y_train)
    accuracies = cross_val_score(estimator=classifier, X=X_train, y = y_train, cv=10)
    accuracy_dic[c] = accuracies.mean()
print(accuracy_dic)
print("The hyperparameter that leads to highest cross validation score is: ")
print(max(accuracy_dic, key=accuracy_dic.get))
```

```
{0.01: 0.8511136642156863, 0.1: 0.8546292892156864, 1: 0.8561902573529412, 10: 0.8557996323529412, 100: 0.8557996323529412, 1000: 0.8557996323529412}
The hyperparameter that leads to highest cross validation score is:
1
```

- For each value of C, we evaluate the accuracies of the model by computing the cross-validation score.
- Compare the score to determine which value of C gives the most accurate result.

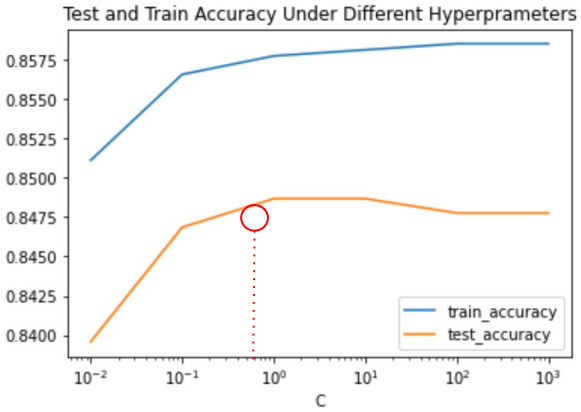
Bias-variance tradeoff with cross-validation

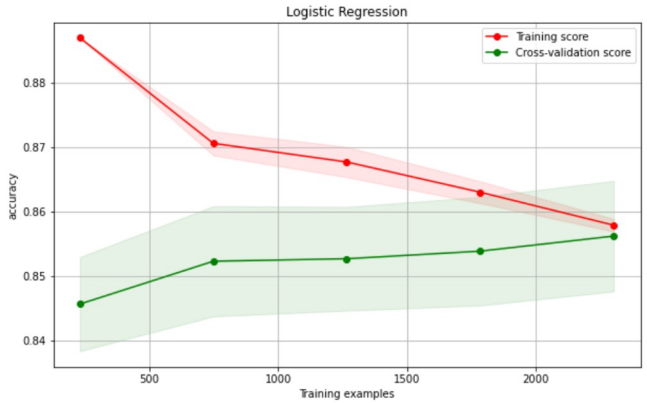
- Bias-variance tradeoff
  - Bias: difference between the actual output and the predicted output.
  - Variance: variability in the model prediction for a given input.
- For cross-validation:
  - Less subgroups: higher bias, lower variance
  - More subgroups: lower bias, higher variance

Data Set for Logistic Regression

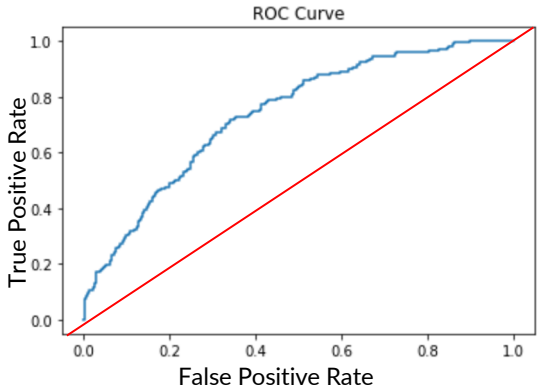
- Coronary Heart Disease Risk Prediction (heartDiseaseData.csv)
- Using males, age, education, smoker or non smoker, cigarettes smoked per day, and traditional risk factors such as blood pressure, cholesterol, diabetes, glucose levels, etc.
- 3656 records and 15 fields
- Dependent variables are **binary** only (0, 1)
- Classification goal: 10-year CHD risk prediction in males aged 32-70 considering various health factors

Accuracy Evaluation

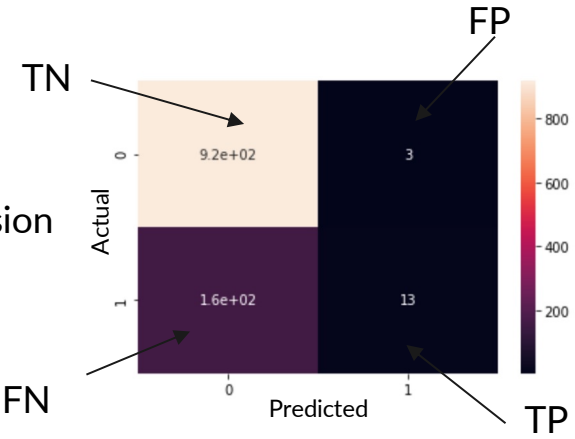




ROC Curve



Confusion Matrix



Directly solving the optimization problem

- from scipy.optimize import fmin_tnc
- Minimizes a function with variables subject to bounds
- Uses gradient information in a truncated Newton algorithm; wraps a C implementation of the algorithm

```
def fit(x, y, C):
    # Use scipy.optimize.fmin_tnc to find parameters minimize the cost function
    theta = np.zeros((1, X.shape[1]+1))
    opt_weights = fmin_tnc(func=cost_function, x0=theta,
                  fprime=gradient,args=(x, y, C))
    parameter = opt_weights[0].T
    return parameter
```

Thank you for listening!