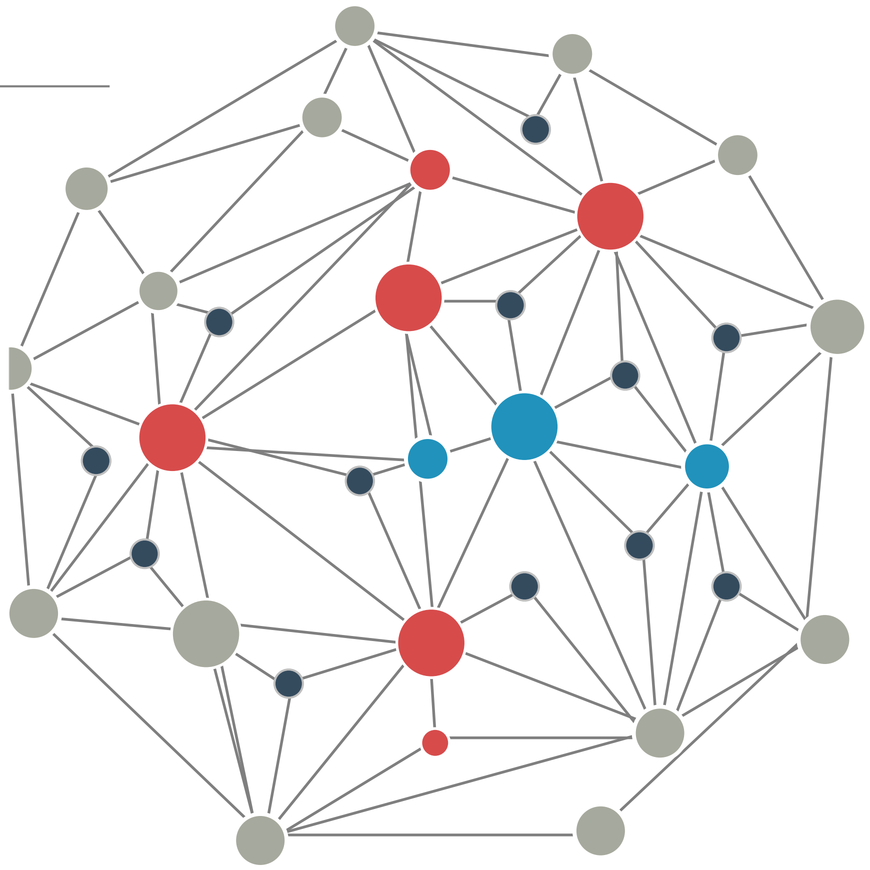
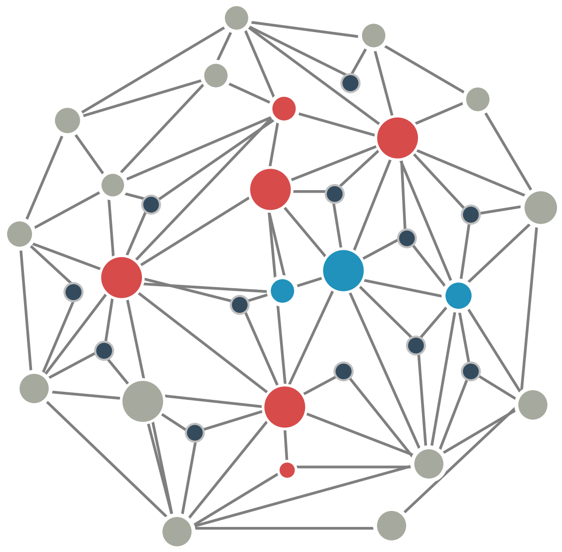
CLASSIFICATION BY LOGISTIC REGRESSION

Group 15 In-class Presentation





/01

Bias-Variance Tradeoff

- **Mathematical Interpretation.**
- **Graphical illustration.**
- **Tradeoff in terms of model complexity**

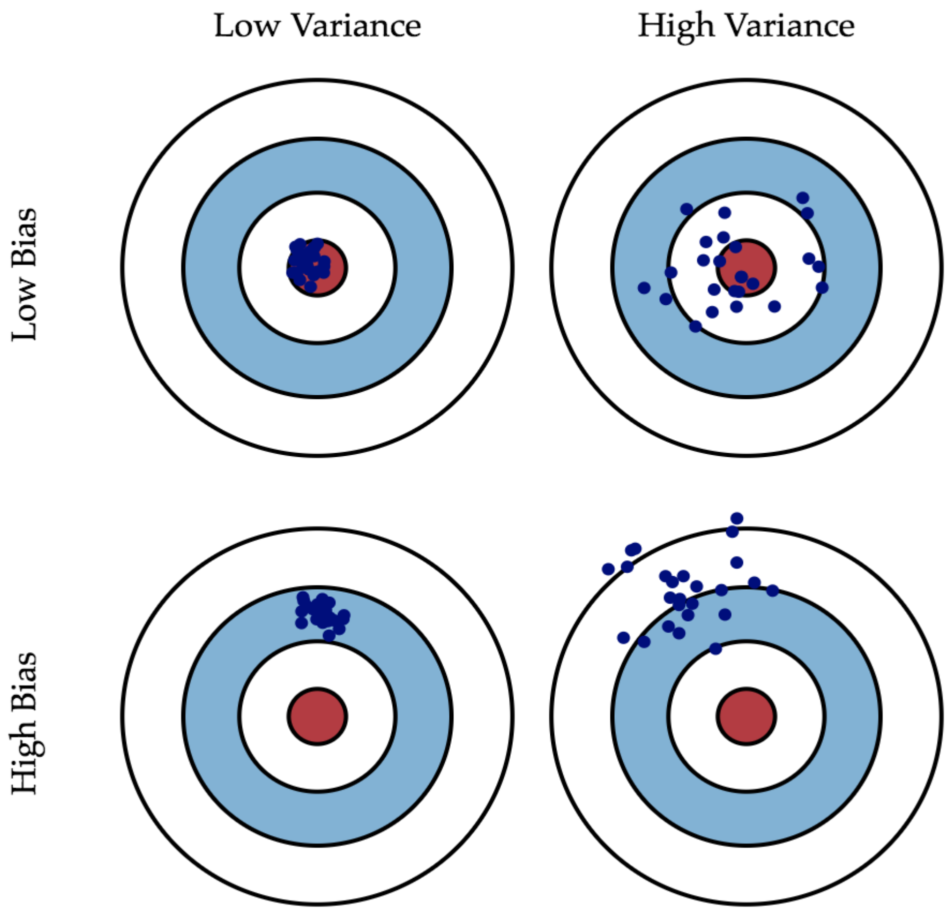
Mathematical Interpretation

For the regression problem, we can define and decompose it bias and variance components.

$$ERROR = \sigma^2 + \mathbb{E}_D\left(\left(h(x) - \mathbb{E}_D\big(h(x)\big)\right)^2\right) + \left(\mathbb{E}_D\big(h(x)\big) - f(x)\right)^2$$

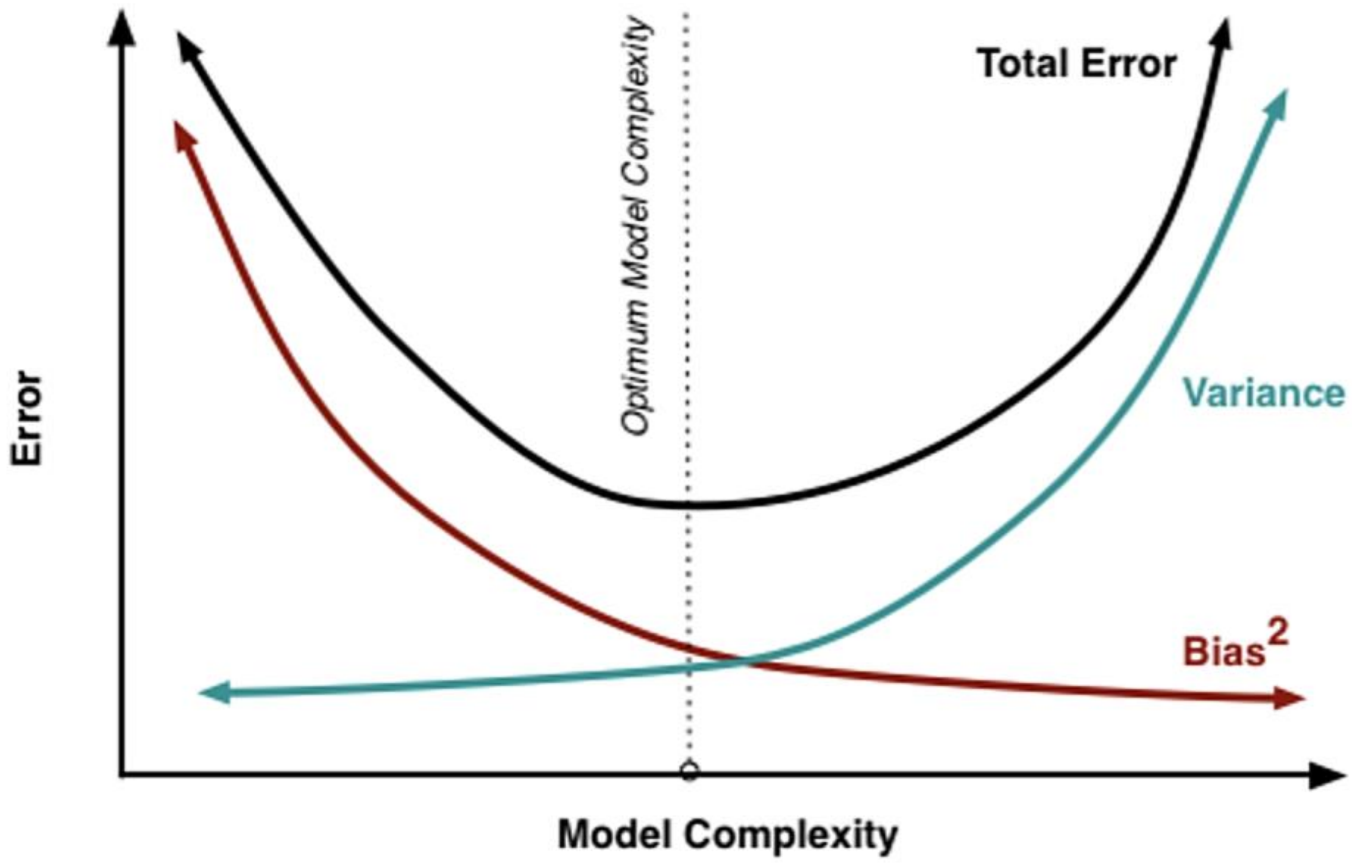
Var ***Bias^2***

Graphical illustration

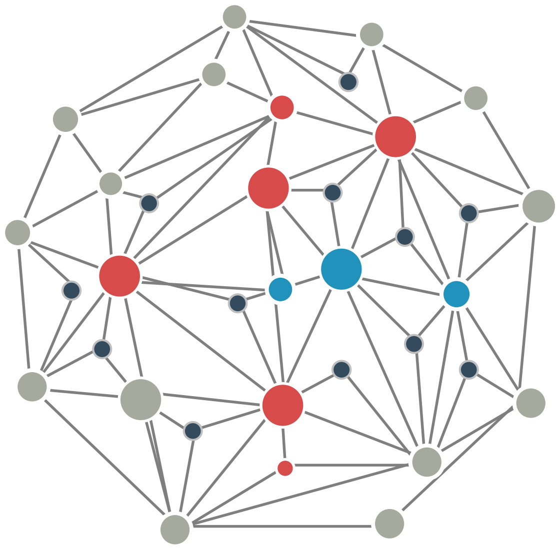


- A graphical visualization of bias and variance using a bulls-eye diagram
- Each hit represents an individual realization of our model
- The center of the target is a model that perfectly predicts the correct values
- Top left case: perfect but impossible to realize due to imperfect models and finite data
- In real world: tradeoff between minimizing bias and minimizing variance

Tradeoff in terms of 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.

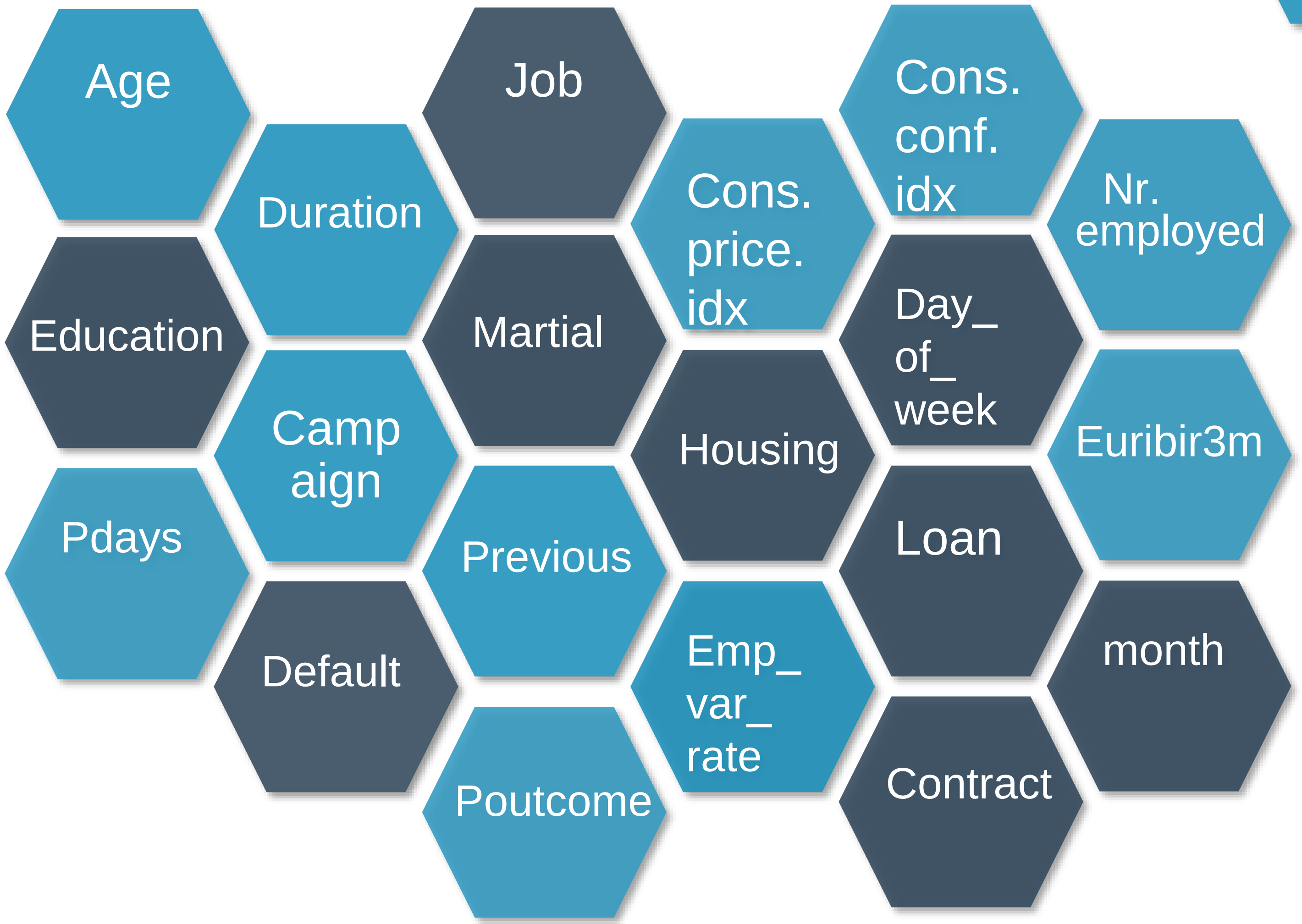


/02

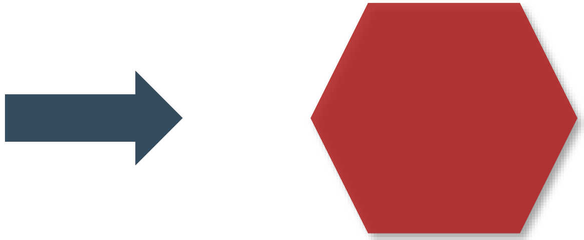
Overview of Dataset

Overview of Dataset

Numeric Categorical

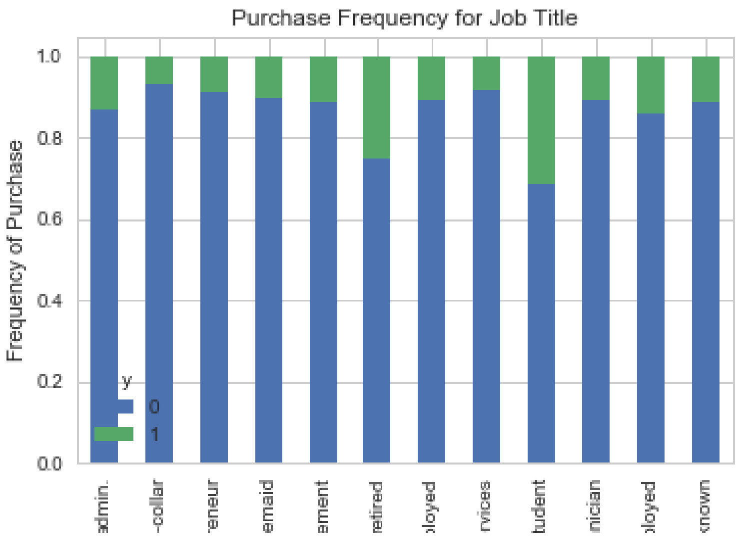


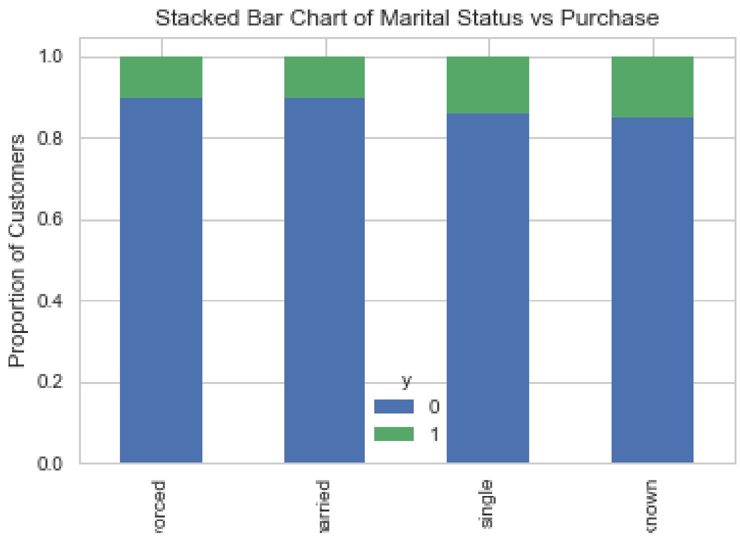
- Including 41188 samples
- 20 features

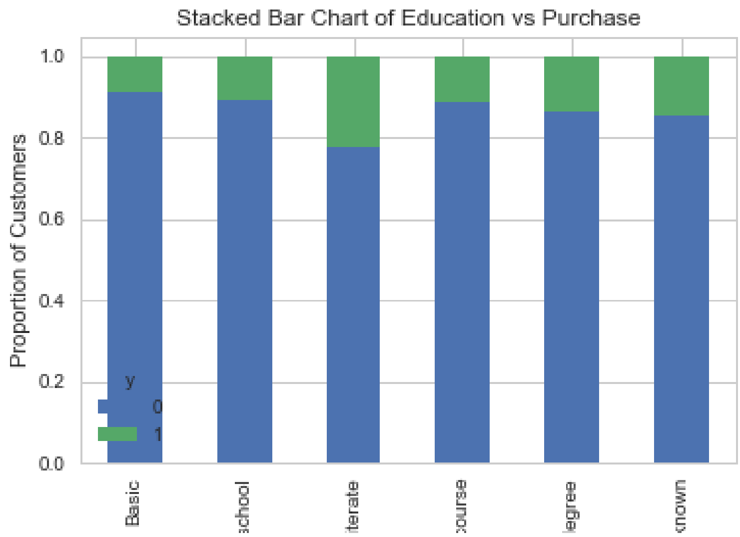


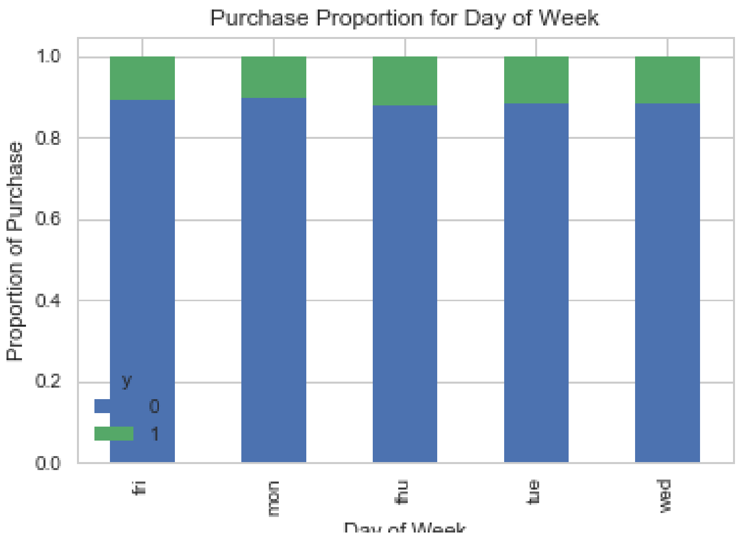
- Predictor: Has the client subscribed a term deposit?

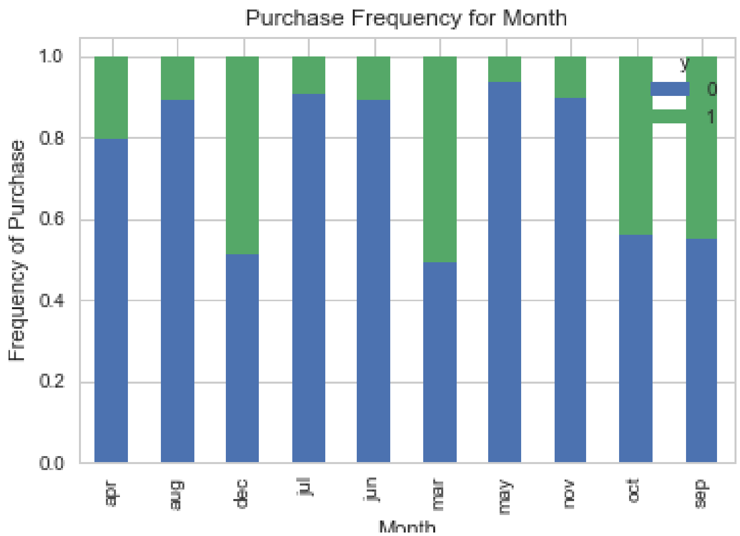
Data Visualization

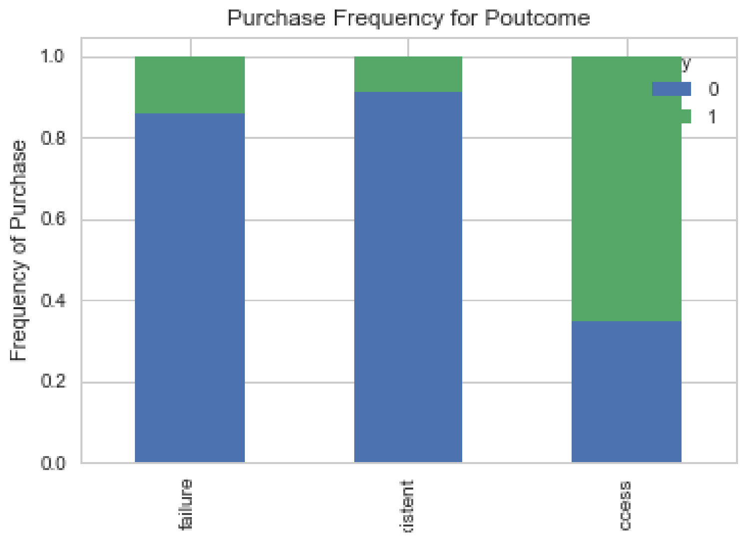


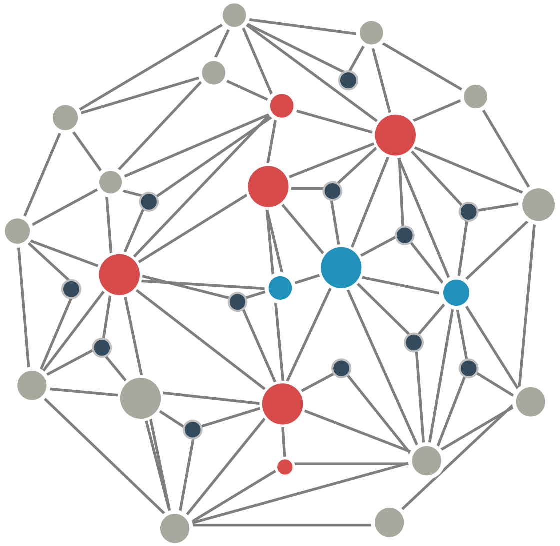












/03

Data Preprocessing

General Idea

1. Convert categorical into dummy variables
2. Over-sampling using SMOTE
3. Recursive Feature Elimination

Convert categorical into dummy variables

In [13]:
```
cat_vars=['job','marital','education','default','housing','loan','contact','month','day_of_week','poutcome']
for var in cat_vars:
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(bankData[var], prefix=var)
    dummy=bankData.join(cat_list)
    bankData=dummy

data_vars=bankData.columns.values.tolist()
to_keep=[i for i in data_vars if i not in cat_vars]
```

Use get_dummies from pandas

In [14]:
```
bank_final=bankData[to_keep]
bank_final.head(3)
```

Out[14]:

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	...	month_oct	month_sep	day_of_week
0	44	210	1	999	0	1.4	93.444	-36.1	4.963	5228.1	...	0	0	
1	53	138	1	999	0	-0.1	93.200	-42.0	4.021	5195.8	...	0	0	
2	28	339	3	6	2	-1.7	94.055	-39.8	0.729	4991.6	...	0	0	

3 rows × 62 columns

Over-sampling using SMOTE

- Works by creating synthetic samples from the minor class (no-subscription) instead of creating copies.
- Randomly choosing one of the **k-nearest-neighbors** and using it to create a similar, but randomly tweaked, new observations.

In [30]:
```
X = bank_final.loc[:, bank_final.columns != 'y']
y = bank_final.loc[:, bank_final.columns == 'y']

from imblearn.over_sampling import SMOTE
os = SMOTE(random_state=0)
X_use, X_keep, y_use, y_keep = train_test_split(X, y, test_size=0.3, random_state=10)

columns = X_use.columns
os_data_X,os_data_y=os.fit_sample(X_use, y_use)
os_data_X = pd.DataFrame(data=os_data_X,columns=columns)
os_data_y= pd.DataFrame(data=os_data_y,columns=['y'])

# We can check the numbers of our data
print("length of oversampled data is ",len(os_data_X))
print("Number of no subscription in oversampled data",len(os_data_y[os_data_y['y']==0]))
print("Number of subscription",len(os_data_y[os_data_y['y']==1]))
print("Proportion of no subscription data in oversampled data is ",len(os_data_y[os_data_y['y']==0])/len(os_data_X))
print("Proportion of subscription data in oversampled data is ",len(os_data_y[os_data_y['y']==1])/len(os_data_X))
```

Over-sampling using SMOTE

The output is:

```
length of oversampled data is  51232
Number of no subscription in oversampled data 25616
Number of subscription 25616
Proportion of no subscription data in oversampled data is  0.5
Proportion of subscription data in oversampled data is  0.5
```

- Now we have a perfect **balanced** data!

Recursive Feature Elimination

- Recursive Feature Elimination (RFE) is based on the idea to repeatedly construct a model and choose either the best or worst performing feature, setting the feature aside and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted.

- The goal of RFE is to select features by recursively considering smaller and smaller sets of features.

In [32]:

```
bank_final_vars=bank_final.columns.values.tolist()
X=[i for i in bank_final_vars if i!='y']

from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

logreg = LogisticRegression()
rfe = RFE(logreg, 20) # where you set number of variables to choose
rfe = rfe.fit(os_data_X, os_data_y.values.ravel())
```

Bias and Variance

```
In [18]: def bias(y_predict,y):
             y = y.values
             avg = np.average(y_predict)
             return np.average(avg-y)**2
```

```
In [19]: def variance(y_predict):
             return np.var(y_predict)
```

- Bias: use predict labels and ground truth labels to compute bias
- Variance: directly compute the variance of predict labels

Hyperparameter Tuning

```
In  [20]: from sklearn.model_selection import KFold

def hyperparameter(X, y, penal, K):
    # result_list is a list of tuples (num_features, train_accuracy, test_accuracy)
    # where numFeatures is the number of words used as features
    result_list = []
    best = {'train':0,'test':0,'model':None,'param':None}
    for param in np.logspace(-5, 1.0, 30):
        kf = KFold(n_splits=K)
        kf.get_n_splits(X)
        param_keep = []

        for train_index, test_index in kf.split(X):
            X_train, X_test = X.iloc[train_index], X.iloc[test_index]
            y_train, y_test = y.iloc[train_index], y.iloc[test_index]

            clf = LogisticRegression(C=param, solver="liblinear", penalty=penal).fit(X_train, y_train)
            y_train_predict = clf.predict(X_train)
            y_test_predict = clf.predict(X_test)

            train_accuracy = accuracy_score(y_train, y_train_predict)
            test_accuracy = accuracy_score(y_test, y_test_predict)
            #scores = clf.predict_proba(X_train)

            bia = bias(y_train_predict,y_train)
            var = variance(y_train_predict)
            total = bia + var
            param_keep.append((train_accuracy, test_accuracy, bia, var,total))
```

Use grid search to tune inverse weight of L1 regularization

Split data into K=4 folds for cross validation

Calculate bias and variance

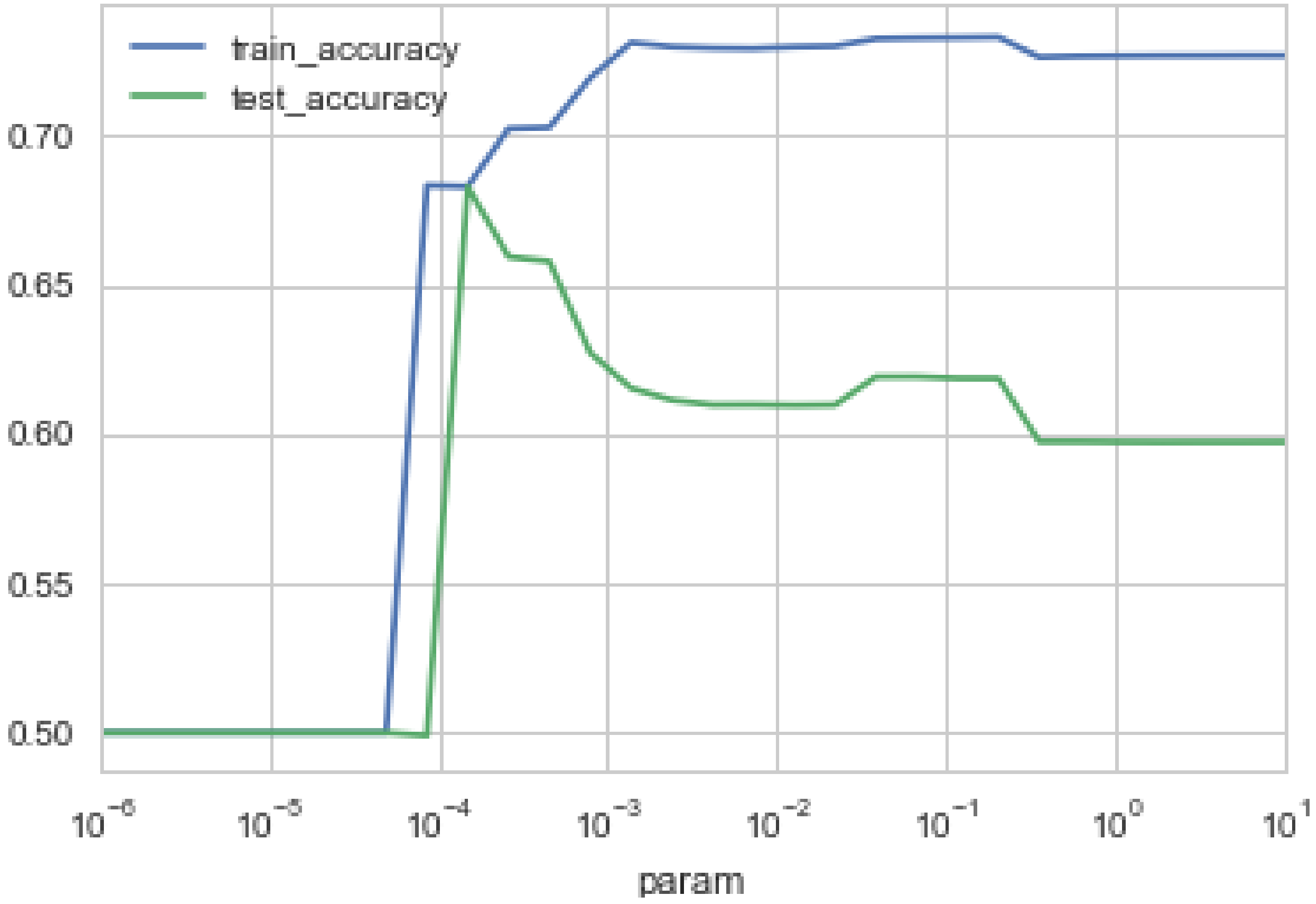
Tuning Hyperparameter

```
    result_list.append((param, sum([i[0] for i in param_keep])/K, sum([i[1] for i in param_keep])/K,\
                        sum([i[2] for i in param_keep])/K, sum([i[3] for i in param_keep])/K, sum([i[4] for i in param_keep])/K ))

    if result_list[-1][2]>best['test']:
        best['train'] = result_list[-1][1]
        best['test'] = result_list[-1][2]
        best['model'] = clf
        best['param'] = param
result_df = pd.DataFrame(result_list, columns=["param", "train_accuracy", "test_accuracy", "bia", "var","total"])
return result_df, best
```

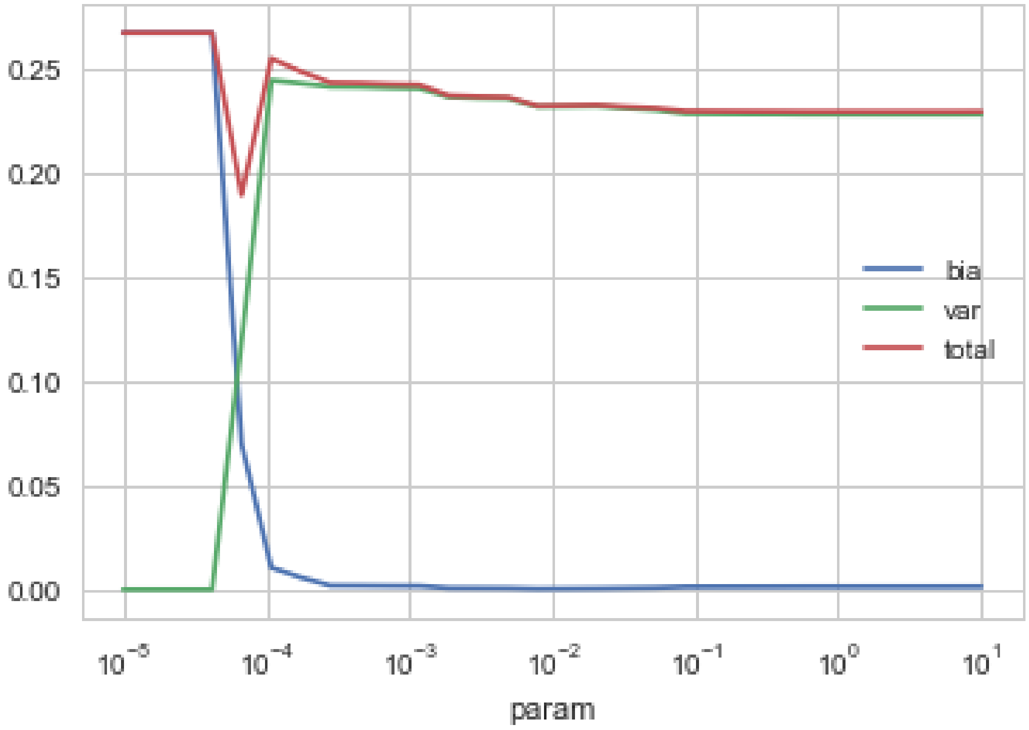
Store the model with the highest test accuracy

Relationship between accuracy and L1 regulation



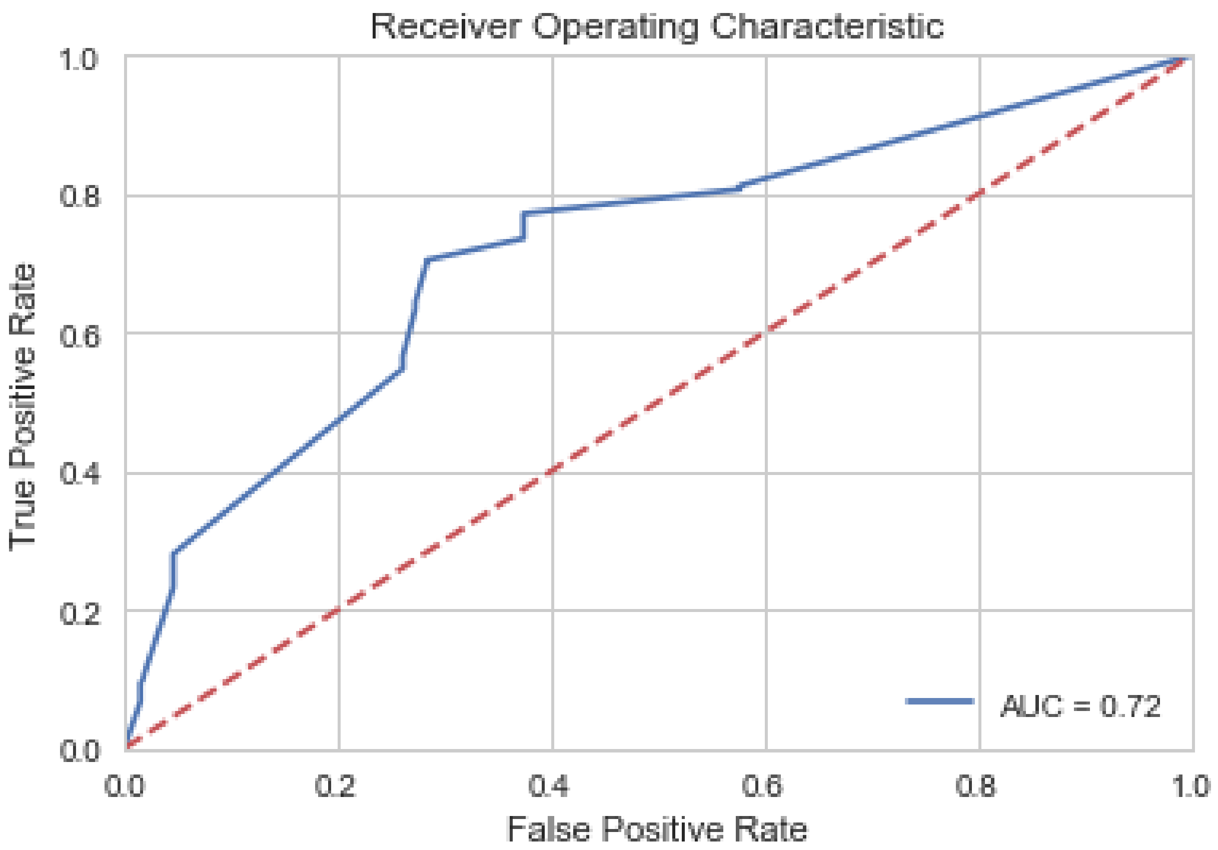
x-axis is inverse weight of L1 regularization, indicates how "strong" L1 regularization

Bias and Variance Tradeoff



- Bias grow when we try to reduce variance and the same case for variance.
- Can not achieve low bias and low variance at the same time
- Need to tradeoff bias and variance in real model to improve the performance

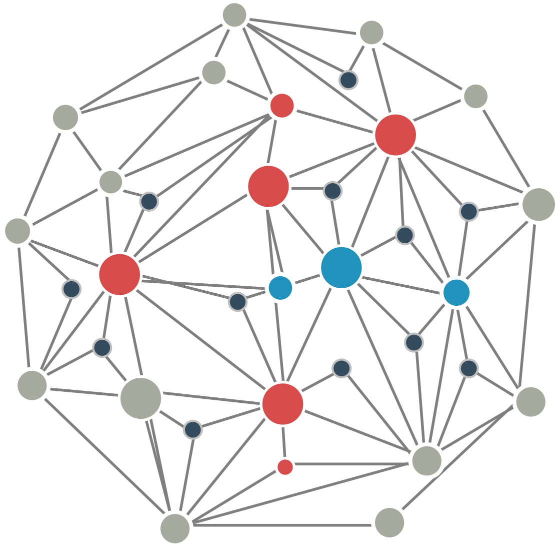
ROC plot and corresponding AUC



Precision Score: 66.620%

Recall Score: 73.806%

F1 Score: 70.029%



/ADDITION

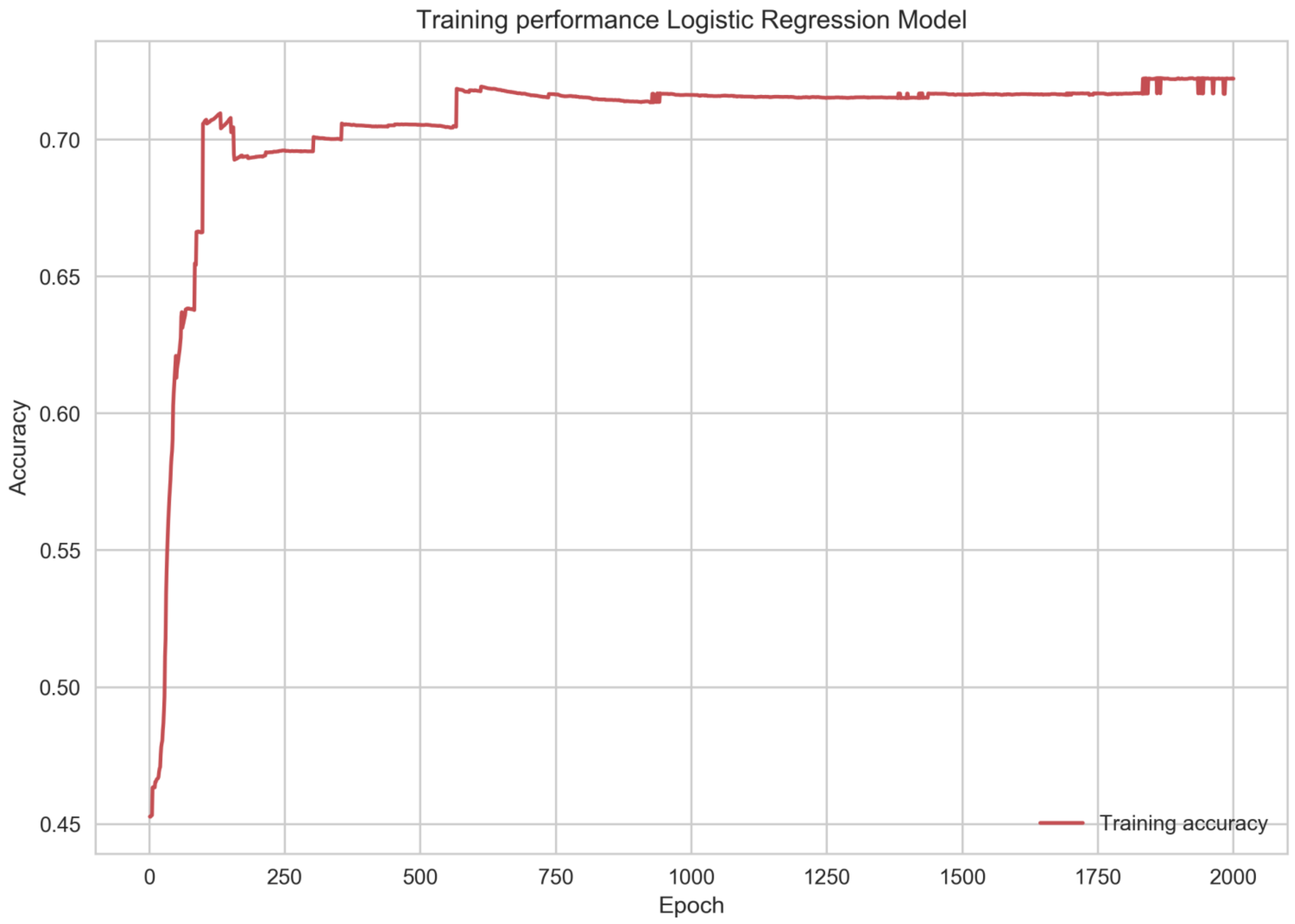
Bias-variance Tradeoff in TensorFlow

Code in TensorFlow

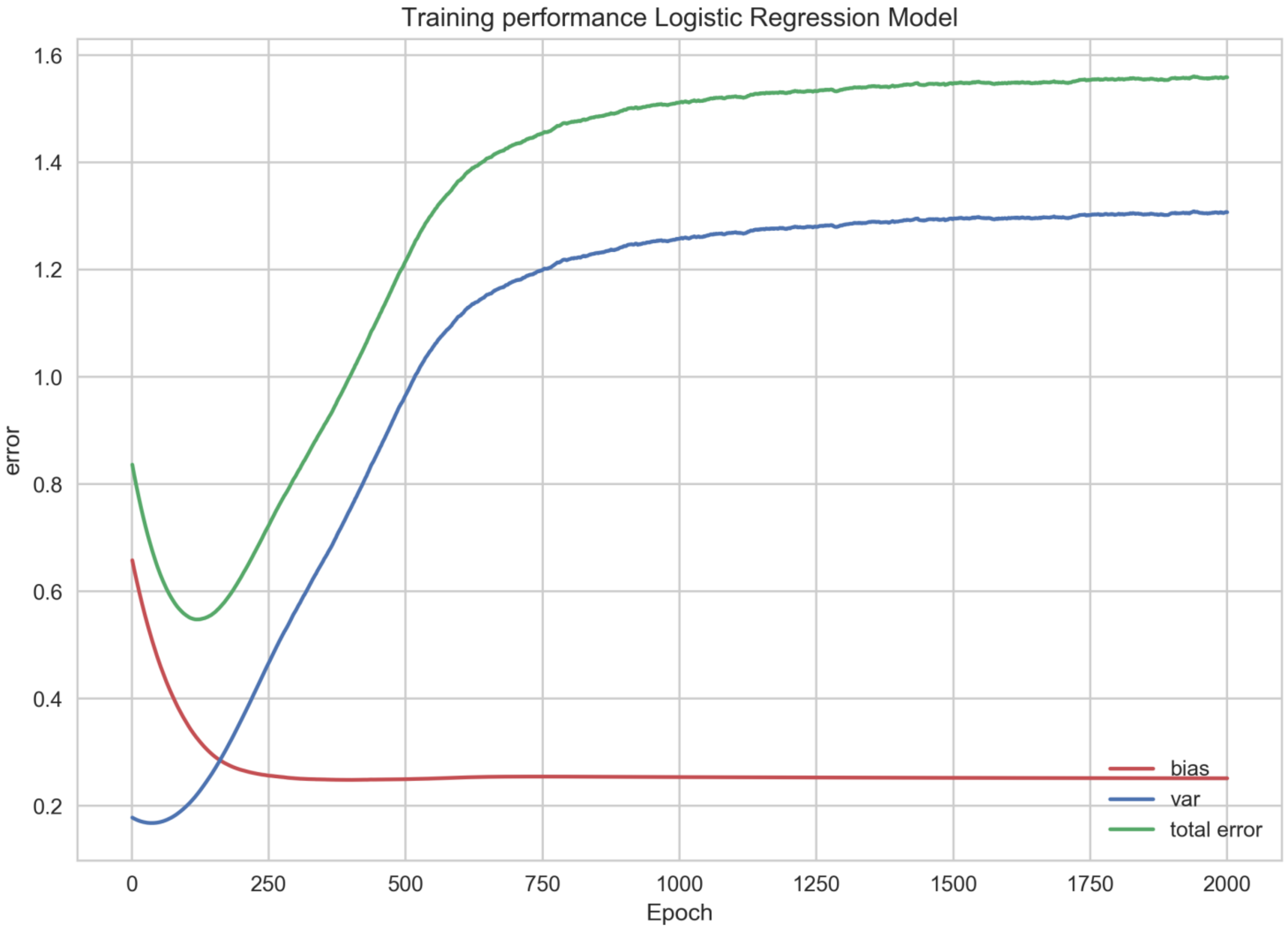
```
with tf.Session() as sess:
    sess.run(init)
    for i in range(iteration):
        minibatch_x,minibatch_y = minibatch(input_data,label_one_hot,batch_size)
        for j in range(len(minibatch_x)):
            model_input_x,model_input_y = minibatch_x[j],minibatch_y[j]
            sess.run(optimizer,feed_dict={X: model_input_x,
                                          Y: model_input_y})
        loss_train_tmp,acc_train_tmp = sess.run([loss,accuracy],feed_dict={X: input_data,
                                                                           Y: label_one_hot})
        bias_train_tmp,var_train_tmp,total_train_tmp = sess.run([bias,var,total],feed_dict={X: input_data,
                                                                                            Y: label_one_hot})
        train_loss_plot.append(loss_train_tmp)
        train_acc_plot.append(acc_train_tmp)
        bias_plot.append(bias_train_tmp)
        var_plot.append(var_train_tmp)
        total_plot.append(total_train_tmp)
        if i % 5 == 0:
            print(i)

    acc_final = sess.run(accuracy,feed_dict = {X:input_data,
                                               Y:label_one_hot})
```

Result in TensorFlow



Result in TensorFlow



THANKS

Group 15