

CS6053_Homework3_Fall23

October 30, 2023

1 Foundations of Data Science Fall 2022 - Homework 3 (30 points)

1.0.1 Student Name: Tekuri Sai Akhil

1.0.2 Student Net Id: st5050

In this assignment we will be looking at data generated by particle physicists to test whether machine learning can help classify whether certain particle decay experiments identify the presence of a Higgs Boson. One does not need to know anything about particle physics to do well here, but if you are curious, full feature and data descriptions can be found here:

- <https://www.kaggle.com/c/higgs-boson/data>
- http://higgsm1.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf

The goal of this assignment is to learn to use cross-validation for model selection as well as bootstrapping for error estimation. We'll also use learning curve analysis to understand how well different algorithms make use of limited data. For more documentation on cross-validation with Python, you can consult the following:

- http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

```
[1]: import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, roc_auc_score
```

1.1 ### Part 1: Data preparation (6 Points)

Create a data preparation and cleaning function that does the following: - Has a single input that is a file name string - Reads data (the data is comma separated, has a row header and the first column EventID is the index) into a pandas **dataframe** - Cleans the data - Convert the feature **Label** to numeric (choose the minority class to be equal to 1) - Create a feature **Y** with numeric label - Drop the feature **Label** - If a feature has missing values (i.e., -999): - Create a dummy variable for the missing value - Call the variable **orig_var_name + _mv** where **orig_var_name** is the name of the actual var with a missing value - Give this new variable a 1 if the original variable is missing - Replace the missing value with the average of the feature (make sure to compute the mean on records where the value isn't missing). You may find pandas' **.replace()** function useful.

- After the above is done, rescales the data so that each feature has zero mean and unit variance (hint: look up sklearn.preprocessing) - Returns the cleaned and rescaled dataset

Hint: as a guide, this function can easily be done in less than 15 lines.

```
[2]: # Place your code here
def pre_process(file_name):
    df = pd.read_csv(file_name)

    # creating feature 'Y' based in 'Label'
    if np.sum(df['Label']=='b') < np.sum(df['Label']=='s'):
        df['Y'] = np.where(df['Label'] == 's', 0, 1)
    else:
        df['Y'] = np.where(df['Label'] == 'b', 0, 1)
    df = df.drop('Label', axis = 1)

    # creating dummy variable for the missing value
    mv_columns = []
    for orig_var_name in df.columns:
        if np.sum(df[orig_var_name]== -999)>0:
            mv_columns.append(orig_var_name+"_mv")
            df[orig_var_name+"_mv"] = np.where(df[orig_var_name] == -999, 1, 0)
            mean = np.mean(np.where(df[orig_var_name] == -999, 0,
↳df[orig_var_name]))
            df[orig_var_name] = np.where(df[orig_var_name] == -999, mean,
↳df[orig_var_name])

    Y = np.array(df['Y'])
    X_mv = np.array(df[mv_columns])
    mv_columns.append('Y')
    X = df.drop(mv_columns, axis =1)
    x_columns = X.columns

    # Preprocessing
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    Y = Y.reshape((Y.shape[0],1))
    X = np.concatenate((X, X_mv),axis = 1)
    X = np.concatenate((X, Y),axis = 1)

    X = pd.DataFrame(X, columns = list(x_columns)+mv_columns)
    return X
```

1.2 ### Part 2: Basic evaluations (6 Points)

In this part you will build an out-of-the box logistic regression (LR) model and support vector machine (SVM). You will then plot ROC for the LR and SVM model.

1. Clean the two data files included in this assignment (`data/boson_training_cut_2000.csv` and `data/boson_testing_cut.csv`) and use them as training and testing data sets.

(1 Point)

```
[3]: # Place your code here

# preprocess train and test
train = pre_process('data/boson_training_cut_2000.csv')
test = pre_process('data/boson_testing_cut.csv')

X_train, Y_train = np.array(train.loc[:,~train.columns.isin(['Y'])]), np.
    ↪array(train.loc[:,train.columns.isin(['Y'])])
X_test, Y_test = np.array(test.loc[:,~test.columns.isin(['Y'])]), np.array(test.
    ↪loc[:,test.columns.isin(['Y'])])
```

2. On the training set, build the following models:

- A logistic regression using sklearn's `linear_model.LogisticRegression()`. For this model, use `C=1e30`.
- An SVM using sklearn's `svm.svc()`. For this model, specify that `kernel="linear"`.

For each model above, plot the ROC curve of both models on the same plot. Make sure to use the test set for computing and plotting. In the legend, also print out the Area Under the ROC (AUC) for reference.

(Hint: to get the prediction thresholds that are necessary for the AUC, use function `predict_proba()` for Logistic Regression and for the classifier you choose if it has it. If you work with SVM, use function `decision_function()`.)

(4 Points)

```
[4]: import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

#Train LogisticRegression
log_reg = LogisticRegression(C=1e30).fit(X_train, Y_train.ravel())
logistic_train_score = log_reg.score(X_train, Y_train)
logistic_test_score = log_reg.score(X_test, Y_test)

#print both train and test scores
print("logistic_train_score, logistic_test_score = "+
    ↪str(logistic_train_score)+", "+str(logistic_test_score))

#Train SVM
svm = SVC(kernel='linear').fit(X_train, Y_train.ravel())
SVM_train_score = svm.score(X_train, Y_train)
SVM_test_score = svm.score(X_test, Y_test)
```

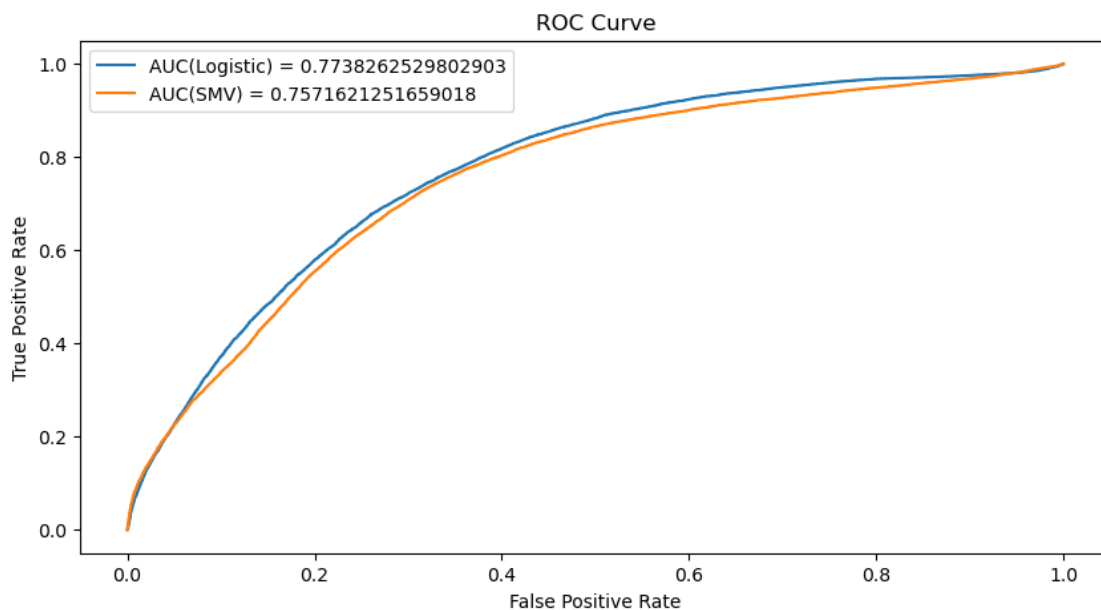
```
#print both train and test scores
print("SVM_train_score, SVM_test_score = "+ str(SVM_train_score)+" ,\n
      ↪"+str(SVM_test_score))
```

```
logistic_train_score, logistic_test_score = 0.7083541770885443, 0.71998
SVM_train_score, SVM_test_score = 0.695847923961981, 0.6976
```

```
[5]: #Now we plot the AUCs
      # Place your code here

      # ROC_AUC score on test data
      log_auc_roc = roc_auc_score(Y_test, log_reg.predict_proba(X_test)[: ,1])
      svm_auc_roc = roc_auc_score(Y_test, svm.decision_function(X_test))

      # Plot the ROC curves
      plt.figure(figsize=(10, 5))
      plt.plot(roc_curve(Y_test, log_reg.predict_proba(X_test)[: ,1])[0],\n
               ↪roc_curve(Y_test, log_reg.predict_proba(X_test)[: ,1])[1],\n
               ↪label='AUC(Logistic) = '+str(log_auc_roc))
      plt.plot(roc_curve(Y_test, svm.decision_function(X_test))[0], roc_curve(Y_test,\n
               ↪svm.decision_function(X_test))[1], label='AUC(SMV) = '+str(svm_auc_roc))
      plt.title('ROC Curve')
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.legend()
      plt.show()
```



3. Which of the two models is generally better at ranking the test set? Are there any classification thresholds where the model identified above as “better” would underperform the other in a classification metric (such as TPR)?

(1 Point)

- 1) Logistic regression model is generally better than SVM model in this case at ranking the test set because of slightly higher value of AUC.
- 2) But there is a range of threshold for TPR values when the SVM model is better than logistic regression model.
- 3) You can see in the curve there is a small range of values of True Positive Rate (TPR) (between 0.0 and 0.2) the logistic regression model underperforms SVM model.

1.2.1 Part 3: Model selection with cross-validation (8 Points)

We think we might be able to improve the performance of the SVM if we perform a grid search on the hyper-parameter C . Because we only have 1000 instances, we will have to use cross-validation to find the optimal C .

1. Write a cross-validation function that does the following: - Takes as inputs a dataset, a label name, # of splits/folds (k), a sequence of values for C (cs) - Performs two loops - Outer Loop: **for each f in $\text{range}(k)$:** - Splits the data into `data_train` & `data_validate` according to cross-validation logic - Inner Loop: **for each c in cs :** - Trains an SVM on training split with $C=c$, `kernel="linear"` - Computes `AUC_c_k` on validation data - Stores `AUC_c_k` in a dictionary of values - Returns a dictionary, where each key-value pair is: `c: [auc-c1, auc-c2, .. auc-ck]` —

```
[6]: #from sklearn.cross_validation import *
import sklearn.model_selection as model_selection

def xValSVM(data, label, k, cs):

    #k-fold indexes
    kf = model_selection.KFold(n_splits=k)
    kfold_indexes = list(kf.split(data))
    AUC = {}
    for c in cs:
        AUC[c] = []

    # perform k-fold cross validation
    for f in range(k):
        X_train, Y_train = np.array(data.loc[kfold_indexes[f][0], ~data.columns.
↪isin([label]))], np.array(data.loc[kfold_indexes[f][0], data.columns.
↪isin([label]))])
        X_validation, Y_validation = np.array(data.
↪loc[kfold_indexes[f][1], ~data.columns.isin([label]))], np.array(data.
↪loc[kfold_indexes[f][1], data.columns.isin([label]))])
        for c in cs:
            svm = SVC(C=c, kernel='linear').fit(X_train, Y_train.ravel())
```

```

        AUC_c_k = roc_auc_score(Y_validation, svm.
↪decision_function(X_validation))
        AUC[c].append(AUC_c_k)

    return AUC

```

2. Using the function written above, do the following: - Generate a sequence of 10 C values in the interval $[10^{-8}, \dots, 10^1]$ (i.e., do all powers of 10 from -8 to 1). 2. Call `aucs = xValSVM(train, 'Y', 10, cs)` 3. For each c in cs , get `mean(AUC)` and `StdErr(AUC)` 4. Compute the value for `max(meanAUC-StdErr(AUC))` across all values of c . 5. Generate a plot with the following: a. $\log_{10}(c)$ on the x-axis b. 1 series with `mean(AUC)` for each c c. 1 series with `mean(AUC)-stderr(AUC)` for each c (use 'k+' as color pattern) d. 1 series with `mean(AUC)+stderr(AUC)` for each c (use 'k-' as color pattern) e. a reference line for `max(AUC-StdErr(AUC))` (use 'r' as color pattern)

```

[7]: # Place your code here
cs = []
i = 10
while i>=1e-8:
    cs.append(i)
    i=i/10

# 10-fold cross validation
aucs = xValSVM(train, 'Y', 10, cs)

mean_AUC = []
std_AUC = []

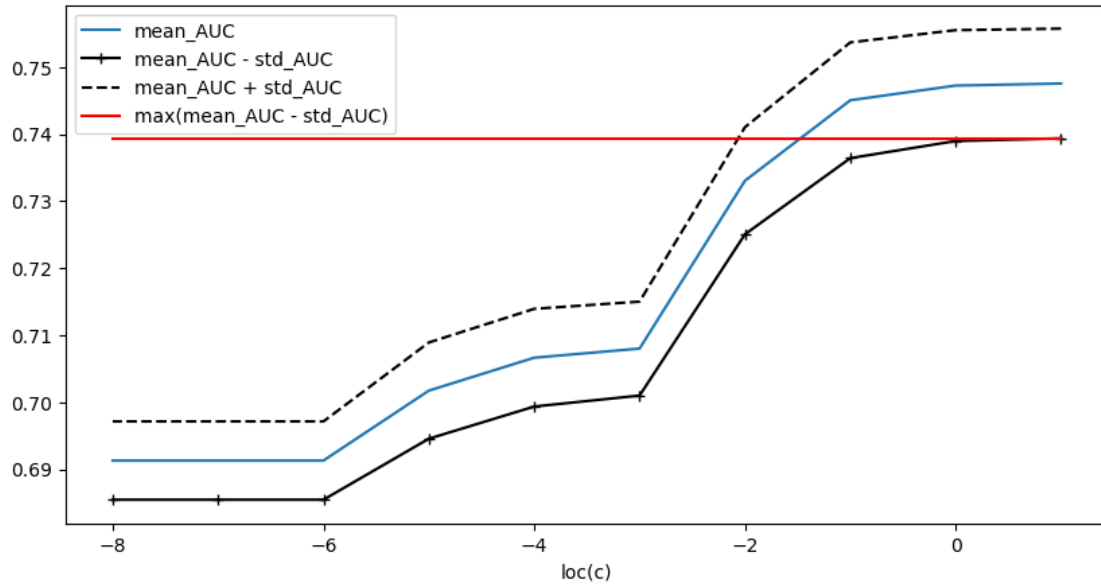
for c in cs:
    mean_AUC.append(np.mean(aucs[c]))
    std_AUC.append(np.std(aucs[c])/np.sqrt(len(aucs[c])))

max_diff = np.max(np.subtract(mean_AUC, std_AUC))

#plot the required metrics
plt.figure(figsize=(10, 5))
plt.plot(np.log10(cs), mean_AUC, label = 'mean_AUC')
plt.plot(np.log10(cs), np.subtract(mean_AUC, std_AUC), 'k+-', label = 'mean_AUC_
↪std_AUC')
plt.plot(np.log10(cs), np.add(mean_AUC, std_AUC), 'k--', label = 'mean_AUC +_
↪std_AUC')
plt.plot(np.log10(cs), [max_diff]*10, 'r', label = 'max(mean_AUC - std_AUC)')

plt.xlabel('log(c)')
plt.legend()
plt.show()

```



Did the model parameters selected beat the out-of-the-box model for SVM? (1 point)

- 1) The AUC value we got from the out-of-the-box model is 0.75
- 2) The mean AUC and $\max(\text{mean_AUC} - \text{std_AUC})$ values we got from doing cross-validation on parameter C are all less than 0.75
- 3) Therefore we can say that the model parameters selected did not beat the out-of-the-box model for SVM

1.3 ###Part 4: Learning Curve with Bootstrapping (10 Points)

In this HW we are trying to find the best linear model to predict if a record represents the Higgs Boson. One of the drivers of the performance of a model is the sample size of the training set. As a data scientist, sometimes you have to decide if you have enough data or if you should invest in more. We can use learning curve analysis to determine if we have reached a performance plateau. This will inform us on whether or not we should invest in more data (in this case it would be by running more experiments).

Given a training set of size N , we test the performance of a model trained on a subsample of size N_i , where $N_i \leq N$. We can plot how performance grows as we move N_i from 0 to N .

Because of the inherent randomness of subsamples of size N_i , we should expect that any single sample of size N_i might not be representative of an algorithm's performance at a given training set size. To quantify this variance and get a better generalization, we will also use bootstrap analysis. In bootstrap analysis, we pull multiple samples of size N_i , build a model, evaluate on a test set, and then take an average and standard error of the results.

1. Create a bootstrap function that can do the following:

```
def modBootstrapper(train, test, nruns, sampsize, lr, c):
```

- Takes as input:

- A master training file (train)
- A master testing file (test)
- Number of bootstrap iterations (nruns)
- Size of a bootstrap sample (sampsiz)
- An indicator variable to specific LR or SVM (lr=1)
- A c option (only applicable to SVM)
- Runs a loop with (nruns) iterations, and within each loop:
 - Sample (sampsiz) instances from train, with replacement
 - Fit either an SVM or LR (depending on options specified). For SVM, use the value of C identified using the 1 standard error method from part 3.
- Computes AUC on test data using predictions from model in above step
- Stores the AUC in a list
- Returns the mean(AUC) and Standard Error(mean(AUC)) across all bootstrap samples

```
[8]: # Place your code here
np.random.seed(42)
def modBootstrapper(train, test, nruns, sampsiz, lr, c):
    '''
    Samples with replacement, runs multiple train/eval attempts
    returns mean and stdev of AUC
    '''
    n = train.shape[0]
    AUC = []
    for i in range(nruns):
        indexs = np.random.choice(n, sampsiz)
        X_train, Y_train = np.array(train.loc[indexs,~train.columns.
        ↪isin(['Y'])]), np.array(train.loc[indexs,train.columns.isin(['Y'])])
        X_test, Y_test = np.array(test.loc[:,~test.columns.isin(['Y'])]), np.
        ↪array(test.loc[:,test.columns.isin(['Y'])])

        #lr=1 for SVM
        if lr==1:
            svm = SVC(C=c, kernel='linear').fit(X_train, Y_train.ravel())
            AUC_c = roc_auc_score(Y_test, svm.decision_function(X_test))
            AUC.append(AUC_c)
        else:
            log_reg = LogisticRegression(C = 1e30, max_iter=1000).fit(X_train,
            ↪Y_train.ravel())
            AUC_c = roc_auc_score(Y_test, log_reg.predict_proba(X_test)[:,:1])
            AUC.append(AUC_c)

    return np.mean(AUC), np.std(AUC)/np.sqrt(len(AUC))
```

2. For both LR and SVM, run 20 bootstrap samples for each samplesiz in the following list: samplesizes = [50, 100, 200, 500, 1000, 1500, 2000]. (Note, this might take 10-15 mins ... feel free to go grab a drink or watch Youtube while this runs).

Generate a plot with the following: - $\log_2(\text{samplesize})$ on the x-axis - 2 sets of results lines, one for LR and one for SVM, the set should include - 1 series with mean(AUC) for each sampsize (use the color options 'g' for svm, 'r' for lr) - 1 series with $\text{mean(AUC)}-\text{stderr(AUC)}$ for each c (use '+' as color pattern, 'g','r' for SVM, LR respectively) - 1 series with $\text{mean(AUC)}+\text{stderr(AUC)}$ for each c (use '-' as color pattern 'g','r' for SVM, LR respectively)

(6 Points. Same as above, this section does not work without above section, so the points are coupled)

```
[9]: # Place your code here
samplesizes = [50, 100, 200, 500, 1000, 1500, 2000]
LR_means = []
LR_stds = []
SVM_means = []
SVM_stds = []

# Perform bootstrapping
for samplesize in samplesizes:
    LR_mean, LR_std = modBootstrapper(train, test, 20, samplesize, 0, 0.01)
    LR_means.append(LR_mean)
    LR_stds.append(LR_std)

    SVM_mean, SVM_std = modBootstrapper(train, test, 20, samplesize, 1, 0.01)
    SVM_means.append(SVM_mean)
    SVM_stds.append(SVM_std)
```

```
[10]: #Plot lr bootstrap
# Place your code here

plt.figure(figsize=(10, 5))

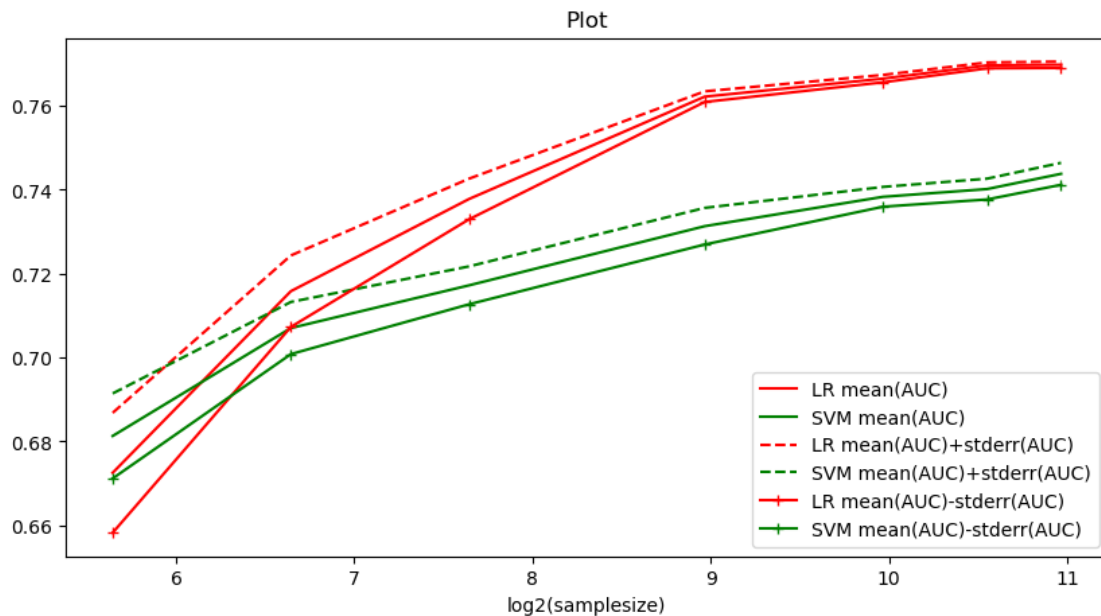
plt.plot(np.log2(samplesizes),LR_means,color = 'r', label = "LR mean(AUC)")
plt.plot(np.log2(samplesizes),SVM_means,color = 'g', label = "SVM mean(AUC)")

plt.plot(np.log2(samplesizes),np.add(LR_means, LR_stds),'r--', label = "LR_
↪mean(AUC)+stderr(AUC)")
plt.plot(np.log2(samplesizes),np.add(SVM_means, SVM_stds),'g--', label = "SVM_
↪mean(AUC)+stderr(AUC)")

plt.plot(np.log2(samplesizes),np.subtract(LR_means, LR_stds),'r+-', label = "LR_
↪mean(AUC)-stderr(AUC)")
plt.plot(np.log2(samplesizes),np.subtract(SVM_means, SVM_stds),'g+-', label =
↪"SVM mean(AUC)-stderr(AUC)")

plt.title('Plot')
plt.xlabel('log2(samplesize)')
```

```
plt.legend()
plt.show()
```



3. Which of the two algorithms are more suitable for smaller sample sizes, given the set of features? If it costs twice the investment to run enough experiments to double the data, do you think it is a worthy investment?

(2 Points)

- 1) SVM is more suitable for smaller sample size.
- 2) If we have small sample size, it is definitely worth the investment to run enough experiments to double the data because as we can see in the plot above the variance of the model decreases as the sample size increases in both Logistic regression and SVM. Even if it takes twice the cost, it's better to have a model with low variance.
- 3) If we already have high enough sample size, then it doesn't make sense to invest in more experiments because the variance of the model will be already small and increasing the sample size might not produce any better model.

4. Is there a reason why cross-validation might be biased? If so, in what direction is it biased? (Hint: refer to the book - The Elements of Statistical Learning figure 7.8)?

(2 Points)

- 1) In part 3 we used 10-fold cross-validation
- 2) Number of samples in train dataset = 2000 samples ($\log_2(2000) = 10.9$)
- 3) For $k=10$ cross-fold validation the model would train on 1800 samples ($\log_2(1800) = 10.8$)
- 4) Therefore train dataset has 2000 samples and 10-fold cross-validation would estimate the performance of our classifier over training set of size 1800 samples.
- 5) From the figure above the performance for sample size 1800 ($\log_2(1800) = 10.8$) is same as

the the performance for sample size 2000($\log_2(2000) = 10.9$). So the 10-fold cross-validation would not suffer from much bias.

But if we increase the k in our k -fold validation the point where the performance for the sample size used in k -fold cross validation ($n - (k/100) * n$) is significantly less than the performance for the sample size in train dataset (n) then there will be bias and the bias would be upward because we will be underestimating the performance (AUC_ROC).