**ISB2020: Genomics for Systems Biology** 4/26/2021

# K-Fold CV vs. Stratification By Cancer Type on RNASeq Data

*Name: Kamaru-Deen Lawal*
*Email: kal246@pitt.edu*

## 1 - Introduction

Our goal is to determine whether or not performing random $k$-fold cross validation or stratification based on cancer type leads to better binary classifier performance when using RNASeq data subsetted from the TCGA dataset as features. Each row in the feature data frame corresponds to a patient sample, while each column represent the individual features. The target, on the other hand, is a vector with the same number of rows as the train dataframe. Each element of the target vector contains a binary label representing whether or not the corresponding sample in the feature data frame is positive or negative for the given mutation. Returning to the initial problem statement, we want to determine whether or not performing $k$-fold cross validation or stratification based on cancer type leads to a better binary classifier. We start with a quick review of $k$-fold cross validation. $k$-fold cross validation is simply the procedure of splitting a dataset into $k$ subsets of roughly equal size, and then using $k-1$ of the subsets to train the model, and the $k$-th subset to validate the model. This procedure is repeated for all $k$ folds, and from there the selected metric of choice is averaged for each of the folds. In the context of our problem we simply let $k = 5$, and follow the aforementioned procedure. Next we do a quick review of stratified cross validation. Stratified cross-validation is the same as $k$-fold cross validation, but instead of randomly breaking up the data into $k$ sub-groups we break the data up according to some external criterion. Assuming we have $k$ criterions we use $k-1$ of the criterions to train a model, and the $k$-th criterion to validate the model. Similarly to before we average over each resultant evaluation metric for the folds. One interesting distinction worth noting is that while each of the folds in $k$-fold cross validation is roughly the same size, the size of the folds in stratified cross validation vary depending on the external criterion being used to separate the data (see Figure 5 for the cancer subtype distribution in the supplement). In the context of our problem we are stratifying based on cancer subtype, which we have stored in an external CSV file for each sample of the TCGA dataframe. Before getting into our methods and results we will start with a basic literature overview.

## 2 - Literature Overview

A basic preliminary literature overview showed that ElasticNet performs exceptionally well on datasets that are similar to ours. Specifically datasets that have severe class imbalance in the target with a large number of features. We recall the cost function for elastic net regularization below.

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda\sum_{i=0}^{M}|W_j| + \lambda\sum_{i=0}^{M}W_j^2$$

In addition there was a very useful paper that explained how precision oncology attempts to use genomic evidence to match patients with treatments but can at times fail to identify all patients who may respond. The paper describes an algorithm that used RNASeq data and mutations from 33 different cancer types across the TCGA project to predict abnormal molecular states in tumors. The method used in the paper has variable performance across cancer types but overall is sensitive and specific over generalizable cell-line data. The approach can also be applied generally to other genes and pathways.

## 3 - Methods

As stated in the introductory section we have three main dataframes that we need to complete our analysis. The RNASeq data subsetted from the TCGA dataframe, the target dataframe, and the metadata dataframe. Moving forward we will refer to the process of performing $k$-fold cross validation as unstratified analysis, and the process of performing stratified based cross validation as stratified analysis. As previously stated, each row of the target dataframe contains a 9072 length vector with the presence/absence of mutations for various feature mutations. We are specifically interested in assessing model performance for the following proteins in the target dataframe: TP53, PIK3CA , KRAS, PTEN, ARID1A, RB1, FBXW7, NRAS, CDKN2A, CTNNB1, HRAS, NF1,

IDH1, KMT2D, and NFE2L2. These proteins were selected because they have the highest number of positive cases in the entire dataset. Its also worth noting that like most binarized datasets in healthcare, there is severe class imbalance. In our unstratified analysis the metadata dataframe is irrelevant so the procedure for setting up the data required for training and testing is as simple as merging the RNASeq dataset with the target protein of interest (see Code Block in Supplement). In our stratified analysis the metadata dataframe is needed in order for us to split the data, so we start by taking the set of diseases from the metadata dataframe and mapping them to $\mathbb{N}_0$. From there we merge the RNASeq data to the metadata, and then merge that resultant dataframe to the target of interest during the stratified cross-validation stage (see Code Block in supplement). As stated in the From here we trained ElasticNet and RandomForest binary classifiers for both the unstratified and stratified analyses and assessed performance (see Code Block 3 in supplement). Its worth noting that there are a multitude of ways that parameters can be selected for classifiers. I ended up merging the RNASeq data with target BAP1, and performing GridSearch for the $criterion$ and $max features$ in the Random Forest classifier, and the $C$ value in Logistic Regression classifier. It is also worth noting that I selected the BAP1 protein because it was one of the proteins that we are not assessing classifier performance for.

# 4 - Results

After training each the the ENet and RF classifiers for the stratified and unstratified cases, we found the mean AUPR for each of the proteins we sought to assess performance for. The mean AUPRs for the individual proteins are not useful for each of the individual proteins, but they are useful when compared against one another. We assessed (1) stratified vs. unstratified elasticnet performance for the 5-fold cross validation, (2) stratified vs. unstratified randomforest performance for the 5-fold cross validation, (3) elasticnet vs. randomforests for the unstratified analysis, and (4) elasticnet vs. randomforests for the stratified analysis. Each of the resultant plots can be seen under figures in the supplement section. The plots show the mean AUPR for the protein under the stratified analysis on the $x$-axis, and the mean AUPR for the unstratified analysis on the $y$-axis. Any point to the left of the line $y = x$ indicates that the unstratified analysis performed better, while any point to the right of the line indicates that the stratified analysis performed better. Both (1) and (2) show us that the unstratified analysis very clearly outperformed the stratified analysis. The plot corresponding to (3) shows that elasticnet performed better for most of the proteins under question, while the plot corresponding to (4) shows slightly more balance between the elasticnet and randomforest classifiers.

# 5 - Discussion

This problem is very interesting, and I had a great time working on it. There are some fairly obvious directions to build on this work that I have started under Dr. Benos, Dr. Chikina, and Dr. Kostka. Firstly, I think further exploration can be done on how to select the hyper-parameters for the models that we settled on. I chose BAP1 as the protein to find the hyper-parameters because it was mutation with the highest number of positive cases after the fifteen proteins that I used for training. This seems like a fairly naive method, and there must be more interesting methods that can be used. In addition it would be interesting to see how support vector machines perform on this task. I briefly began experimenting with some models to see how they would perform, but they ended up taking too much time. The next most obvious step would be to see how to more efficiently parallelize model training. The model training took about 15 days on AWS c6g.2xlarge instances. I used a fairly naive algorithm to train and validate the models that simply performed the training in an iterative fashion. While this gets the job done, each of the proteins is independent of the other proteins when performing the stratified analysis so some type of algorithm can be devised to get the job done faster.

# 6 - Acknowledgements

I would like to thank my father, Kamaru Lawal, for inspiring me to constantly push my limits in the realm of academia.

# References

Way GP, Sanchez-Vega F, La K, Armenia J, Chatila WK, Luna A, Sander C, Cherniack AD, Mina M, Ciriello G, Schultz N; Cancer Genome Atlas Research Network, Sanchez Y, Greene CS. Machine Learning Detects Pan-cancer Ras Pathway Activation in The Cancer Genome Atlas. Cell Rep. 2018 Apr 3;23(1):172-180.e3. doi: 10.1016/j.celrep.2018.03.046. PMID: 29617658; PMCID: PMC5918694.

# Supplement

## Code Block

```
%%capture
import os, pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import defaultdict
from sklearn import metrics
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_roc_curve, accuracy_score, auc, average_precision_score,
    confusion_matrix, roc_curve, precision_recall_curve, plot_precision_recall_curve,
    precision_recall_fscore_support
from sklearn.linear_model import LogisticRegression

# read in data
data = pd.read_csv("../data_remote/DataMatched.txt", delimiter="\t")
data.rename(columns={"Unnamed: 0":"features"}, inplace=True)
data.drop(data.tail(1).index,inplace=True)

# read in target info
target = pd.read_csv("../data_remote/TargetMutations.txt", delimiter="\t")
target.rename(columns={"Unnamed: 0":"features"}, inplace=True)

# read in info
info = pd.read_csv("../data/infoMatched.txt", delimiter="\t")
info.drop(columns=["Unnamed: 0"], axis=1, inplace=True)


# store/load in x data
datapath = "../data/x.npy"
if not os.path.exists(datapath):
    # store data as numpy array
    x = data.values
    # remove first column (feature names)
    x = x[:,1:]
    # transpose features for classifier
    x = x.T
    np.save(datapath, x)
else:
    x = np.load(datapath, allow_pickle=True)
# store target as numpy array

targetpath = "../data/tp53.npy"
if not os.path.exists(targetpath):
    y = target.loc[target["features"] == "TP53"].values
    y = np.asarray(y.tolist()[0][1:])
    np.save(targetpath, y)
else:
    y = np.load(targetpath, allow_pickle=True)

# setup data frame for merge
data.set_index(keys="features", inplace=True)
trans_data = data.T
trans_data.reset_index(inplace=True)
trans_data.rename(columns={"index":"SAMPLE_BARCODE"}, inplace=True)
trans_data.set_index("SAMPLE_BARCODE", inplace=True)

# setup info frame for merge
info.set_index("SAMPLE_BARCODE", inplace=True)
```

```
info.drop(columns=["PATIENT_BARCODE", "SUBTYPE"], axis=1, inplace=True)

# change DISEASE to DISEASE_ID

## create disease dictionary
diseases = info.DISEASE.unique()
disease_dict = dict(zip(diseases, range(len(diseases))))
invdisease_dict = {v: k for k, v in disease_dict.items()} # needed for visualization

## DISEASE -> DISEASE_ID
info["DISEASE_ID"] = info["DISEASE"].map(disease_dict)
info.drop(columns=["DISEASE"], axis=1, inplace=True)
print("Disease Dictionary")
print(disease_dict)
print()
print("Inverse Disease Dictionary")
print(invdisease_dict)

# output feature/target shape
print("features shape:", x.shape)
print("target:", y.shape)
## save additional proteins to be used as Y targets
more_proteins="TP53,PIK3CA,KRAS,PTEN,ARID1A,RB1,FBXW7,NRAS,CDKN2A,CTNNB1,HRAS,NF1,IDH1,KMT2D,NFE2L2"
more_proteins=more_proteins.split(",")
print(more_proteins)

# store mean_auc + mean_aupr for each Y value in more_proteins
y_mean_aucs = defaultdict(float)
y_mean_auprs = defaultdict(float)

for prot in more_proteins:
    # store target for training
    y = target.loc[target["features"] == prot].values
    y = np.asarray(y.tolist()[0][1:])
    # y = y[:99]
    frame["target"] = y
    y_prot_name = prot

    random_state=777
    # define elasticnet classifier
    classifier = LogisticRegression(penalty="elasticnet", random_state=random_state, solver="saga",
        l1_ratio=.5, n_jobs=-1)

    # store parameters for ROC + AUPR Visualization/Analysis

    ## AUPR Variables
    y_real = []
    y_proba = []

    # train/predict
    fig, ax = plt.subplots()
    for disease_index in list(disease_dict.values()):
        # split data based on disease class
        train = frame.loc[frame["DISEASE_ID"] != disease_index]
        test = frame.loc[frame["DISEASE_ID"] == disease_index]

        # drop disese id (frame -> X|y)
        train.drop(columns=["DISEASE_ID"], axis=1)
        test.drop(columns=["DISEASE_ID"], axis=1)

        # save y_train, y_test
        y_train, y_test = train["target"], test["target"]
```

```
    # drop target (X|y -> X)
    X_train, X_test = train.drop(columns=["target"], axis=1), test.drop(columns=["target"], axis=1)

    # fit model + save probabilities for aupr
    probas_ = classifier.fit(X_train, y_train).predict_proba(X_test)

    # save predictions for future analysis
    preds = classifier.fit(X_train, y_train).predict(X_test)
    predpath = "../data/" + y_prot_name + "_" + invdisease_dict[disease_index] + "_STRATPREDS.npy"
    np.save(predpath, preds)

    viz = plot_precision_recall_curve(classifier, X_test, y_test,
                      name=y_prot_name + ' PR Curve fold
                          {}'.format(invdisease_dict[disease_index]),
                      alpha=0.3, lw=1, ax=ax)
    y_real.append(y_test)
    y_proba.append(probas_[:, 1])

# visualize
y_real = np.concatenate(y_real)
y_proba = np.concatenate(y_proba)
precision, recall, _ = precision_recall_curve(y_real, y_proba)

# update + save y_mean_auprs for plotting against unstratified y_mean_auprs
y_mean_auprs[y_prot_name] = average_precision_score(y_real, y_proba)
print(y_mean_auprs)
y_mean_auprs_path = "../data/strat_mean_auprs.pkl"
pickle.dump(y_mean_auprs, open(y_mean_auprs_path, "wb" ))

plt.plot(recall, precision, color='b',
        label=r'Precision-Recall (AUC = %0.2f)' % (average_precision_score(y_real, y_proba)),
        lw=2, alpha=.8)
ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
      title="Precision-Recall curve")
ax.set_title(y_prot_name + "Stratified PR Curve")
ax.legend(loc="lower right")
plt.figure(figsize=(11,11))
plt.show()
```

## Figures

Figure 1: Elastic Net Plot

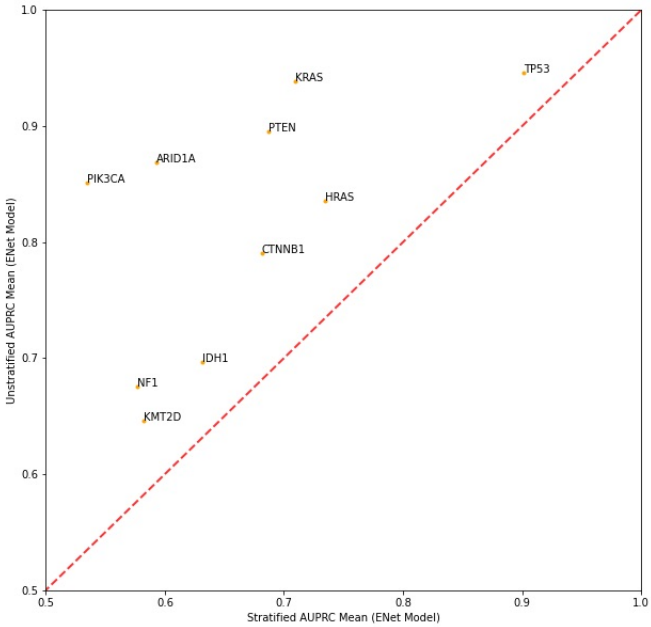


Figure 2: Random Forests Plot

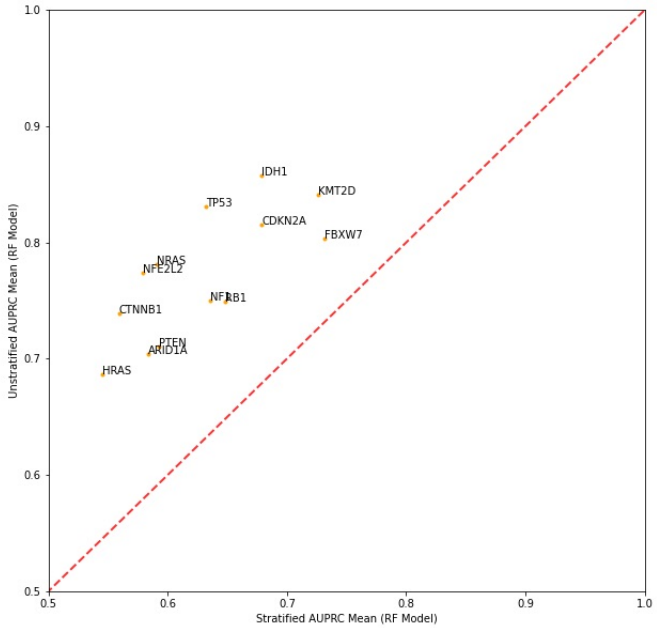

Figure 3: Elasticnet vs. Randomforests Unstratified

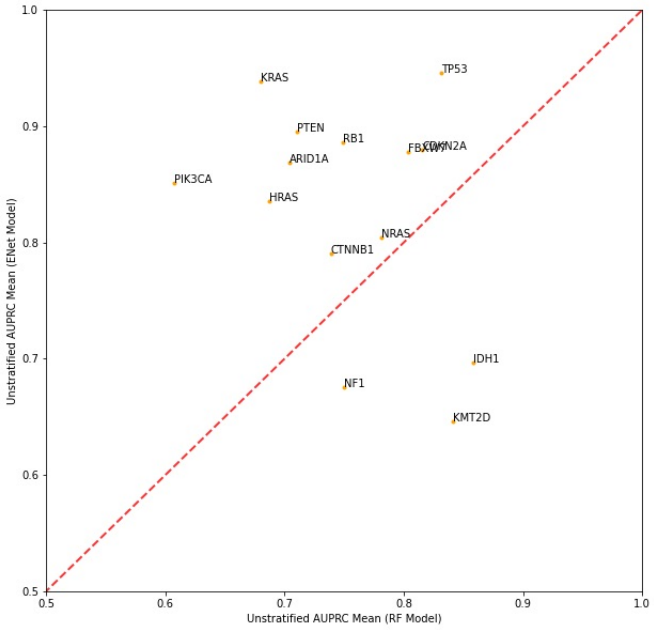


Figure 4: Elasticnet vs. Randomforests Stratified

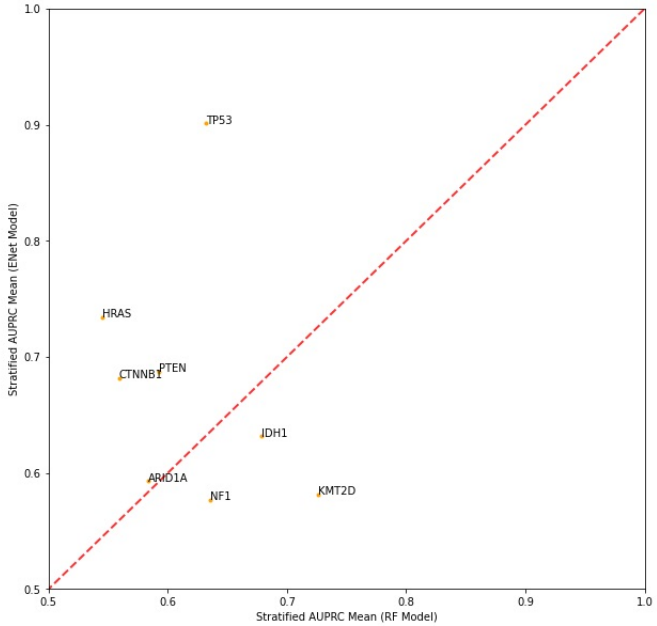

Figure 5: Distribution of Cancer Substypes