#Logistic regression analysis:
#We used logistic regression as a linear classification tool to relate experimentally determined phenotypes to the Rosetta computed energy landscape. During this type of analysis, hyperplanes in feature space are defined and used to separate observations belonging to different classes. All data preparation and logistic regression analyses were performed using R version 3.6.1. Stratified sampling and logistic regression using the linear discriminant analysis method (see commands below) were conducted with the dplyr and carat packages.
#Data stratification:
#Stratified sampling reduces the likelihood of randomly sampling the same region of the energy landscape when selecting data to form calibration sets; sampling across the range of ????G values increases the probability of sampling members of each phenotype, which is necessary for determining the energy partitions. The comprehensive Rosetta computed ????GTotal vs ????GInterfacial energy landscape of all single (285) and double or paired mutants (37,905) was generated and divided into 4 evenly populated strata (I, II, III, and IV). To generate the strata, the data were first partitioned using the median ????GTotal, and then these subgroups were further partitioned by their median ????GInterfacial. The strata boundaries were drawn from this ~38,000-member set because these would be the only data available at this stage if we were applying SPORT to a new split protein system. The 193 experimentally characterized TEVp variants were mapped to these strata for the retrospective stratified bootstrap sampling analysis.
#Commands:
```
library(caret)
library(dplyr)
#LDA of our actual calibration set
cal <- read.csv("actual_calibration_set.csv", header=T)
new <- read.csv("stratified_after_calibration_set.csv", header=T)
train <- cal
test <- new
train_ddg <- cbind(train$ddG_total, train$ddG_interfacial)
train_pheno <- as.factor(as.character(train$phenotype))
test_ddg <- cbind(test$ddG_total, test$ddG_interfacial)
test_pheno <- factor(test$phenotype, levels=levels(train_pheno))
lda_model <- train(x=train_ddg, y=train_pheno, method="lda",
metric="Accuracy")
lda_pred <- predict(lda_model, test_ddg)confusionMatrix(lda_pred,
as.factor(test_pheno))

#Retrospective LDA using stratified bootstrap sampling of characterized
variants
data <- read.csv("stratified_all_tested_data.csv", header=T)
for(size in 1:7) #since there are 4 strata, we are sampling 4, 8, ., 28
variants
{
  for(samp_set in 1:10)
  {   set.seed(size*samp_set+100)
    strat_samp <- as.data.frame(data %>% group_by(strata) %>%
sample_n(size) %>% ungroup)
    train <- data[strat_samp[,1],]
    test <- data[-strat_samp[,1],]
    train_ddg <- cbind(train$ddG_total, train$ddG_interfacial)
```

```
    train_pheno <- as.factor(as.character(train$phenotype))
    test_ddg <- cbind(test$ddG_total, test$ddG_interfacial)
    test_pheno <- factor(test$phenotype,
levels=levels(train_pheno))lda_model <- train(x=train_ddg, y=train_pheno,
method="lda", metric="Accuracy")
    lda_pred <- predict(lda_model, test_ddg)
    cm <- confusionMatrix(lda_pred, as.factor(test_pheno))
    temp_byclass <- as.table(cm$byClass)
    temp_overall <- as.table(cm$overall)
    if(samp_set==1)
    {
      byclass <- temp_byclass
      overall <- temp_overall
    } else
    {
      byclass <- rbind(byclass, temp_byclass)
      overall <- rbind(overall, temp_overall)
    }
  }
  notInd <- subset(byclass, row.names(byclass)=="Class: not inducible")
  Ind <- subset(byclass, row.names(byclass)=="Class: inducible")
  Dead <- subset(byclass, row.names(byclass)=="Class: dead")
  all_overall <- overall[1:2,]
  row.names(all_overall) <- c('mean','sd')
  all_notInd <- byclass[1:2,]
  row.names(all_notInd) <- c('mean','sd')
  all_Ind <- all_notInd
  all_Dead <- all_Ind
  for(param in 1:length(overall[1,]))
  {
    all_overall[1,param] <- mean(overall[1:length(overall[,1]),param],
na.rm = TRUE)
    all_overall[2,param] <- sd(overall[1:length(overall[,1]),param], na.rm
= TRUE)
  }
  for(param in 1:length(byclass[1,]))
  {
    all_notInd[1,param] <- mean(notInd[1:length(notInd[,1]),param], na.rm =
TRUE)
    all_notInd[2,param] <- sd(notInd[1:length(notInd[,1]),param], na.rm =
TRUE)
    all_Ind[1,param] <- mean(Ind[1:length(Ind[,1]),param], na.rm = TRUE)
    all_Ind[2,param] <- sd(Ind[1:length(Ind[,1]),param], na.rm = TRUE)
    all_Dead[1,param] <- mean(Dead[1:length(Dead[,1]),param], na.rm = TRUE)
    all_Dead[2,param] <- sd(Dead[1:length(Dead[,1]),param], na.rm = TRUE)
  }
  if(size==1)
  {
    mean_overall <- all_overall[1,]
    sd_overall <- all_overall[2,]
    mean_notInd <- all_notInd[1,]
    sd_notInd <- all_notInd[2,]
    mean_Ind <- all_Ind[1,]
    sd_Ind <- all_Ind[2,]
    mean_Dead <- all_Dead[1,]
    sd_Dead <- all_Dead[2,]
```

```
  } else
  {
    mean_overall <- rbind(mean_overall,all_overall[1,])
    sd_overall <- rbind(sd_overall,all_overall[2,])
    mean_notInd <- rbind(mean_notInd,all_notInd[1,])
    sd_notInd <- rbind(sd_notInd,all_notInd[2,])
    mean_Ind <- rbind(mean_Ind,all_Ind[1,])
    sd_Ind <- rbind(sd_Ind,all_Ind[2,])
    mean_Dead <- rbind(mean_Dead,all_Dead[1,])
    sd_Dead <- rbind(sd_Dead,all_Dead[2,])
  }
}
```