**Predicting the Functional Impact of KCNQ1 Variants with Artificial Neural Networks**

Saksham Phul1,3, Georg Kuenze1,3,6, Carlos G. Vanoye4, Charles R. Sanders1,2, Alfred L. George, Jr.4, Jens Meiler1,3,4,6,\*

1Center for Structural Biology, Vanderbilt University, Nashville, TN 37240, USA

2Department of Biochemistry, Vanderbilt University, Nashville, TN 37232, USA

3Department of Chemistry, Vanderbilt University, Nashville, TN 37235, USA

4Department of Pharmacology, Northwestern University Feinberg School of Medicine, Chicago, IL 60611, USA

5Department of Pharmacology, Vanderbilt University, Nashville, TN 37240, USA

6Institute for Drug Discovery, Leipzig University, Leipzig, SAC 04103, Germany

\*Correspondence to: Jens Meiler, PhD, [jens@meilerlab.org](mailto:jens@meilerlab.org)

**Protocol Capture**

Table of content

1) Features generation…………………………………………………………………………….3

2) Model Training…………………………………………………………………….…………..4

3) Performance analysis……………………………………………………………….………….7

4) Input sensitivity analysis…………………………………………………….…………………9

1. *Features generation*

All the models for this work were developed using Bio Chemical library (BCL). Before an ANN model is trained in BCL, we need to prepare a bin file that consists of descriptors and result labels. The following command line below generates a bin file that BCL can read for the purpose of training.

bcl-apps-static.exe GenerateDataset \

-source "ProteinMutationsDirectory(./, key file=./keys.txt, aa class=AAComplete, mutation extension=.data\_file.csv, suffix=.pdb, add self mutation fraction=0.0)" \

-output <output\_bin\_file\_for\_model.bin> \

-feature\_labels <feature\_file.obj> \

-result\_labels <resultslabel\_file.obj> \

-id\_labels 'Combine (MutationId)'

The input required for the above command line are:-

1. keys.txt file that contains name of the protein structure file
2. a protein structure in pdb format named as mentioned in keys.txt
3. file named as .uniref50.0.00001.ascii4 and .nr.ascii4 contains position specific scoring matrix from two different databases. These files are read through features\_file.obj file
4. a data file that contains that variant IDs with functional data.

The command line computes biophysical features from the protein structure and evolutionary data from NCBI non-redundant database and uniref50 database. The *feature\_file.obj* is the file that instructs the BCL program which features shall be computed. The *result\_file.obj* file instructs the BCL program which labels shall be extracted from data csv file and passed to the model to train against these labels.

Position specific scoring matrix can be generated using psiblast.exe with following command line

psiblast -db <Name of the database> -query <protein sequence in fasta format> -num\_iterations <No. of iterations> -inclusion\_ethresh <hit threshold> -evalue <hit threshold> -num\_alignments 0 -out\_pssm <Output file name> -comp\_based\_stats 0 -use\_sw\_tback >> output.log 2>&1

1. *Model Training*

Once we have generated a bin file, an ANN model can be trained using the following BCL command :-

bcl-apps-static.exe model: Train \

"NeuralNetwork (

balance=True, \

balance target ratio=1, \

balance max repeats=100, \

transfer function = Rectifier (0.05), \

weight update = Simple(alpha=<momentum>, eta=<learning rate>), \

input dropout type=Zero, \

objective function =Accuracy(cutoff=0.5), \

input noise=0.0, \

iteration weight update=Attenuate(0.0,0.0,0.0,0.001), \

shuffle=True, \

steps per update=1, \

dropout(<input\_layer\_dropout>, <first hidden layer dropout>, <second hidden layer dropout>), \

hidden architecture (<No.\_of\_ neurons\_in\_first\_layer, No.\_of\_ neurons\_in\_second\_layer),

rescale output dynamic range=True,rmsd report frequency=1, scaling=AveStd)"

-max\_minutes <number minutes to train>

-max\_iterations <No. of iterations>

-opencl Disable

--result\_averaging\_window 0

-final\_objective\_function <objective function for training>

-feature\_labels <features\_file.obj>

-result\_labels <results\_file.obj>

-id\_labels '<id\_labels>'

-training "<training\_data>"

-monitoring "<validation\_data>"

-independent "<prediction\_data>"

-print\_independent\_predictions “<storage predicted result to>”

-storage\_model "File (directory=<storage model address>, prefix=model, key=<Model ID>)"

This command line trains a single model with fixed training, validation, and independent dataset. Here, architecture of ANN can be specified using hidden architecture tag. weight update tag sets learning rate and momentum for the network. Using -training, -monitoring, and -independent tags, training data, validation data, and independent dataset respectively can be passed to model to validate model performance. For this work, we adopted a 25-fold cross validation technique to remove any biasness and have better generalizability while making the predictions. A slurm script *(script\_chunk.s*h) assisted in validation by dividing the dataset into 25 subsets. This script requires 25 threads that run the calculation on 25 different independent subsets simultaneously. Each thread runs over 25 validation datasets to predict a single independent dataset. The BCL command line below divides the dataset into 25 subsets wherein one subset was used as monitoring, one subset for independent testing, and rest for training.

Subset(filename=<output\_bin\_file\_for\_model.bin>, number chunks=<Number of subsets>, chunks=<Specify the subsets to use>)

We can alter chunks in above BCL command to specify subsets to use for -training , -monitoring , and -independent flags in training BCL command line. The for loop and if/else conditionals were used in bash *Script\_chunk.sh* to vary the number of subsets for monitoring and validation.

---------------------------------------------------------Script\_chunk.sh------------------------------------------------------------

#!/bin/bash

#SBATCH --nodes=1

#SBATCH --ntasks=1

#SBATCH --mem-per-cpu=2G

#SBATCH --time=0:50:00

#SBATCH --array=0-24

#SBATCH --job-name="Q1PredVarBio"

no\_monitoring\_set=24

divide=25

# first hidden layer size

netsize="32"

# second hidden layer size

netsize2="12"

# input layer dropout

visdrop="05"

# first hidden layer dropout

hdrop="33"

hdrop2="33"

iteration=1200

learning=0.002

momem=0.5

input\_file=a1q1.mutations.combined155123.csv

features=features\_combined.obj

results=resultslqts\_combined.obj

MODEL\_ID=${SLURM\_ARRAY\_TASK\_ID}

#this loop runs over the monitoring sets and model ID runs over the independent sets

for a in $(seq 0 $no\_monitoring\_set);

do

if [ $a -ne $MODEL\_ID ];

then

MONIND="Subset (filename= <output\_bin\_file\_for\_model>, number chunks=$divide, chunks=[$a])"

TRAIN="Subset(filename=full.whet.multimer\_combined.bin,number chunks=$divide,chunks=[0]+[1]+[2]+[3]+[4]+[5]+[6]+[7]+[8]+[9]+[10]+[11]+[12]+[13]+[14]+[15]+[16]+[17]+[18]+[19]+[20]+[21]+[22]+[23]+[24]-[$a]-[$MODEL\_ID])"

else #boundary cases

if [ ${a} == 0 ];

then

MONIND="Subset (filename= <output\_bin\_file\_for\_model>, number chunks=$divide, chunks=[1])"

TRAIN="Subset(filename=full.whet.multimer\_combined.bin,number chunks=$divide,chunks=[0]+[1]+[2]+[3]+[4]+[5]+[6]+[7]+[8]+[9]+[10]+[11]+[12]+[13]+[14]+[15]+[16]+[17]+[18]+[19]+[20]+[21]+[22]+[23]+[24]-[1]-[$MODEL\_ID])"

else

b=$(($a-1))

MONIND="Subset (filename=full.whet.multimer\_combined.bin,number chunks=$divide, chunks=[$b])"

TRAIN="Subset(filename=full.whet.multimer\_combined.bin,number chunks=$divide,chunks=[0]+[1]+[2]+[3]+[4]+[5]+[6]+[7]+[8]+[9]+[10]+[11]+[12]+[13]+[14]+[15]+[16]+[17]+[18]+[19]+[20]+[21]+[22]+[23]+[24]-[$b]-[$MODEL\_ID])"

fi

fi

IND="Subset (filename=full.whet.multimer\_combined.bin, number chunks=$divide, chunks=[$MODEL\_ID])"

IDN=$1

rm -f models/$IDN/model`printf %06d $MODEL\_ID`.\*

#BCL training command line

bcl-apps-static.exe model:Train "NeuralNetwork(balance=True, balance target ratio=1,balance max repeats=100,transfer function = Rectifier(0.05),weight update = Simple(alpha=$momem, eta=$learning),input dropout type=Zero, objective function =Accuracy(cutoff=0.5),input noise=0.0,iteration weight update=Attenuate(0.0,0.0,0.0,0.001),shuffle=True, steps per update=1,dropout(0.$visdrop,0.$hdrop,0.$hdrop2),hidden architecture($netsize,$netsize2),rescale output dynamic range=True, rmsd report frequency=1,scaling=AveStd)" -max\_minutes 300 -max\_iterations $iteration -opencl Disable --result\_averaging\_window 0 -final\_objective\_function 'Accuracy(cutoff=0.5)' -feature\_labels $features -result\_labels $results -id\_labels 'Combine(MutationId)' -training "$TRAIN" -monitoring "$MONIND" -independent "$IND" -print\_independent\_predictions ./results/$IDN/independent${a}\_${MODEL\_ID}\_monitoring${MODEL\_ID}\_number${a}.gz -storage\_model "File(directory=models/$IDN/${a}/,prefix=model, key=$MODEL\_ID)" > ./log\_files/$IDN/independent${MODEL\_ID}\_monitoring${MODEL\_ID}\_number${a}.txt

cp results/$IDN/independent${a}\_\* results/$IDN/$a

done

exit 0

To better organize predictions by the model, we use run.sh script that creates separate folders – one for predictions obtained with every validation set. We can use run.sh directly to run this model. This script prepares the <output\_bin\_file\_for\_model> file and then, submits the script\_chunks.sh to slurm scheduler. It also cleans up the folder with temporary files before the start of the *script\_chunk.sh*. Command line to run the model conveniently is :

./run.sh <Name\_of\_model>

1. *Performance analysis*

Once the predictions are made by the ANN model, we can change into result folder and compute model performance. In the result folder, there are 25 subfolders – one for every of the 25 different validation sets. In each subfolder, 25 prediction files exist that were created with different prediction datasets using same validation set. Here, we can concatenate the compressed output files into a single txt file with the command :-

zcat independent\*.gz > concat.txt

This will output *concat.txt* file. First column of the concat.txt file has the variant id. Second column to fifth column have the prediction values for peak current (IKs), voltage of half-maximal activation (V1/2), activation time τact and deactivation time τdeact respectively. Sixth column to the last column have the functional labels (0 for normal and 1 for dysfunctional) used for training in the same order as predicted values. BCL recognizes this pattern and can directly evaluate the performance using the command line: -

bcl-apps-static.exe model:ComputeStatistics \

-input concat.txt \

-obj\_function 'ContingencyMatrixMeasure (measure=MCC, parity=1, cutoff=0.5, adjustable cutoff=0)' 'ContingencyMatrixMeasure(measure=MCC, parity=1,cutoff=0.5,adjustable cutoff=1)' \

-filename\_obj\_function <final\_result.txt>

This will generate *final\_result.txt* that contained the average MCC for four prediction labels. This command will also output log messages to the screen that contain individual MCC and ROC values for the four prediction labels.

*Concat.txt* contains predictions for silent and missense mutations. Missense mutations can be separated from silent mutations using a python script *delete\_silent.py*. The results reported in the paper were obtained considering only missense mutations.

To run this script

python delete\_silent.py

---------------------------------delete\_silent.py-----------------------------------------------

def reading\_file(name):

a=[]

with open(name,"r") as f:

content=f.read().splitlines()

for i in range(0,len(content),1): # separates the missense mutations from silent mutations

if(content[i][0]!=content[i][4]):

a.append(content[i])

return a

def writing\_file(filename,data):

with open(filename,"w") as f:

for p in range(0,len(data),1):

k=data[p].split(",")

d1=[k[0],k[1],k[2],k[3],k[4],k[5],k[6],k[7],k[8]]

d1=(str(d1).replace("[",""))

d1=(str(d1).replace("]",""))

d1=(str(d1).replace("'",""))

f.write(str(d1))

f.write("\n")

f.close()

a1=reading\_file("concat.txt")

writing\_file("deleted\_silent.txt",a1)

----------------------------------------------END---------------------------------------------------------------

Input to this python script is *concat.txt* that will output *deleted\_silent.txt* file. We evaluate the performance of our model on only missense mutations using following command line:-

bcl-apps-static.exe model:ComputeStatistics \

-input deleted\_silent.txt \

-obj\_function 'ContingencyMatrixMeasure (measure=MCC, parity=1, cutoff=0.5, adjustable cutoff=0)' 'ContingencyMatrixMeasure(measure=MCC, parity=1,cutoff=0.5,adjustable cutoff=1)' \

-filename\_obj\_function <final\_result.txt>

1. *Input sensitivity analysis*

The consistency method can be implemented using command line :-

bcl-apps-static.exe descriptor:ScoreDataset

-source 'Subset(filename= <output\_bin\_file\_for\_model.bin>)'

-score "InputSensitivityNeuralNetwork(storage=File(directory=<address to model directory>, prefix=model),weights=ScoreDerivativeEnsemble(consistency=1))"

-output <output file containing input sensitivity >

This command line requires the path to BCL generated model files, output bin file, and name of the output file. The output file will have the input sensitivity for our model.