**Capstone Project Final Report**

**Model mutant p53 transcriptional activity (active vs inactive) using features from biophysical simulations**

**Background and Problem**

Over 6 million people worldwide die of cancer each year. Cancer is caused by the accumulation of genetic mutations in two critical pathways: normal cell growth and programmed cell death (apoptosis). Defects in the cell growth pathway can result in uncontrolled cellular proliferation - tumors. Tumor suppressor proteins such as p53 normally trigger cell death in affected cells and destroy the tumor. Mutations in the p53 gene occur in more than 50% of human cancers.

Studies have shown that p53 mutants can be reactivated through second site suppressor (‘cancer rescue’) mutations. Reactivated p53 holds great therapeutic promise because animal models have shown that reintroduction of active p53, even in advanced tumors, leads to tumor regression. There have been many efforts to find small molecule drugs that mimic the cancer rescue effect. Despite some promising discoveries, studying a large and more diverse collection of cancer rescue mutations that reactivate p53 cancer mutants is required to understand the p53 mutation-structure-function relationship.

Unfortunately, testing all possible mutation combinations to determine their cancer rescue effects experimentally is infeasible due to time and expense. Therefore, it would be very desirable to have a computer model to run in silico experiments on virtual mutants. Such a model could narrow down the list of likely cancer rescue mutants to a number that reasonably could be tested in the laboratory.

**Data set**

The data set for this project is from the UCI Machine Learning Repository.

<https://archive.ics.uci.edu/ml/datasets/p53+Mutants>

Biophysical models of mutant p53 proteins yield features which can be used to predict p53 transcriptional activity. All class labels were determined experimentally.

The data consists of 16,772 instances and 5409 attributes per instance. Attributes 1-4826 represent 2D electrostatic and surface based features. 4827-5408 represents 3D distance based features. The 5409th attribute is the class attribute which is either active or inactive. The class labels are to be interpreted as follows: ‘active’ represents transcriptionally competent, active p53 whereas the ‘inactive’ label represents cancerous, inactive p53.

**Solution and Client**

This is a binary classification problem – classifying the mutations as active or inactive. Identifying the features that play a crucial role in this classification will also help in understanding the mutations.

This will help cancer researchers in the biotech or pharma industry working specifically on protein engineering, drug development or basic science research.

**Relevant Papers**

Danziger, S.A., Baronio, R., Ho, L., Hall, L., Salmon, K., Hatfield, G.W., Kaiser, P., and Lathrop, R.H. (2009) Predicting Positive p53 Cancer Rescue Regions Using Most Informative Positive (MIP) Active Learning, PLOS Computational Biology, 5(9), e1000498  
  
Danziger, S.A., Zeng, J., Wang, Y., Brachmann, R.K. and Lathrop, R.H. (2007) Choosing where to look next in a mutation sequence space: Active Learning of informative p53 cancer rescue mutants, Bioinformatics, 23(13), 104-114.  
  
Danziger, S.A., Swamidass, S.J., Zeng, J., Dearth, L.R., Lu, Q., Chen, J.H., Cheng, J., Hoang, V.P., Saigo, H., Luo, R., Baldi, P., Brachmann, R.K. and Lathrop, R.H. (2006) Functional census of mutation sequence spaces: the example of p53 cancer rescue mutants, IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM, 3, 114-125.

**Exploratory Data Analysis**

**Download and Processing**

The data set consisted of 2 files – the data file, K9.data and the K9.instance.tags file that contains the names of the mutations. Both files were downloaded to the local hard drive. The .data file was read in using pd.read\_csv and saved to a dataframe, ‘df’. It was more than 1 GB in size. df.info and df.dtypes gave details about the number of rows, columns and also the data type of the features. There are 31,420 entries and 5410 features and all the columns are of float type except for one which is an object type. The instace.tags file was also downloaded as a data frame and added as an additional column ‘mutations’ to the dataframe, df.

df.head() gave a preview to the first 5 rows. The columns or features are numbered, with no description. There are missing values in some rows/columns and an entire column (column no. 5409) of NaNs.

To minimize the confusion between numbered columns and rows, the prefix ‘2D’ was attached to the first 4826 column numbers and the prefix ‘3D’ to the 4827th – 5408th column using string methods to identify the 2D and 3D features. The 5409th column was labeled as ‘Type’ as it indicated the class label. The column with NaNs was dropped. There were also some rows with ‘?’. Since, there were no descriptions of the features available and there were 5408 of them for each mutation, it seemed best to drop all missing values after converting them to NaNs.

The ‘mutations’ column contains the mutation information for each entry/row.

For eg: ‘%a119e\_l125p’ denotes 2 point mutations separated by ‘\_’, the first one is ‘a119e’ and the second one is ‘l125p’. The ‘a119e’ is read as – amino acid ‘a’ at position 119 of the protein chain is replaced by amino acid ‘e’. Based on the number of ‘\_’, the ‘count’ of the mutations was determined and added as a column to the dataframe, df.

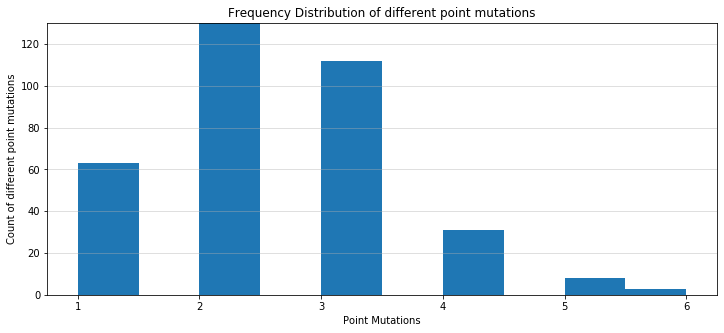
The dataset has 1-6 point mutations. In human mammary carcinoma, the mutation sites of p53 tend to occur within one single exon (coding region) or a short distance from another, implying that the distance of mutations may be of importance for affecting the function of p53 (1). Here we used 1,2,3,4 and 5 distance features for 1 pt., 2 pt., 3pt., 4pt., 5pt. and 6pt. mutations. The distance features represent the distance between adjacent mutations, ie., the distance between the first and second mutations (difference in position numbers), second and third, and so on. For e.g., in a 3pt. mutation, there will be 2 distance features – one, the distance between the first and second mutation positions and the second, the distance between the second and third mutation positions. Five such distance features named ‘distance1’,…,’distance5’ were added to the data set.

**Reference** Tao Huang, Shen Niu, Zhongping Xu, Yun Huang, Xiangyin Kong, Yu-Dong Cai and Kuo-Chen (2011) ChouPredicting transcriptional activity of multiple site p53 mutants based on hybrid properties. PLoS One 6(8): e22940.

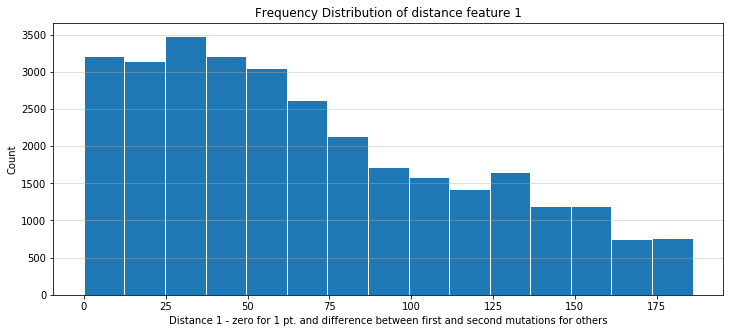
There were 31269 ‘inactive’ entries and only 151 ‘active’ entries in the dataset initially. But, after removing the missing values and NaNs, adding the ‘count’ and distance features, the data frame has 31159 entries with 5416 columns. The number of ‘inactive’ entries is now 31008 and 151 ‘active’ entries.

**Data Story**

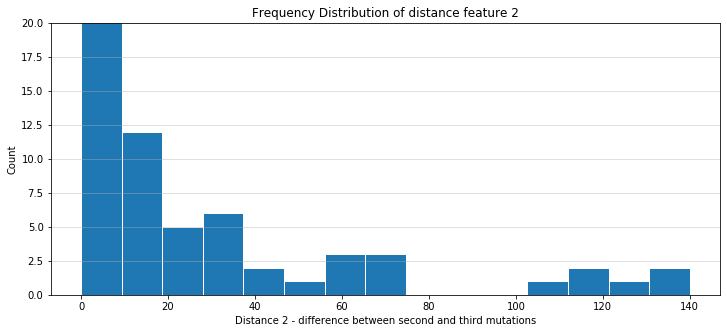
The 2D features are also known as Surface Property Maps and are just numbers representing the difference between the surface map of 'normal' amino acid (wild-type) and the mutated one. The 3D features, also known as Structure Distance Maps, are representing difference in magnitude of the distance changes in the 3D structure. This kind of multidimensional data is very hard to visualize for any specific feature correlation. We can try feature selection and dimensionality reduction on the data set and then try to visualize the key components. These features are bound to have highly correlated variables and dropping the highly correlated variables is part of feature selection.



Since there are multiple sites (1 pt to 6 pt) of mutations, we can see the distribution of the data set among these different types by plotting a histogram. The histogram shows that there are close to 31000 entries for the 2 pt mutation. The 3 pt and 1 pt have approx. 110 and 60 entries. The rest are represented by 30, 8 and 3 entries for 4 pt, 5 pt, and 6 pt mutations. Though, the data set is skewed with 31008 'inactive' and 151 'active' entries and we have both classes represented in all types of mutations. This distribution also tells us which ‘distance feature’ we can expect to be more prevalent.



Since, there are 31000 2 pt mutations; the ‘distance1’ will have the most number of entries. The distance features were plotted to observe the range of the differences between adjacent mutations. The 'distance1' feature which contains the difference in position between the first and second mutations shows that half of them lie between 0-90.



The 'distance2' feature represents the distance between the second and third mutations. This shows an even narrower range - most values are between 0-10 i.e., the adjacent mutations are really close most of the time. The 'distance3', 'distance4' and 'distance5' features were also seen to give a similar result. It is safe to say that in multiple mutations (more than 2 pt) adjacent mutations are mostly located close together - within a distance of 0-20.

The distributions of the active and inactive classes among the different mutations are in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mutations Count | Inactive | Active | Total |  |
| 1 | 55 | 8 | 63 |
| 2 | 30877 | 65 | 30942 |
| 3 | 49 | 63 | 112 |
| 4 | 24 | 7 | 31 |
| 5 | 2 | 6 | 8 |
| 6 | 1 | 2 | 3 |

**Statistical Analysis**

**Feature Selection and Feature Extraction**

There are many methods of feature selection – filter methods, wrapper methods and embedded methods. While the filter method may not be very accurate, the other two methods are computationally expensive and work when there are not more than 20 - 30 features.

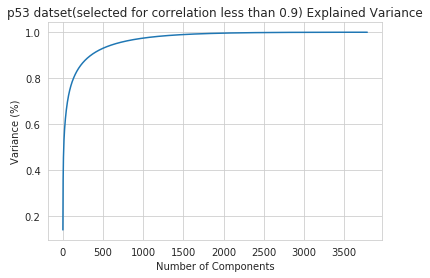
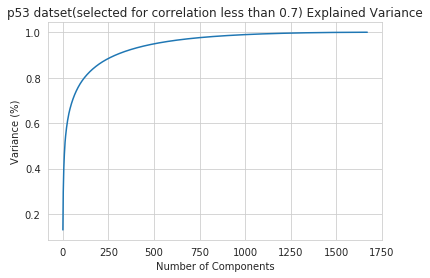
Principal component analysis is a technique for feature extraction — so it combines our input variables in a specific way, then we can drop the “least important” variables while still retaining the most valuable parts of all of the variables. PCA also helps in visualizing multidimensional data. PCA reduces data by geometrically projecting them onto lower dimensions called principal components (PCs), with the goal of finding the best summary of the data using a limited number of PCs.

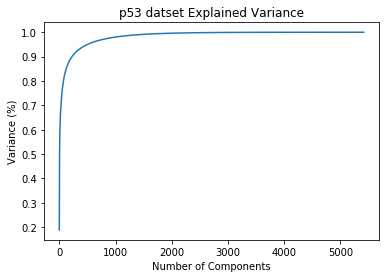
In this data set, the 2D features are numbers representing the difference between the surface map of 'normal' amino acid (wild-type) and the mutated one. The 3D features represent the difference in magnitude of the distance changes in the 3D structure. This kind of multidimensional data is very hard to visualize for any specific feature correlation. We can try feature selection and dimensionality reduction on the data set and then try to visualize the key components. These features are bound to have highly correlated variables and dropping the highly correlated variables is part of feature selection – filter method.

The correlation matrix of the numerical predictors was generated and the features showing high correlation are dropped - a correlation coefficient of 1 shows maximum correlation. Two different cut off scores were chosen – correlation coefficients of 0.7 and 0.9 - and the selected dataset from both cut offs were analyzed. When the cut off was 0.7, we dropped 3744 features from 5414 features whereas with a cut off of 0.9, we dropped 1625 features.

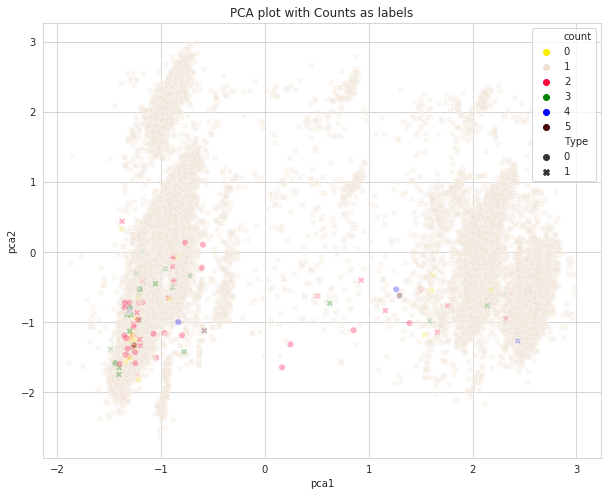
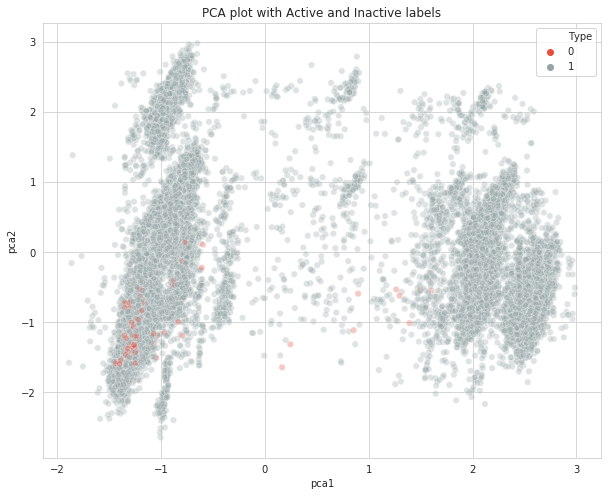
The numerical features of these 2 selected datasets and the entire data set was then standardized using the minmax\_scale. This transforms the features by scaling to a range between 0 and 1.

The scaled data were converted to data frames and then used for PCA. The PCA model was then analyzed for their explained variance ratio and a cumulative of the variation is plotted. This graph shows the number of components along the X axis and the percentage of variation explained along the Y axis. This graph was the same across all the 3 data sets – 2 datasets selected from correlation matrix cut offs and the entire dataset. Approx. 250 components were needed to explain 90% of the variation in data. Considering this, we can use just the PCA analysis for the feature selection/dimensionality reduction instead of using the correlation matrix and a cut off.





The PCA plots were plotted using the first 2 components. In the first plot, the ‘Type’ variable was also plotted as different colors. In the second PCA plot, the ‘count’ variable was the third dimension and the ‘Type’ was also included as different style for ‘active’ and ‘inactive’.

  
 Since the data is skewed to begin with and only 30% of the variation is explained by the 2 principal components, we cannot see a good partitioning of the clusters but we can at least visualize the data.

The next step would be to run a baseline model using different classifying algorithms and use cross validation to get a good prediction score.

**In-Depth Analysis**

Once, the data set has been cleaned and explored, it is ready to be used for machine learning models. We have a binary classification problem – the proportion of the active class is 0.5 to 99.5 of the inactive class. The tendency of most classifiers is to predict the majority class. Class imbalance is addressed in every step of model building – from splitting of the data set to train/test sets to adding weights by the classifiers.

The process of in depth analysis can be divided into Splitting the data, Performance criteria, Baseline models, Hyperparametric tuning and Cross Validation, Combining Models.

**Splitting the data**

In addition to the preprocessed and scaled data, PCA is a good way to speed up the fitting of an algorithm. If using the PCs instead of the actual data proves to have approximately the same performance and is faster, it can be used for parametric tuning and crossvalidation where different parameter combinations are checked using RandomSearchCV. My selections of models are based on primarily, performance, but also on computational resources - hence, the use of PCs.

The preprocessed scaled data set is used split into training and test sets – with 0.7:0.3 ratio and because of the imbalance, the split is made in such a way that there are both classes in the training and testing data. The training set is then fit and transformed into the principal component space while the test set are only transformed. We have 2 training sets and 2 test sets – one actual data and the other PCA transformed – for fitting a model.

**Performance Criteria**

Model performance is usually evaluated on the accuracy of the predictions. In an imbalanced data set, the accuracy is not an adequate metric. When the prediction of the minority class is equally important, the precision and recall for that class seems to be the better metric. F1 Score combines recall and precision using the harmonic mean. The F1 score can be averaged in 3 different ways – ‘micro’, ‘macro’ and ‘weighted’. The ‘macro’ average calculates the metrics for each label and finds their un-weighted mean without taking class imbalance into account. Since, the prediction of the minority class is important (predicting the active mutations), we choose the F1 macro score.

Another metric that is used is the Matthews Correlation Coefficient (MCC). This is used in binary classifications and like the F1 score uses the true and false positives and negatives but is a more balanced measure which can also be used if class sizes are different, It’s a value between -1 and +1, -1 representing an inverse prediction, 0 an average random prediction and +1, a perfect prediction.

I have used the F1 macro average values and the MCC score along with an eye on the precision and recall values for the class 1 predictions as a performance evaluator.

**Baseline Models**

To begin with, for this binary classification, I have selected to work with 6 classifiers – Logistic Regression (LG), Naïve Bayes (NB), Random Forest (RF), Support Vector Machine (SVM), Gradient Boosting (GB) and XGBoosting (XGB). I chose these classifiers to run a baseline model and based on their performance selected the classifiers for hyperparameter tuning and cross validation.

Here is a table showing the baseline performance of all the classifiers. The precision and recall are values for the minority class 1 (active) only. The colored ones represent the classifiers selected for hyperparameter tuning.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models | Precision | Recall | f1 macro | MCC |
| LG | 0.28 | 0.78 | 0.70 | 0.46 |
| NB | 0.01 | 0.80 | 0.42 | 0.07 |
| RF | 0.50 | 0.11 | 0.59 | 0.23 |
| SVM-l | 0.51 | 0.69 | 0.79 | 0.59 |
| GB | 0.34 | 0.33 | 0.67 | 0.33 |
| XGB | 0.59 | 0.36 | 0.72 | 0.46 |

Based on the above measures, I selected LG, RF, SVM and XGB for further tuning their parameters to see if they could perform better than their baseline model. Though Random Forest did not have a very good recall or MCC score, I wanted to see if tuning will help its performance as they are very sensitive to imbalanced data sets. SVM – I tried the ‘linear’ and ‘rbf’ kernel but the ‘linear’ kernel performed way better than the ‘rbf’.

The PCs did not perform as well as the actual dataset and always took as much time in fitting the algorithm. This could be because of the class imbalance proportion and the number of PCs – only 0.5 % of the data set belongs to class 1 and we chose PCs that explained 97% of the data set.

**Hyperparameter Tuning and Cross Validation**

Hyperparameters are parameters that are external to the model and highly responsible for getting good performance with models. These cannot be learned from the training process and express high level properties of the model. It determines the complexity of the model or the speed of learning. Finding the right combination of parameters involves the RandomSearch or GridSearch across Cartesian products of different sets of hyperparameters.

Cross Validation is a technique to identify how well our model performed and a need to test that our model is well trained without any underfitting or overfitting. I have used the StratifiedKFold cross validation keeping the class imbalance in mind and with K= 5.

For each model we created a dictionary of different parameters with different values and implemented the RandomizzedSearchCV which randomly selects the parameter combinations and also does a stratified KFold and the best model is selected by their scoring – f1 macro was used.

Parameter grid for different classifiers

|  |  |
| --- | --- |
| Models | Parameters that were tested |
| Logistic Regression | Penalty – ‘l1’, ‘l2’; C – 0.001,0.01,0.1,1,10 |
| Random Forests | n-estimators – 10,20,30,40,50,60,70,80,90,100,200,300  max\_features - 10,20,30,40,50,60,70,80,90,100,200 |
| Support Vector Machines | linear; C – 0.1,1,10,100,1000  rbf; C – 0,1; gamma – auto |
| XGBoost | Learning\_rate – 0.01,0.1,0.05; gamma – 0,1,5; subsample – 0.8,1; scale\_pos\_weight – 1, 99 |

Here is a table showing how the tuning increased the performance of the selected classifiers. The colors show the scores that improved over the baseline.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models | Precision | Recall | F1 macro | MCC |
| LG | 0.42 | 0.76 | 0.77 | 0.56 |
| RF | 0.55 | 0.13 | 0.61 | 0.27 |
| SVM-l | 0.51 | 0.69 | 0.79 | 0.59 |
| XGB | 0.49 | 0.67 | 0.78 | 0.57 |

The next step would be to combine the Logistic Regression, SVM-linear and XGB models into an ensemble and see how the score improves.

**Ensemble Classifier**

Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms, After creating standalone models, a Voting classifier can be used to wrap the models and average the predictions of the submodels when asked to make predictions for new data. The predictions of the sub models can be weighted and there are advanced methods to learn how to weigh the predictions – this is called stacking.

Here we use the voting ensemble model for classification using the VotingClassifier. We made 2 types of classifiers – one with 3 sub models – LG, SVM-l, XGB (LSX) and another one with just 2 models SVM-l, XGB (SX). The classifiers were initially tested directly on the test data (test1) we had used so far. Then, the scaled data was also split again with another random\_state to create another train/test set (test2) and the classifiers were tested on this set too. We did the 5-fold stratified cross validation and the mean was taken.



We do not see a huge difference in the performance of both classifiers on the test sets.

Next, we trained the classifiers on the 2 train sets and then predicted the test set and evaluated their f1 macro and MCC scores. The following figures show the performance of the 2 ensemble classifiers in comparison to the 2 individual tuned sub models. There is a figure for the f1 score and one for the MCC score.





Though, the ensemble did slightly better than the sub models, the SVM-linear model was the best standalone model.

**Concluding thoughts**

* Making use of AWS and thereby increasing computational efficiency
* Class Imbalance problem – Oversampling or using SMOTE
* Using other algorithms like clustering to learn more about the data set
* Explore the parameter grid in some of the classifiers to get a better performance
* Use the ensemble classifier with weights or use other ensemble methods like stacking