**HiSeq model**

**Creating a proper dataset**

**01.**

We downloaded 3 datasets for genomic type from TCGA.

1. KIRC\_RPPA.data 🡪 Clear cell
2. KIRP\_RPPA.data 🡪 Papilary cell
3. KICH\_RPPA.data 🡪 Chromophobe

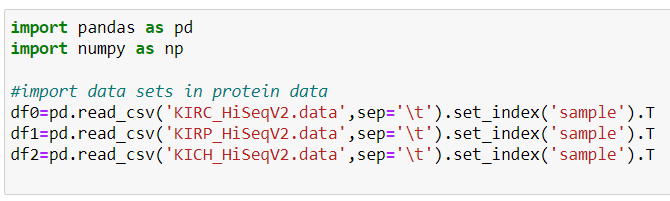
**02.**

In the datasets, columns contained patients and rows contained the genomic types (features). Since in TCGA, they divided the datasets for each subtype, in each dataset there were not a target row. So, we added a target row as ‘Subtype’ for each dataset with some integer values. Values can define as.

* ‘0’: KIRC
* ‘1’: KIRP
* ‘2’: KICH

**03.**

Then we started to create our models in python 3 environment. Here we imported all the datasets.

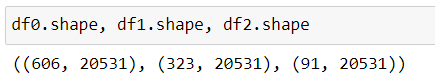


Reason for using set\_index is, if we did not define the index, by default Pandas will create an index for the DataFrame. It makes things a little more confusing, because by default the “index” is just the range of numbers starting at 0.

Here we swap the rows and columns using *transpose()* method to get the features and label into columns and patients into rows.

**04.**

Here are the shapes of each dataset.

****

We can clearly see that the number of features is not equal. We manually check some random features and they were same in all 3 datasets.

**05.**

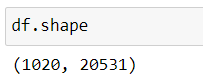
Now we need to combine these 3 datasets. Since the features are not exactly same in 3 datasets, we could not use merge method. So we used *pandas.concat* method for combine the datasets.



In concat, it simply merge the datasets if the labels (column names) are same and if not, it just adds the column. By adding columns which are not in other datasets, makes some ‘NULL’ values (missing values) to the dataset.

**06.**

Here is the shape of new data frame (Genomic data).



Here showing 1020 records. It just the combination of df0, df1 and df2 (606+323+91=1020).

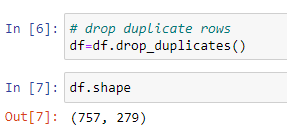
Here showing 20531 columns. Since in 3 datasets there were more similar columns and some different columns for each other, number of columns are differed from df0, df1, df2 and df. We actually need to remove non similar columns for further process. Since those non similar columns have missing values in combined datasets, we can easily check those values In the prepossessing steps.

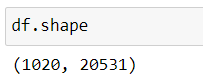
**Data Preprocessing**

**Data Cleaning**

**01.**

First, we try to drop duplicate records.

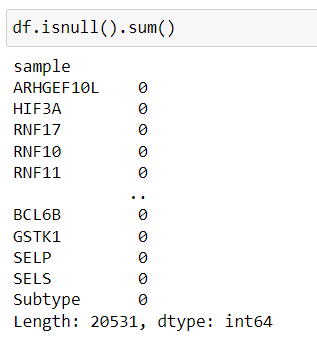




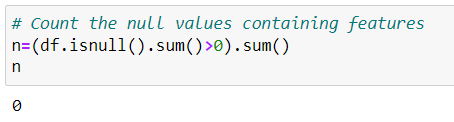
But it seems that there weren’t any duplicates (number of records are remaining same after dropping duplicates).

**02.**

Then we check for missing values.



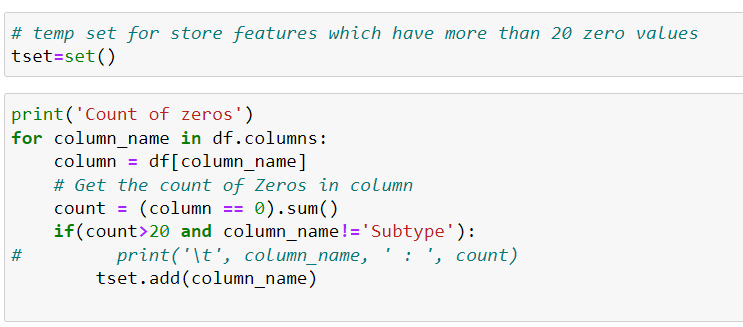
By this visualization it is difficult to take a decision about null values. So, we count the features which containing null values and try to visualize them.



There are no null values in this dataset. So doesn’t need to handle null values.

**03.**

Then we removed the features which has more than 20 zero values.

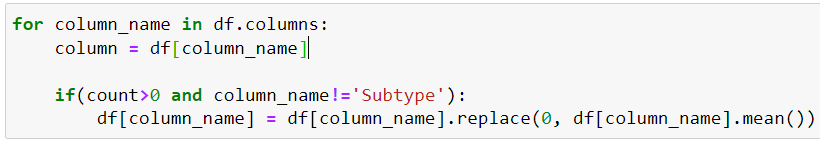


Removed feature count: 5,927

Remaining features: 14,604

**04.**

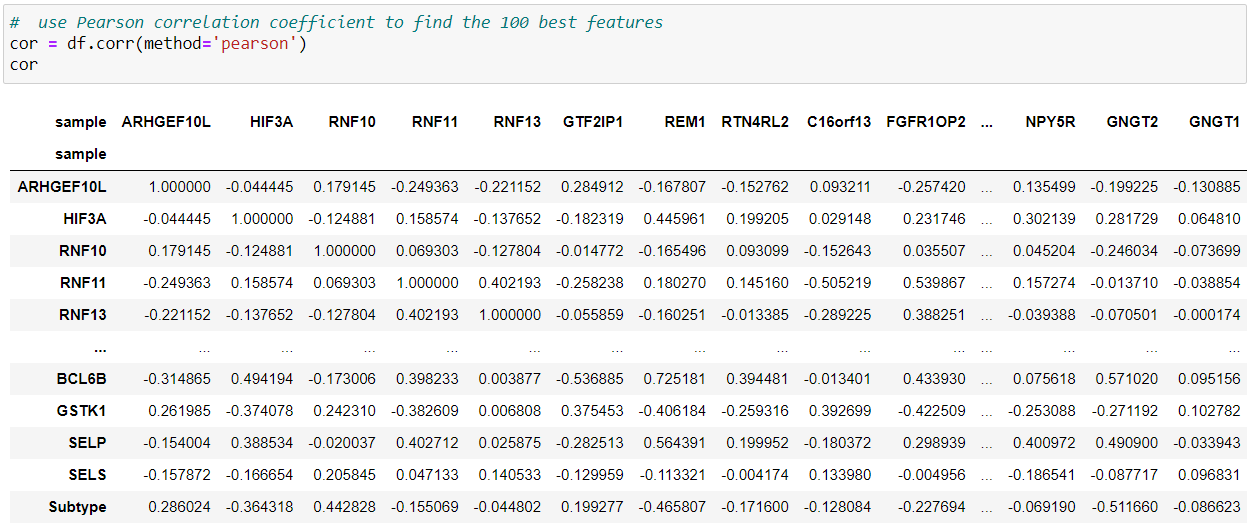
When we carefully refer the initial 3 datasets, we found no Boolean types features. Due to that, we replaced remaining zero values with the mean of relevant column(feature)



**Feature Selection - Pearson correlation coefficient**

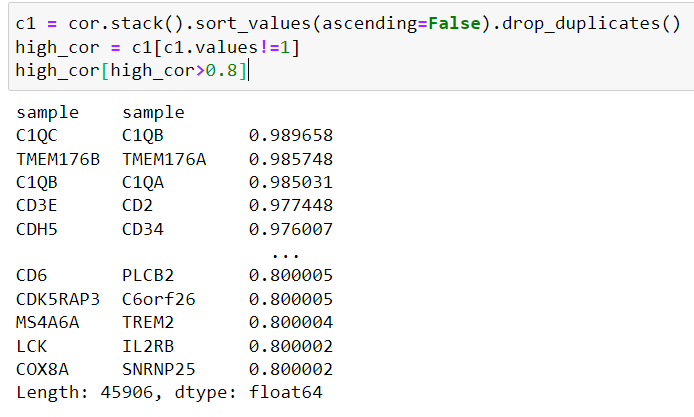
**01.**

Getting the correlation coefficients in dataset

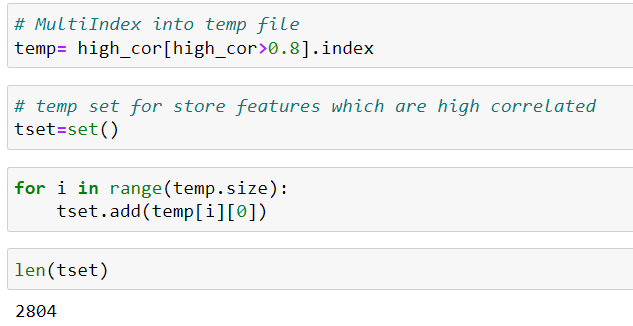


**02.**

In this step we removed the features which overly corelated. For that we take a threshold value of 0.80 and removed relevant features.



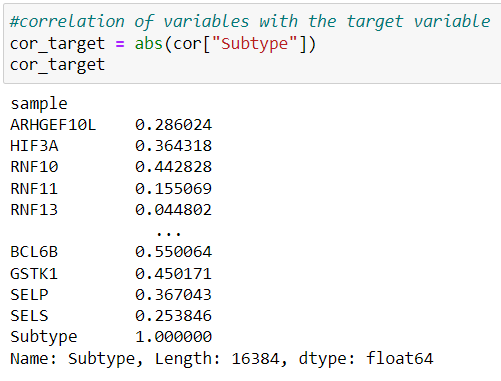
Since these values are high, it means these are somewhat linearly dependent with other features. hence those effect on the dependent variable is almost same. So we can drop one of them.



Removed feature count: 2804

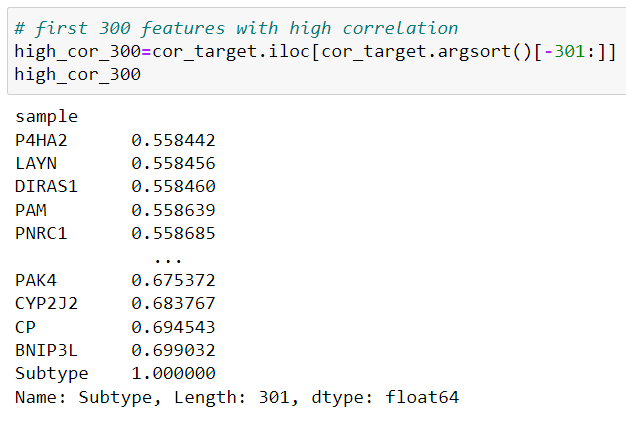
**2.**

Taking the absolute correlation of features with the target variable.



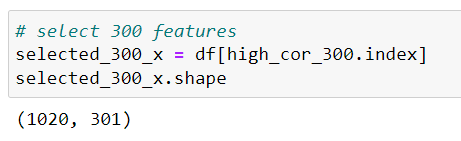
**3.**

Initially I got the best 300 features for further process.



Here features are taken in ascending order of coefficient. So, we can see that it is in the range of 0.558442 to 0.699032. Actually, this is a small correlation range. (Without the label)

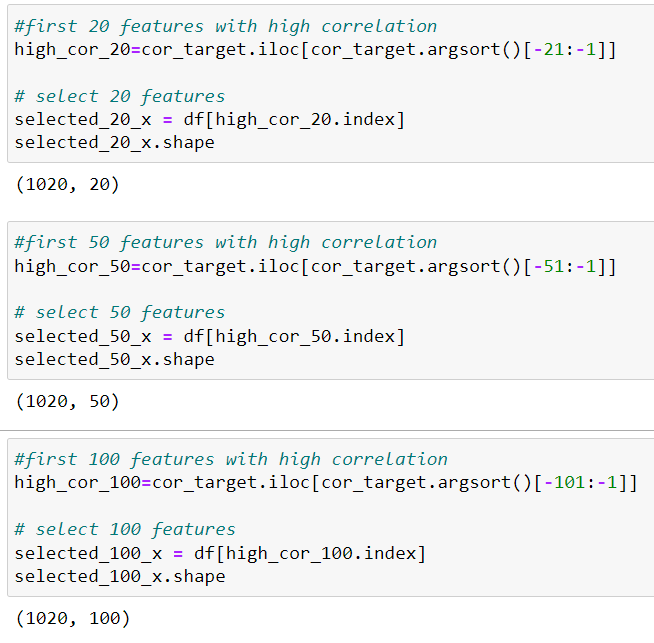
Selected 300 features. (Without the label)



**4.**

Likewise in the step 3, I build models with getting different feature amounts.

In model creation and evaluation step (In future), I considered the performance of the models and obtained 4 feature amounts (20, 50, 100)

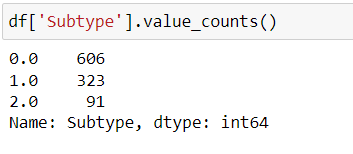


**Dataset balancing**

**01.**

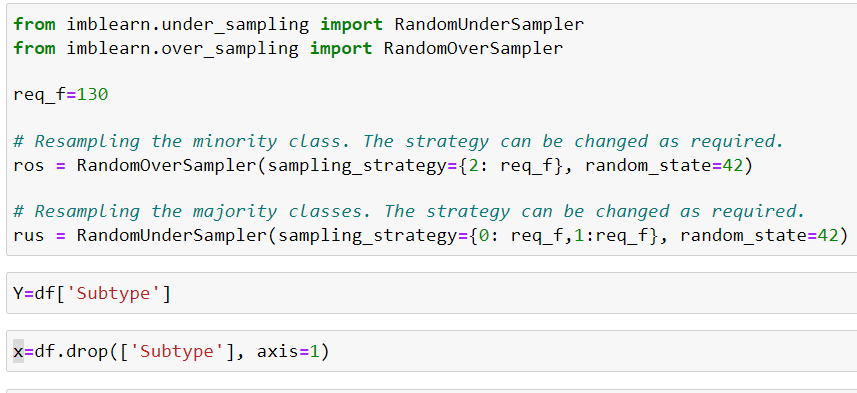
Then we viewed the value count of each subtype.

* Initial feature count from each class (Before sampling)



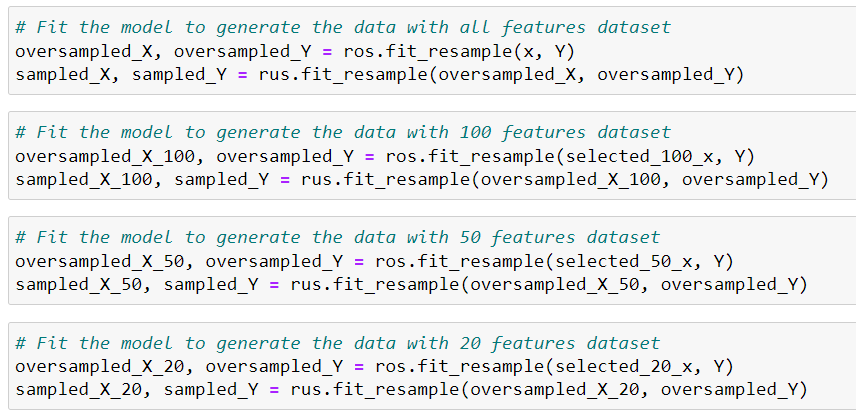
As it shows the data frame is highly imbalanced. So, I decided to handle these imbalancing. For that I used both Oversampling and Undersampling.

* Initial steps of sampling process

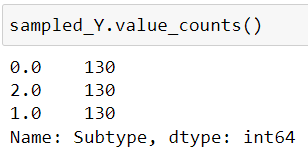


At last, I want to find out the required feature count which gives the best performing model. So, I did the sampling process for all data frames which have different number of features.

* Implement the sampling process for whole data frames



* Derived feature count from each class (After sampling)



**Test SVM**

**01.**

First, we try to find the best kernel to use for further process. Here we used cross validation for this test.

* Accuracy SVM with rbf kernel: 0.87 (+/- 0.07)
* **Accuracy SVM with poly kernel: 0.93 (+/- 0.05)**
* Accuracy SVM with linear kernel: 0.93 (+/- 0.06)
* Accuracy SVM with sigmoid kernel: 0.55 (+/- 0.08)

poly kernel performed well.

**02.**

Then, we find the accuracy without data transformation. Then we tried both standardization and normalization. So, we observed that without data transformation performed well with SVM. Here also used cross validation for this test.

* **Accuracy SVM without transformation: 0.93 (+/- 0.05)**
* Accuracy SVM with Standardization: 0.82 (+/- 0.11)
* Accuracy SVM with Normalization: 0.93 (+/- 0.06)

**03.**

Then we considered other datasets, which we created with different number of features (20,50,100) and evaluate those models using SVM by cross validations.

* Accuracy SVM using best 20 features: 0.86 (+/- 0.05)
* Accuracy SVM using best 50 features: 0.88 (+/- 0.03)
* Accuracy SVM using best 100 features: 0.88 (+/- 0.06)

We tried more than 100 number of features for a better performance.

* Accuracy SVM using best 150 features: 0.91 (+/- 0.04)
* Accuracy SVM using best 200 features: 0.92 (+/- 0.04)
* Accuracy SVM using best 230 features: 0.93 (+/- 0.04)
* Accuracy SVM using best 250 features: 0.93 (+/- 0.05)
* Accuracy SVM using best 280 features: 0.93 (+/- 0.05)
* **Accuracy SVM using best 290 features: 0.94 (+/- 0.06)**

So, number of feature with 290 shows the best performance.

**Test RF**

**01.**

First, we try different combinations of parameters with the help of GridSearchCV. Here we tested best 3 parameters in RF and took the best combination.

* Best parameters are:
  + 'criterion': 'entropy'
  + 'max\_depth': 8,
  + 'n\_estimators': 500

Then we did the cross validation.

**02.**

Then, we find the accuracy without data transformation. After that we tried both standardization and normalization. But we observed all the performance were same. Reason is, **RF does not need a data transformation since it is a Tree-based model and they are not based on the distance where features have an effect on one another**.

* Accuracy RF with best\_300 params: 0.91 (+/- 0.07)

**03.**

Then we test the model for other datasets, which we created with different number of features (20, 50, 100) and evaluate those models’ using RF by cross validations.

* Accuracy RF using best 20 features: 0.88 (+/- 0.06)
* Accuracy RF using best 50 features: 0.91 (+/- 0.07)
* Accuracy RF using best 100 features: 0.90 (+/- 0.08)

Then we selected the feature amount larger than 100

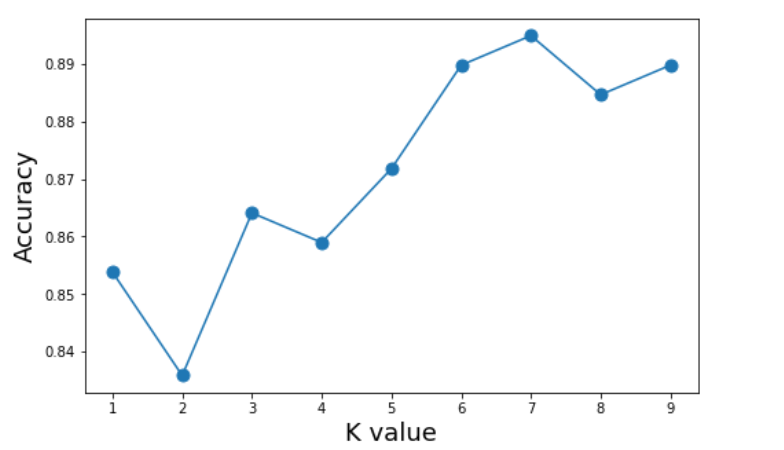
* Accuracy RF using best 150 features: 0.91 (+/- 0.07)
* Accuracy RF using best 200 features: 0.91 (+/- 0.07)
* Accuracy RF using best 250 features: 0.91 (+/- 0.07)
* **Accuracy RF using best 260 features: 0.93 (+/- 0.05)**
* Accuracy RF using best 270 features: 0.93 (+/- 0.05)
* Accuracy RF using best 280 features: 0.91 (+/- 0.06)
* Accuracy RF using best 290 features: 0.91 (+/- 0.07)

When comparing the above results, we can clearly say feature amount of 260, 270 will provide a better accuracy.

**Test KNN**

**01.**

First, we try to find the best k value to use for further process. Here accuracy is got from cross validation’s (cv=5) mean value.



Here k=7 is showing the best accuracy. And we have done the remaining process for k=7

**02.**

For k=7; We find the accuracy without data transformation. After that we tried both standardization and normalization. Here standardization performed well with KNN. Here also for testing we used cross validation for this selection.

* Accuracy KNN without transformation: 0.89 (+/- 0.05)
* **Accuracy KNN with Standardization: 0.91 (+/- 0.05)**
* Accuracy KNN with Normalization: 0.90 (+/- 0.05)

**03.**

Then we continued without transformation for other datasets, which we created with different number of features (20,50,100) and evaluate those models using KNN by cross validations.

* Accuracy KNN using best 20 features: 0.87 (+/- 0.08)
* Accuracy KNN using best 50 features: 0.89 (+/- 0.06)
* Accuracy KNN using best 100 features: 0.91 (+/- 0.07)

Then we tried with feature amount larger than 100

* **Accuracy KNN using best 150 features: 0.91 (+/- 0.03)**
* Accuracy KNN using best 200 features: 0.91 (+/- 0.07)
* Accuracy KNN using best 230 features: 0.91 (+/- 0.06)
* Accuracy KNN using best 250 features: 0.91 (+/- 0.06)
* Accuracy KNN using best 270 features: 0.89 (+/- 0.05)
* Accuracy KNN using best 280 features: 0.89 (+/- 0.06)
* Accuracy KNN using best 290 features: 0.90 (+/- 0.05)

Here 150 gives the best performance

**Overall Results – Genomic Data**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of features** | **Cross Validation Score** | | |
| **SVM (Without transformation)** | **RF** | **KNN (Normalized with min max scaler)** |
| 300 | 0.93 (+/- 0.05) | 0.91 (+/- 0.07) | 0.91 (+/- 0.05) |
| 290 | **0.94 (+/- 0.06)** | 0.91 (+/- 0.07) | 0.90 (+/- 0.05) |
| 280 | 0.93 (+/- 0.05) | 0.91 (+/- 0.06) | 0.89 (+/- 0.06) |
| 270 |  | 0.93 (+/- 0.05) | 0.89 (+/- 0.05) |
| 260 |  | 0.93 (+/- 0.05) |  |
| 250 | 0.93 (+/- 0.05) | 0.91 (+/- 0.07) | 0.91 (+/- 0.06) |
| 230 | 0.93 (+/- 0.04) |  | 0.91 (+/- 0.06) |
| 200 | 0.92 (+/- 0.04) | 0.91 (+/- 0.07) | 0.91 (+/- 0.07) |
| 150 | 0.91 (+/- 0.04) | 0.91 (+/- 0.07) | 0.91 (+/- 0.03) |
| 100 | 0.88 (+/- 0.06) | 0.90 (+/- 0.08) | 0.91 (+/- 0.07) |
| 50 | 0.88 (+/- 0.03) | 0.91 (+/- 0.07) | 0.89 (+/- 0.06) |
| 20 | 0.86 (+/- 0.05) | 0.88 (+/- 0.06) | 0.87 (+/- 0.08) |

Highest accuracy we obtained is around 94% and SVM abled to provide that. And the number of features we used is 290. For this we didn’t used any transformation method. Finally, we diced to select SVM as the best model for Genomic data.

* Here below are the parameters we used for SVM
* kernel='poly'
* Selected features (290):

|  |  |
| --- | --- |
| 1 | RHBDD2 |
| 2 | MPI |
| 3 | IL18R1 |
| 4 | ARNT |
| 5 | PHF7 |
| 6 | C6orf150 |
| 7 | TTC27 |
| 8 | SPHK2 |
| 9 | SLC25A26 |
| 10 | PNCK |
| 11 | ABTB2 |
| 12 | RIPK1 |
| 13 | HPS3 |
| 14 | PDGFD |
| 15 | MYO1B |
| 16 | LOC100128822 |
| 17 | EIF2C1 |
| 18 | RAB11B |
| 19 | DIRC3 |
| 20 | UBD |
| 21 | ARHGAP8 |
| 22 | SFT2D2 |
| 23 | SCIN |
| 24 | C3orf1 |
| 25 | KCNJ8 |
| 26 | TPD52L1 |
| 27 | RBM22 |
| 28 | 7-Mar |
| 29 | RAB40B |
| 30 | BIRC3 |
| 31 | ALDOC |
| 32 | PDP1 |
| 33 | AFAP1 |
| 34 | KTELC1 |
| 35 | NRL |
| 36 | DCLRE1B |
| 37 | FAM186B |
| 38 | BCAT2 |
| 39 | FOXL1 |
| 40 | CXorf57 |
| 41 | ARHGEF18 |
| 42 | PDDC1 |
| 43 | AK3L1 |
| 44 | ATXN1 |
| 45 | TIPRL |
| 46 | AFF3 |
| 47 | C3orf64 |
| 48 | 7-Sep |
| 49 | TMEM116 |
| 50 | SLC15A4 |
| 51 | APLP2 |
| 52 | CAMK4 |
| 53 | OSBPL2 |
| 54 | DCUN1D3 |
| 55 | RAB11FIP4 |
| 56 | LMTK3 |
| 57 | DDIT4 |
| 58 | CMC1 |
| 59 | CMPK2 |
| 60 | RSU1 |
| 61 | SSR1 |
| 62 | TROVE2 |
| 63 | AKT2 |
| 64 | C3orf18 |
| 65 | PISD |
| 66 | PPIP5K1 |
| 67 | STRBP |
| 68 | C5orf15 |
| 69 | AP3S1 |
| 70 | ZNF792 |
| 71 | PLOD1 |
| 72 | SLC2A1 |
| 73 | OGDHL |
| 74 | TMEM22 |
| 75 | PGAM5 |
| 76 | AHSA1 |
| 77 | NMUR1 |
| 78 | IL10RB |
| 79 | CRTC1 |
| 80 | TOM1 |
| 81 | SMPD1 |
| 82 | C5orf36 |
| 83 | KLC3 |
| 84 | IL1RAP |
| 85 | ANO10 |
| 86 | BIK |
| 87 | TRDMT1 |
| 88 | PARD6A |
| 89 | KIAA0284 |
| 90 | ARHGAP39 |
| 91 | CHCHD7 |
| 92 | TMEM53 |
| 93 | C19orf26 |
| 94 | ZNF629 |
| 95 | PPM1H |
| 96 | JAG1 |
| 97 | HNMT |
| 98 | BNIP2 |
| 99 | TSSK6 |
| 100 | CCND1 |
| 101 | BCL7A |
| 102 | SLC25A12 |
| 103 | ALPK2 |
| 104 | ALS2CL |
| 105 | SGCB |
| 106 | PWWP2B |
| 107 | GDI2 |
| 108 | HSD17B11 |
| 109 | MN1 |
| 110 | PXMP4 |
| 111 | NEDD4 |
| 112 | POLR3C |
| 113 | DNAJA3 |
| 114 | DHCR7 |
| 115 | SLC9A9 |
| 116 | HTRA1 |
| 117 | HIP1R |
| 118 | MYO19 |
| 119 | IFIT5 |
| 120 | C15orf52 |
| 121 | DNAJC5 |
| 122 | RB1 |
| 123 | C22orf45 |
| 124 | FBXO17 |
| 125 | ACAD10 |
| 126 | DUSP1 |
| 127 | ANP32E |
| 128 | CUX1 |
| 129 | CHST15 |
| 130 | TLCD1 |
| 131 | ANKRD32 |
| 132 | KCTD3 |
| 133 | GPR85 |
| 134 | GORASP1 |
| 135 | MGAT3 |
| 136 | PCBP4 |
| 137 | FBXO2 |
| 138 | SLC29A2 |
| 139 | ACTR3B |
| 140 | CRLF3 |
| 141 | SRGAP2 |
| 142 | BCO2 |
| 143 | MSH6 |
| 144 | OSMR |
| 145 | FOXRED1 |
| 146 | UBAC1 |
| 147 | DOK7 |
| 148 | CCDC88A |
| 149 | FOXJ3 |
| 150 | PLSCR4 |
| 151 | PLCB1 |
| 152 | BCAR1 |
| 153 | DIRAS2 |
| 154 | ABL2 |
| 155 | EHD2 |
| 156 | C10orf41 |
| 157 | PAIP2 |
| 158 | KHDRBS1 |
| 159 | DDX28 |
| 160 | TM4SF18 |
| 161 | PPT1 |
| 162 | RBMS3 |
| 163 | WDFY1 |
| 164 | SIRT5 |
| 165 | CAPN1 |
| 166 | SERTAD2 |
| 167 | SERPINB9 |
| 168 | CD302 |
| 169 | RLF |
| 170 | GTPBP5 |
| 171 | C9orf21 |
| 172 | DEF8 |
| 173 | TMEM51 |
| 174 | INSIG2 |
| 175 | RALB |
| 176 | GIT2 |
| 177 | TMEM37 |
| 178 | PRPF40B |
| 179 | TAF5 |
| 180 | CADM4 |
| 181 | AIDA |
| 182 | LDHD |
| 183 | NAP1L4 |
| 184 | ACAD11 |
| 185 | SNX33 |
| 186 | RAB8B |
| 187 | PLEKHG2 |
| 188 | KCTD1 |
| 189 | ATP6V0E2 |
| 190 | ZNF777 |
| 191 | ANTXR2 |
| 192 | SLC25A38 |
| 193 | COQ10A |
| 194 | C3orf31 |
| 195 | TMEM8B |
| 196 | DDR1 |
| 197 | ABLIM3 |
| 198 | FYN |
| 199 | SDR39U1 |
| 200 | SRD5A3 |
| 201 | HMGB3 |
| 202 | C15orf59 |
| 203 | LPCAT1 |
| 204 | CALCOCO2 |
| 205 | MBNL2 |
| 206 | AAGAB |
| 207 | PPFIA4 |
| 208 | DTNB |
| 209 | FAM110B |
| 210 | TGFA |
| 211 | KIAA1522 |
| 212 | CUEDC1 |
| 213 | METRNL |
| 214 | CKB |
| 215 | ICK |
| 216 | FAM115C |
| 217 | KATNAL1 |
| 218 | ITGB1 |
| 219 | CD109 |
| 220 | HAGHL |
| 221 | KCNK3 |
| 222 | CASP10 |
| 223 | ARRDC3 |
| 224 | PAG1 |
| 225 | TMEM55B |
| 226 | REPIN1 |
| 227 | ADRA1B |
| 228 | ADCK1 |
| 229 | SCAMP4 |
| 230 | ProSAPiP1 |
| 231 | DARS |
| 232 | SHROOM4 |
| 233 | TMEM45A |
| 234 | SLC2A5 |
| 235 | C3orf58 |
| 236 | F8 |
| 237 | SLC39A14 |
| 238 | PSKH1 |
| 239 | C19orf52 |
| 240 | ZNF25 |
| 241 | STARD13 |
| 242 | FTO |
| 243 | LOC644538 |
| 244 | RNF219 |
| 245 | ADPRH |
| 246 | EHBP1 |
| 247 | FAM160B1 |
| 248 | DBN1 |
| 249 | TRPM4 |
| 250 | WIPI2 |
| 251 | LOX |
| 252 | MID1IP1 |
| 253 | GDE1 |
| 254 | C14orf159 |
| 255 | ZDHHC23 |
| 256 | TCP11L1 |
| 257 | RIT1 |
| 258 | ST14 |
| 259 | LLGL2 |
| 260 | C19orf73 |
| 261 | PPP3R1 |
| 262 | ACLY |
| 263 | BNIP3 |
| 264 | TM7SF2 |
| 265 | DYRK3 |
| 266 | CLEC16A |
| 267 | TMEM26 |
| 268 | DDX58 |
| 269 | PARP3 |
| 270 | ENPP3 |
| 271 | NR3C1 |
| 272 | C1orf53 |
| 273 | SYNGR3 |
| 274 | SLC25A35 |
| 275 | MAEA |
| 276 | SSPN |
| 277 | PDIA5 |
| 278 | EDN1 |
| 279 | KCNJ11 |
| 280 | PLEKHA2 |
| 281 | FAM155B |
| 282 | MSN |
| 283 | MXI1 |
| 284 | PYGB |
| 285 | CDK19 |
| 286 | TLR3 |
| 287 | PAK4 |
| 288 | CYP2J2 |
| 289 | CP |
| 290 | BNIP3L |