Predicting Cancer Survival

Simon Beymer and Shervone Mayes

December 5, 2019

github link: https://github.com/Tobormai/CS_4347_Project_1

1 Introduction

Work on machine learning in genomics includes predicting sequence specificity of DNA and RNA binding proteins and enhancer and cis-regulatory regions, methylation, gene expression, and control of splicing. The ENCODE project identifies the transcription start sites and cis-regulatory/enhancer elements and transcription-start site sequenecing in addition to helping understand DNA methylation patterns (Consortium et al., 2004). It's effectively used by computer vision and natural language processing by modeling regulatory elements of DNA. This technology has effectively modeled DNA to be the same in forward/reverse which allows for computer models to better model DNA and the base pairing.

By improving this technology and how computers can understand biology allows for better treatment of disease including cancer. Cancer is the second most common killer of people who live in the United States. Decades of research have been conducted in order to better understand the disease and treatment. Both pharmaceutical and technological advances have been essential to improving treatment on cancer. Collaboration on work on cancer has allowed researchers to work together. One such example is The Cancer Genome Atlas dataset which has genomic information for patients with various types of cancer. The Cancer Genome Atlas is a huge program that combines work between the National Cancer Institute and the National Human Genome Research Institute that contains over 2.5 pentabytes of data (Tomczak, Czerwińska, & Wiznerowicz, 2015).This information allows researchers to understand what's going on in cancer cells better. Understanding what genetic variants are present in a genome and predicting the phenotypes is an example and has produced a new software: DeepVariant (Poplin et al.,

2018). This has been used to look at the cellular processes by looking at the DNA sequence. Another paper on looks at simplifying the genomic information that is input to classify cancer types based on DNA (Yuan et al., 2016). This examines somatic point mutations of known cancer types and relies on the Cancer Genome Atlas database. These two studies provide software that allows health care teams to make more informed choices by applying these technologies. Using machine learning in cancer patients has improved the outcome of their treatment by 15 to 20%. Previous researchers incorported genomic data, age, weight, diet, and high-risk habits into analysis for highest accuracy (Kourou, Exarchos, Exarchos, Karamouzis, & Fotiadis, 2015).

Lots of different projects have been conducted on applying machine learning for disease prognosis including, breast cancer, leukemia, colorectal cancer (Lin et al. 2018, Ferroni et al. 2019, Gründner et al. 2018, Pan et al. 2017). These studies use a variety of factors including cytogenic and somatic point mutations as well as age (Lin et al., 2018) Applying these developed models has helped with prognosis and live treatment for patients with breast cancer (Ferroni et al., 2019). This has also helped with prediction of relapse of Acute Lymphoid Leukemia recently (Pan et al., 2017). These models have also been able to predict the efficacy of cancer treatment including chemotherapy and radioactivity treatments (Gründner et al., 2018). One paper looked at DNA microarrays of 25,000 genes and used Pearson's correlation coefficient to find the genes with highest expression that was associated with increased malignancy (Xu, Zhang, Zou, Wang, & Li, 2012). After reducing their input genes to 300, they used reduced feature elimination and found 50 genes that were the most predictive of breast cancer survival (Xu et al., 2012). This suggests that genomic information can be incredibly important in these Machine Learning models. Semi-supervised learning demonstrated that it has been a good model for predicting cancer survival, especially when survival labels are incomplete (Park et al., 2013). For measures of accuracy SSL outperformed ANN and SVM in terms of accuracy, sensitivity, specificity, but SVM performed better with a higher AUC.

The problem that we seek to examine here is building a general model of predicting cancer survival. Other studies have been able to develop predictions towards one specific form of cancer, but we seek to build models that are able to handle robust data and can be generalized towards varaious types of cancers. We are developing several different algorithms to predict cancer survival and then looking at comparing the efficacy of these to other algorithms that have been published.

For our machine learning project, we used an online published dataset that has clinical information from over ten thousand individuals in addition to the mutation types of various cancer-related genes of various types of

cancers. We performed preprocessing to get the data into a usable format and then used a neural network classification algorithm and a logistic regression to predict whether these individuals would survive. The logistic regression achieved an accuracy of 54% and the neural network achieved an accuracy of 58%.

2 Problem Description

Our problem is to build a model that predicts cancer survival. We tried a few different datasets to be able to do this. The ideal dataset would have been through the Cancer Genome Atlas which contains 2.5 pentabytes of data. However, this data is restricted to individuals with access, which is, unfortunately, not us. I did get access to another online database, but this only contained clinical information about the patients. We used a study with the most individuals from www.cbioportal.org as we believed that this would give us the most data to train our robust model.

We are using data available from a 2017 study that genotyped cancerous tumors in over 10,000 individuals and followed them to see how long they lived for after their initial diagnosis (Zehir et al., 2017). This data incorporates sex, smoking history, primary tumor location, metastatic site, and other variables that can be useful. In total, this data contains 414 genes that are associated with cancer and also each of their mutation types. We initially combined the genes and their mutation type to investigate the effects that one mutation may have over another in the same gene. However, we don't have enough data to examine how any specific single nucleotide change would affect survivability. The initial data set contains information from 10,129 individuals and the different mutations that they have in addition to the clinical data that is provided about them.

We were initially a bit stuck on getting the data into a format that can be used for machine learning. There were multiple files that contained different types of information. These datasets were joined on patient-id using pandas in python or VLOOKUP in Excel and saved as a CSV. The CSV was uploaded to Jupyter Notebook to represent the dataframe to allow further preprocessing. In order to implement the Machine Learning techniques, we had to modify the data to include only meaningful variables in addition to truly independent features, a large sample size, and a binary dependent variable for classification. We initially had problems with deciding what features to keep. During feature selection, any variables deemed insignificant were dropped from the dataset entirely. The decision to drop certain variables was a combination of prior biology knowledge and data exploration per-

formed during preprocessing. Seaborn was utilized to visualize the dataset. Seaborn was used to generate bar charts and stacked bar charts to determine the best predictors for the dependent variable. We initially combined all of the types of mutations and the possible genes that the mutation was on. After turning this categorical data to numerical, we had 10,129 rows by 2,500 columns. However, we are concerned that this many independent variables would be difficult with not enough individuals for the algorithm to train on. Those columns deemed insignificant to determining patient outcome were dropped. Performed column-wise distribution of null values to determine if any further handling of missing data needed to be performed. Those columns with object type dtype were deemed possible categorical features in dataset. One-hot encoding strategy was used to convert categorical values into new column and assigned value of '1' or '0' (True/False). One-hot encoding solves problem of unequal weights given to categories within a feature, however, quickly encountered "curse of dimensionality" due to having many categories within dataset. We then narrowed down the number of features considerably. Next we balanced the dataset to contain an even number of dead and alive individuals to improve recall as an unbalanced dataset could improperly show good results. Our dataset contains 5546 rows and 416 independent variable columns, which allows for better training as reducing features allows the model to perform better.

We defined our dependent variable (y) to be VITAL_STATUS. The original data for VITAL_STATUS was not binary and was quantized to '1' for 'ALIVE' and '0' for 'DECEASED'. In fact, upon further inspection there was a significant portion of data that was considered categorical. We employed normalization to encode the dataset to eliminate instances of categorical variables and give continuous variables. Additionally, we used dictionaries to map other columns to desired binary values (e.g. SEX where male = 0, female = 1) which were then normalized. To try to combat issue of sparse dataset after one-hot encoding implemented, the genes that had an occurrence of less than 300 in the dataset were discarded prior to one-hot encoding. This reduced the number of columns.

With the dataset prepared, we implemented the logistic regression and the neural network. The artificial neural network was built using keras and we implemented pre-training on each successive layer to get better performance and to compare the performance between models. These were built successively upon themselves so the first model contained 1000 neurons and a sigmoid function to compare to the true output, then the second contained the previous 1000 nodes after training in addition to 500 new nodes and then the sigmoid function and so on. The total model contains initial dense layer of 1000 nodes, then a 500 node layer, then 400, 200, 100, 100, 50, 25, 16,

8, and a last layer of 1. The activation function is relu and there is a 50% dropout between layers. We used RMSProp with binary_cross entropy loss. We used an 80% 20% training test split and the model performed equally well with a 60% 40% split. We tried a variety of optimizers, dropout, batch normalization, activation functions and found that these parameters worked the best. We found that test performance stopped improving after around 4 epochs, so we implemented early stopping to get optimal results.

2.1 Bonus: Deriving the Properties of Neural Network Algorithm

Additionally, we looked at deriving the characteristics of the models by looking at the time complexity of the algorithms. Simon worked out the big O time complexity for a neural network which is as follows. For a neural network, the variables are put into the first layer of size a and then multiplied by the weights of dimensions a by the size of the second layer (b) as the first operation in forward propogation. This is done once for eaching training example (t).

$$K_{b,t} = W_{b,a} * J_{a,t} + b$$

which is a complexity of $t(a * b)$. Next, this runs through the relu activation function (f) to get the input for the next layer $f(K_{b,t}) = J_{b,t}$ which is of complexity $b * t$. Summing together the complexity to go from the first layer to the second is then $t(a * b) + t(b)$. This simplifies to $(b * t(a + 1))$.Similarly, the number of operations from the second, with layer size b, to the third, with layer size c, is $t(b * c) + t(c)$. This simplifies to $(c * t(b + 1))$. As c,t, and b get larger we can approximate $(a + 1)$ as a and $(b + 1)$ as b. Combining these two equations yields $(t(c * b) + (t(b * a)$. By simplifying,

$$t(a * b + b * c)$$

which is the complexity per epoch (n) for the first two layers. From this we can write the complexity for the feedforward propogation for j layers as

$$\sum_{i=1}^{i=(j-1)} n(t(m_i * m_{i+1})$$

where j is the number of layers, n is the number of epochs, t is the number of training examples, and m represents the number of nodes in a given layer.

Next, I explored the complexity of the backpropogation algorithm. Backpropogation starts with the cross entropy loss function where y is the pre-

dicted output and q is the true output:

$$L(y,q) = -\sum_{i=1}^{i=t} q * ln(y)$$

which is the summed error over all training samples. I will denote this function is used to compare the loss between the real and the expected outcome. The error is then

$$E_{x,t} = f'(K_{x,t})(\odot)(O_{x,t} * ln(Z_{x,t}))$$

where $f'(K_{x,t})$ goes back to the previous layer to update the function. Here, x is the number of nodes in the last layer and t is the number of training examples. The complexity for this is x*t for the element-wise subtraction plus the output of the f' function which is also x*t complexity. Then the change in weight is calculated by $D_{x,v} = E_{x,t} * J_{t,v}$ which has the complexity of x * t * v for matrix multiplication. The weights are then updated by $W_{x,v} = W_{x,v} - Dx, v$ which is complexity x*v. Adding these together gives $(t * x + t * x + x * t * v + x * v)$ which simplifies to $(t(x(v+1)) + x * v)$. As v and x get arbitrarily large, v+1 can be approximated by v and the two x*v can be joined into one term to give equation $(t(x * v)$ where x is the last layer and v is the second to last layer.

Next the complexity for updating the weights between the third to last to the second to last layer is $E_{v,t} = f'(K_{v,t}(\odot)(W_{x,v} * E_{v,t})$ which has complexity $(x * v * t + v * t)$. To update the weights is via function $D_{v,u} = E_{v,t} * J_{t,u}$ which has complexity $(v * t * u)$. Then to update the weights is $W_{v,u} = W_{v,u} - D_{v,u}$ which has complexity $(v * u)$. Summing these together gives complexity $(x * v * t + v * t + v * t * u + v * u)$. Simplifying gives $(t(v(x + 1 + u))) + v * u)$ which further simplifies to $(t(v(x + u)))$. The math is the same to go from a previous layer, b, to this layer, u, so $(t(u(v + b)))$. Combining these three equations yields

$$t * v * x + t(u(v + b) + t(v(x + u))$$

In big O time complexity this is

$$O(t(v * x + v * u + u * b))$$

which is per epoch. This can be written as a sum for j number of layers, n is the number of epochs, t number of training examples, and m number of nodes in a given layer.

$$O(\sum_{i=1}^{i=(j-1)} n(t(m_i * m_{i+1})))$$

This is the same as the feedforward propogation so this is the big O notation for the time complexity of the neural network algorithm.

In the case where all layers are the same number of nodes per layer, m, this can be written as

$$O(n(t(m^{j-1})))$$

. As j becomes larger, j-1 can be approximated by j, which gives $O(n(t(m^j))$. From this we derived the big O complexity for the neural network as

$$O(n^j)$$

where n is the number of nodes in a layer and j is the number of layers.

2.2 Bonus: Deriving the Properties of Logistic Regression Algorithm

This derivation for logistic regression model will focus on the binary response case as our project implementation had only two cases. We assume that the case of interest (or "true") is coded to 1, and the alternative case (or "false") is coded to 0. The logistic regression model assumes that the log-odds of an observation y can be expressed as a linear function of the K input variables x:

$$\log \frac{P(\mathbf{x})}{1 - P(\mathbf{x})} = \sum_{j=0}^{K} b_j x_j$$

Here, we add the constant term b_0, by setting $x_0 = 1$. This gives us K+1 parameters. The left hand side of the above equation is called the logit of P (hence, the name logistic regression).

Let's take the exponent of both sides of the logit equation. Let's take the exponent of both sides of the logit equation.

$$\frac{P(\mathbf{x})}{1 - P(\mathbf{x})} = \exp(\sum_{j=0}^{K} b_j x_j)$$

$$= \prod_{j=0}^{K} \exp(b_j x_j)$$

This immediately tells us that logistic models are multiplicative in their inputs (rather than additive, like a linear model), and it gives us a way to

interpret the coefficients. A useful fact about P(z) is that the derivative $P'(z) = P(z)(1–P(z))$ whose derivation can be achieved using chain rule.

$$P(z) = \frac{\exp z}{1 + \exp z} = (\exp z)(1 + \exp z)^{-1}$$

$$P'(z) = (\exp z)(1 + \exp z)^{-1} + (\exp z)(-1)(1 + \exp z)^{-2}(\exp z)$$

$$= \frac{(\exp z)(1 + \exp z)}{(1 + \exp z)^2} - \frac{(\exp z)^2}{(1 + \exp z)^2}$$

$$= \frac{\exp z}{(1 + \exp z)^2}$$

$$= \frac{\exp z}{1 + \exp z} \cdot \frac{1}{1 + \exp z}$$

$$= P(z)(1 - P(z))$$

Then take the gradient of P with respect to the set of coefficients b, rather than z. In that case, $P'(z) = P(z)(1–P(z))z$‘, where ‘ is the gradient taken with respect to b.

Maximize the log-likelihood, we take its gradient with respect to b:

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_{\substack{i=0 \\ y_i=1}}^{N} \frac{P_i'}{P_i} \mathbf{x}_i - \sum_{\substack{i=0 \\ y_i=0}}^{N} \frac{P_i'}{1 - P_i} \mathbf{x}_i$$

Note: maximum occurs where the gradient is zero.

Equation can be further expanded, recall that $P' = P(1 - P)$:

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_{\substack{i=i \\ y_i=1}}^{N} \frac{P_i(1-P_i)}{P_i}\mathbf{x}_i - \sum_{\substack{i=1 \\ y_i=0}}^{N} \frac{P_i(1-P_i)}{1-P_i}\mathbf{x}_i$$

$$= \sum_{\substack{i=1 \\ y_i=1}}^{N} (1-P_i)\mathbf{x}_i - \sum_{\substack{i=1 \\ y_i=0}}^{N} P_i\mathbf{x}_i$$

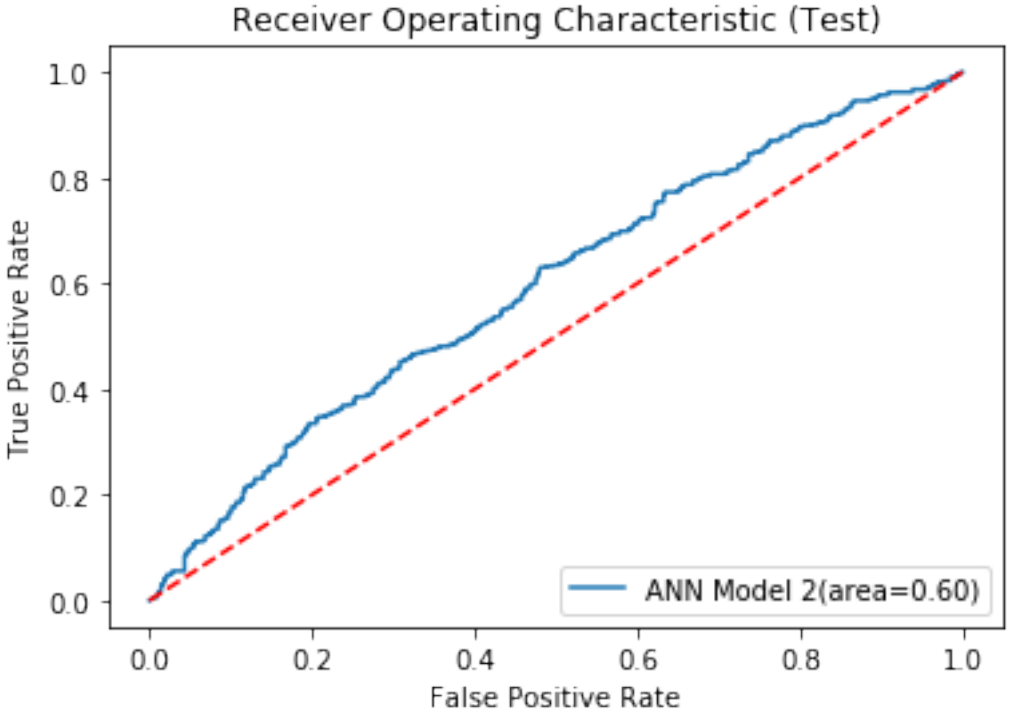
$$= \sum_{i=1}^{N} \left[y_i(1-P_i) - (1-y_i)P_i\right]\mathbf{x}_i$$

Logistic Regression Derivation Summary: 1. Logistic regression models are multiplicative in their inputs. 2. The exponent of each coefficient tells you how a unit change in that input variable affects the odds ratio of the response being true. 3. Logistic regression is coordinate-free: translations, rotations, and rescaling of the input variables will not affect the resulting probabilities. 4. Logistic regression preserves the marginal probabilities of the training data.

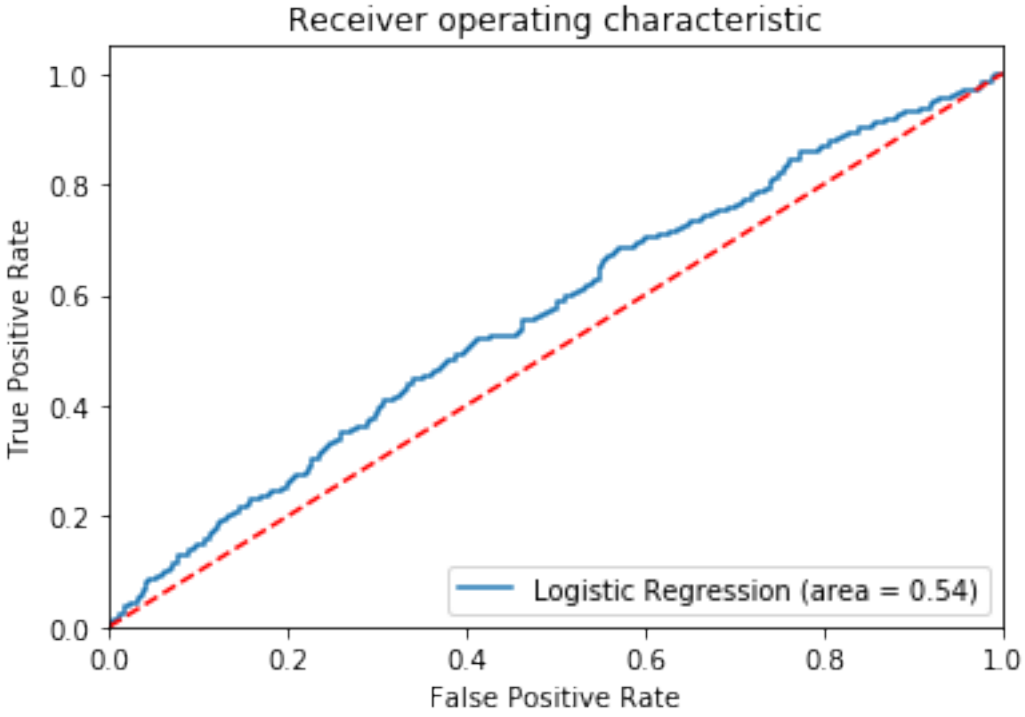
3 Results

We initially achieved an accuracy of approximately 70% for both the neural network and the logistic regression. However, we found that this was because the number of individuals who were alive wasn't balanced against the number of dead individuals such that 70%r were alive and 30% were dead. Therefore, the algorithm could get this accuracy by simply outputting that every individual would survive. However, the specificity and sensitivity were no better than a random guess for the neural network. After we balanced the number of individuals who lived against those who died, the models increased in terms of recall and increased the AUC for both models.

The neural network achieved an accuracy of 58%, an AUC of 0.60 while the Logistic Regression achieved an accuracy of 54%, and an AUC of 0.54.



We think that this low accuracy is due to the improper amount of data and a small training set. The model wasn't easily able to generalize its predictions about the training set to the testing set as the training accuracy could get as high as 80% after lots of successive training but this overfit the training as the accuracy on the testing set didn't improve. We are limited by the dataset that we were provided and our model would have likely performed better if we had more features such as the type of treatment that individuals received and the age at diagnosis.



4 Conclusion and Future Work

Both Shervone and Simon worked in parallel on the data and the models. Shervone worked on exploring the dataset while Simon initially implemented both a logistic regression and neural network. Shervone later implemented another logistic regression model and updated the neural network. Simon further updated the dataset to balance the number of individuals in each outcome. Simon derivied theoretical properties of the neural network algorithm and Shervone derived the theoretical properties of the logistic regression. Simon also wrote up much of the project reports.

From this project, we learned the importance of having a useful dataset. The most important step to effective machine learning is data pre-processing. After this, it's not difficult to build various machine learning models with a good data set. A lot of the backend work has been done by keras or tensorflow or pytorch which makes machine learning easy to get into. After this point, it's all about playing with the features and discovering what can be done in order to maximize the model's performance.

We compared our model with previous research. One study used SVM and Reduced Feature Elimination to identify 50 genes to predict breast cancer survivability with 97% accuracy (Xu et al., 2012).

Therefore, a future goal of this research would be to use a SVM with RFE as another algorithm to explore the optimal set of features to include. We will develop several different models to compare their efficacy. With this data set, we could use label encoded variables instead of dummy variables and compare results for improvements in accuracy. Additionally, we could explore resampling technique called SMOTE. Limitations of dataset size limited alternatives to explore fining tuning with such method. Additional tuning strategies, grid search and random search provide means of preventing overfitting.

The next step towards improving this project would be to get access to the Cancer Genome Atlas which has much more detailed information and more detailed clinical information to provide more features that the model can be trained on.

References

Consortium, E. P., et al. (2004). The encode (encyclopedia of dna elements) project. *Science*, *306*(5696), 636–640.

Ferroni, P., Zanzotto, F. M., Riondino, S., Scarpato, N., Guadagni, F., & Roselli, M. (2019). Breast cancer prognosis using a machine learning approach. *Cancers*, *11*(3), 328.

Gründner, J., Prokosch, H.-U., Stürzl, M., Croner, R., Christoph, J., & Toddenroth, D. (2018). Predicting clinical outcomes in colorectal cancer using machine learning. In *Mie* (pp. 101–105).

Kourou, K., Exarchos, T. P., Exarchos, K. P., Karamouzis, M. V., & Fotiadis, D. I. (2015). Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, *13*, 8–17.

Lin, M., Jaitly, V., Wang, I., Hu, Z., Chen, L., Wahed, M., ... others (2018). Application of deep learning on predicting prognosis of acute myeloid leukemia with cytogenetics, age, and mutations. *arXiv preprint arXiv:1810.13247*.

Pan, L., Liu, G., Lin, F., Zhong, S., Xia, H., Sun, X., & Liang, H. (2017). Machine learning applications for prediction of relapse in childhood acute lymphoblastic leukemia. *Scientific reports*, *7*(1), 7402.

Park, K., Ali, A., Kim, D., An, Y., Kim, M., & Shin, H. (2013). Robust predictive model for evaluating breast cancer survivability. *Engineering Applications of Artificial Intelligence*, *26*(9), 2194–2205.

Poplin, R., Chang, P.-C., Alexander, D., Schwartz, S., Colthurst, T., Ku, A., ... others (2018). A universal snp and small-indel variant caller using deep neural networks. *Nature biotechnology*, *36*(10), 983.

Tomczak, K., Czerwińska, P., & Wiznerowicz, M. (2015). The cancer genome atlas (tcga): an immeasurable source of knowledge. *Contemporary oncology*, *19*(1A), A68.

Xu, X., Zhang, Y., Zou, L., Wang, M., & Li, A. (2012). A gene signature for breast cancer prognosis using support vector machine. In *2012 5th international conference on biomedical engineering and informatics* (pp. 928–931).

Yuan, Y., Shi, Y., Li, C., Kim, J., Cai, W., Han, Z., & Feng, D. D. (2016). Deepgene: an advanced cancer type classifier based on deep learning and somatic point mutations. *BMC bioinformatics*, *17*(17), 476.

Zehir, A., Benayed, R., Shah, R. H., Syed, A., Middha, S., Kim, H. R., ... others (2017). Mutational landscape of metastatic cancer revealed from prospective clinical sequencing of 10,000 patients. *Nature medicine*, *23*(6), 703.