Validation of Biomarkers Predictive of Tumor Location in Coloadenocarcinoma, an Analysis of the TCGA COAD Dataset

Leopold von Seckendorff

2021-10-06

Contents

List of Figures

1 Table of Abbreviations

Abbreviation	Meaning
5-FU	5-Flurouracil
AKT	serine/threonine-specific protein kinase B
AMER1	APC membrane recruitment protein 1
APC	adenomatous polyposis coli
API	application programming interface
AUC	area under the curve
BRAF	B-RAF proto-oncogene, serine/threonine kinase
CMS	concensus molecular subtypes
CRC	colorectal cancer
csv	comma separated value
EGFR	epithelial growth factor receptor
ERK	extracellular signal-regulated kinases
EZH2	polycomb group protein enhancer of zeste homolog 2
FGF9	fibroblast growth factor 9
FPKM	fragments per kilobase of transcript per million mapped reads
FPKM-UQ	fragments per kilobase of transcript per million mapped reads upper quartile
GDC	NCI Genomic Data Commons
gzip	Gnu ZIP
HGF	hepatocyte growth factor
HOX	homeobox
json	JavaScript object notation
KiB	kibibit, 2^10 (1024) bits
KRAS	kirsten rat sarcoma
LMU	Ludwig Maximilians Universität München
MAF	mutation annotation format
MAPK	MAP kinase
MEIS3	meis homeobox 3
MEK	mitogen-activated protein kinase kinase
(c-)MET	mesenchymal epithelial transition (hepatocyte growth factor receptor)
MSI	microsatellite instability

Abbreviation	Meaning
NCI	National Cancer Institute
os	operating system
PD-L1	programmed death ligand 1
PLTP	phospholipid transfer protein
PRAC1	prostate cancer susceptibility candidate protein 1
RAF	rapidly accelerated fibrosarcoma
RAS	rat sarcoma
regex	regular expression
REPL	read-evaluate-print-loop
REST	representational state transfer
RF	random forest
ROC	reciever operating characteristic
RTK	receptor tyrosine kinase
RYR3	ryanodine receptor 3
SSH2	slingshot protein phosphatase 2
SVM	support vector machine
t-SNE	t-distributed stochastic neighbor embedding
TCGA	The Cancer Genome Atlas
UUID	universally unique identifier
xml	extensible markup language

2 Abstract

Tumor localization correlates with prognosis in coloadenocarcinoma, with aboral tumors having a better overall survival. This can be attributed to their better response to biologicals such as the anti-EGFR (epidermal growth factor receptor) cetuximab.

Since the localization of a tumor is trivially determined in a clinical setting, it remains a valuable surrogate parameter for predicting patient outcomes, though it is not a mechanistic explanation. Some possible explanations have been offered: it could be that aboral colonic epithelial cells respond differently to mutagenic stimuli, or that the variation in gut flora from adoral to aboral plays a role in tumor development or behavior. So far, there has been no consensus. By eliminating tumor localization as a confounder, since some aboral tumors behave and develop more like adoral tumors and vice versa, better treatment decisions would be possible. While being slightly more complicated than simply defining the tumor location, testing for a handful of mutations in a tumor specimen is a routine procedure and the increased predictive power of such a model would be of great value for making difficult treatment decisions. It would also represent a starting point for better understanding possible underlying molecular mechanisms.

It was hypothesized that –regardless of the causal relationships– this "sidedness" of coloadenocarcinomas could be reconstructed on a genomic and transcriptomic level. In order to test this hypothesis, data from the TCGA (Tumor Cancer Genome Atlas) database was used in a case-control study design to create expression profiles by training two distinct machine-learning algorithms to predict tumor location. The algorithms identified PRAC1, HOXB13, HOXC9, HOXC6, HOTAIR, PRAC2, and HOXC8 (all members of the homeobox gene family) as well as BST2, PLTP, FN1, ITLN1, and AREG as predictors of localisation. These finding corroborate previous research using various other methods and fit well into the established framework of previously published literature which solidifies the veracity of the machine-learning models as implemented.

As an additional benefit, the workflow for creating the genomic and transcriptomic profiles is very flexible and can be used for further analysis of the TCGA dataset.

3 Kurzzusammenfassung

Tumorlokalisation korreliert mit der Prognose Koloadenokarzinome, wobei aborale Tumore ein besseres Gesamtüberleben zeigen. Dies kann man auf deren bessere Antwort auf Biologika, wie das anti-EGFR (epidermal growth factor receptor) Medikament Cetuximab, zurückführen.

Da die Lokalisation eines Tumors im klinischen Alltag vergleichsweise unkompliziert festgestellt werden kann ist sie nach wie vor ein wichtiger Surrogatparameter für die Vorhersage von Therapieerfolg, obwohl sie keine mechanistische Erklärung ist. Einige mögliche Erklärungen wurden schon vorgeschlagen: es könnte sein, dass die aborale Kolonepithelzellen anders auf mutagene Stimuli reagieren, oder dass die Variation der Flora im Kolon von adoral nach aboral eine Rolle in der Tumorentwicklung und des Verhaltens spielt. Leider gibt es noch keinen Konsens. Die Hypothese wurde aufgestellt, dass –unabhängig von den kausalen Zusammenhängen– diese "Seitigkeit" der Koloadenokarzinomen auf genomischer und transkriptomischer Ebene rekonstruiert werden kann. Indem man die Tumorlokalisation als Confounder eliminiert könnten Therapieentscheidungen besser getroffen werden, da sich manche aborale Tumoren wie adorale Tumore entwickeln und verhalten, und umgekehrt. Obwohl etwas komplizierter als nur Tumorlokalisation zu bestimmen, ist die Suche nach einer Handvoll Mutationen in einer Tumorprobe ein klinisches Routineverfahren und die verbesserte Vorhersagekraft eines solchen Modells wäre wertvoll für schwierige Behandlungsentscheidungen. Es wäre auch ein Startpunkt für weitere Untersuchungen, um die zugrundeliegenden molekularen Mechanismen besser zu verstehen.

Die Hypothese wurde aufgestellt, dass die "Seitigkeit" der Koloadenokarzinome auf genomischer und transcriptomischer Ebene rekonstruiert werden könnte. Um die Hypothese zu überprüfen wurden Daten aus der TCGA (Tumor Cancer Genome Atlas) Datenbank in einer Fall-Kontroll-Studie verwendet, um Expressionsprofile mittels Machine-Learning-Algorithmen zu erarbeiten, welche die Tumorlokalisation vorhersagen können. Die Algorithmen identifizierten sowohl PRAC1, HOXB13, HOXC9, HOXC6, HOTAIR, PRAC2, und HOXC8 (alles Mitglieder der Homeobox Genfamilie) als auch BST2, PLTP, FN1, ITLN1, und AREG als Prediktoren der Tumorlokalisation. Diese Ergebnisse bestätigen bereits publizierte Erkenntnisse und bekräftigen somit die Genauigkeit der Machine-Learning-Algorithmen wie sie hier implementiert wurden.

Als zusätzlicher Nutzen ist der Workflow für die Erarbeitung der genomischen und transcriptomischen Profile sehr flexibel und kann für weitere Analysen der TCGA Daten verwendet werden.

4 Background

Colorectal cancer is the third most common tumor in males globally behind lung and prostate cancers, and the second most common in females globally behind breast cancer (1). 40% of all colorectal cancer patients are 75 years old or older (2). While regular screening and the associated early detection of precancerous lesions has resulted in favorable trends in some Western countries (3), other countries with historically low incidences such as Japan, the Czech Republic, and Slovakia have seen a marked increase in cases (3). This can be attributed in part to the "Westernization" of these countries, which is associated with increased prevalence of fat and meat consumption, lack of physical excercise, and smoking (4)(5)(6). Many developing countries can be expected to go through such growth phases as well. As such, the prevention and treatment of colorectal cancer plays an important role in global health.

While regular screening has reduced the mortality of colorectal cancers measurably in countries that have such programs, there are none-the-less some neoplasias which were not or could not be detected at an early enough stage for therapeutic resection. In these cases of late stage tumors, chemotherapy plays an important role. According to the 2008 S3 clinical guidelines for the treatment of colorectal cancer published by the Association of the Scientific Medical Societies in Germany, a number of drugs are commonly used in the treatment of colorectal cancer, including 5-Fluorouracil, Capecitabine (a 5-FU prodrug), Irinotecan, Oxaliplatin, Trifluridine, and tipiracil (2). In most cases these chemotherapeutics are used in combinations of two or more and are sometimes combined with targeted therapy using antibodies (2), such as anti-EGFR cetuximab or panitumumab, which are both used in the treatment of metastatic colon cancer.

Since 1990, when Bufill et al. first described differences between colorectal tumors proximal and distal to the splenic flexure (7)(in subsequent literature synonymously referred to as right-/left-sided or aboral/adoral), there has been a movement in the medical field to understand colorectal cancer as a heterogenous group of cancers rather than one distinct cancer. Although other prognostic factors have been discovered since, tumor location has remained an important parameter in clinical settings and is relied upon to be both prognostic and predictive, with right-sided tumors being generally more aggressive and having

worse overall survival (8). Cetuximab in particular is ineffective in metastasized right-sided carcinomas (8). Attempts have been made to explain this difference on a genomic level.

KRAS-gene and NRAS-gene mutation status has been shown to be a main predictor of cetuximab response (9) and BRAF-gene mutations have prognostic value for KRAS-gene wild-type tumors (10). Both KRAS-gene and BRAF-gene mutations influence the signaling pathway for growth factors as shown in Fig 1. These mutations cause this signaling pathway to be activated and remain in an *on* state independent of stimuli, thereby explaining the lack of response to anti-EGFR therapy. Unfortunately, all patients inevitably develop resistance to this treatment and about half of these cases are the result of KRAS-gene or NRAS-gene mutations (11).

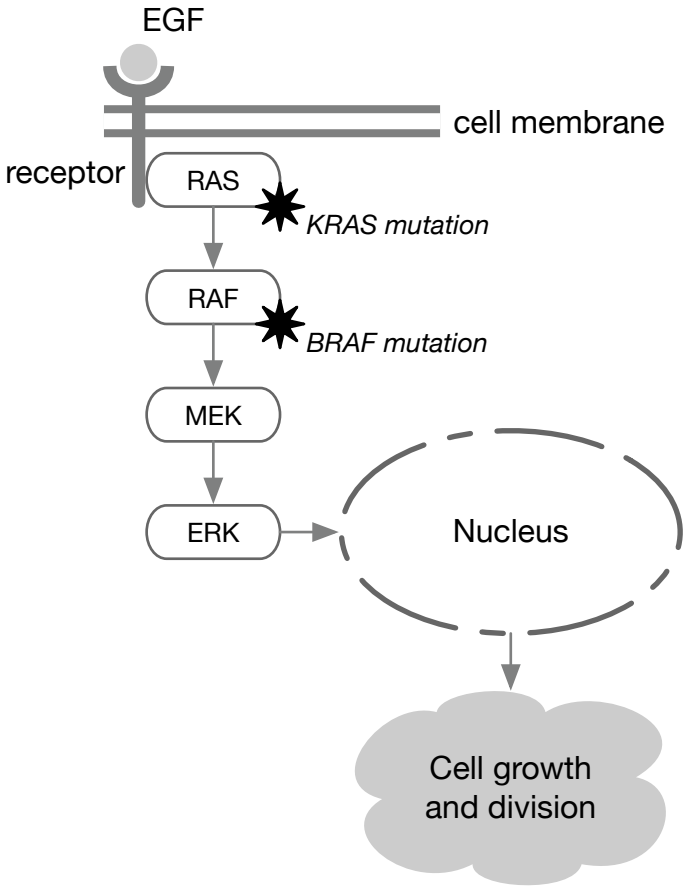

Figure 1: RAS-RAF-MEK-ERK Pathway. The black stars represent potential mutations which could result in gain of oncogenic function.

In patients who do not develop cetuximab resistance due to KRAS-gene, NRAS-gene, or BRAF-gene mutations, additional predictors have been identified. Amplification of the MET receptor has been identified as an additional driver of anti-EGFR resistance (11). MET and its cognate hepatocyte growth factor (HGF) are responsible for cell mobilization, tissue repair, and wound healing in differentiated cells and play an importand role in embryogenesis (12). The amplification of this receptor in colorectal cancer leads to anti-EGFR resistance, probably by counteracting MAPK and AKT inhibition (11). Polycomb group protein enhancer of zeste homolog 2 (EZH2) expression was recently positively correlated with overall survival in KRAS-gene wild-type CRC patients treated with anti-EGFR medications (13), suggesting a protective role in the development of resistance, and fibroblast growth factor 9 (FGF9) upregulation in KRAS-gene wild-type CRC patients has been associated with resistance to anti-EGFR therapy (14). A number of others have also attracted clinical attention, such as HERZ amplification (15).

All these factors together account for most (>80% (11)) of cetuximab-resistant colorectal carcinomas. The remaining cases hint towards further, as of yet unkown markers for EGFR resistance.

In 2015 an international consortium of experts published a classification system to distinguish between so-called consensus molecular subtypes, or CMS for short, each with unique mutation spectra and clinical progressions (16) in the hope that a more generalized, multi-parameter framework might help to give a better basis for the molecular classification. The paper lays out four major subgroups denoted CMS1 through 4, as shown in Fig. 2 (16). The paper also analysed the proportion of CMS in relation to tumor location, as seen in Fig. 3 (16)

The CMS paper further strengthened the conjecture that colorectal cancer is not one but a complex, heterogeneous group of diseases. It also showed tendencies in the tumor-biology associated with colorectal tumor localization. However, as remarked in the paper, the poor genotype-phenotype correlation in colorectal cancers make it difficult to pinpoint biomarkers beyond those previously validated (16). It also falls short of describing a molecular mechanism or signature by which anti-EGFR resistance develops in colorectal cancer and how this response is associated with tumor location. As such, left/right remains a good predictor of anti-EGFR response in RAS wild-type colorectal cancer. It

CMS1 MSI immune	CMS2 Canonical	CMS3 Metabolic	CMS4 Mesenchymal
14%	37%	13%	23%
MSI, CIMP high, hypermutation	SCNA high	Mixed MSI status, SCNA low, CIMP low	SCNA high
BRAF mutations		*KRAS* mutations	
Immune infiltration and activation	WNT and MYC activation	Metabolic deregulation	Stromal infiltration, TGF-β activation, angiogenesis
Worse survival after relapse			Worse relapse-free and overall survival

Figure 2: Description of the 4 major CMS

Figure 3: Proportion of CMS in relation to tumor location. The labels 1 through 4 denote CMS 1 through 4.

was hypothesized that by developing a better understanding of the molecular differences underpinning tumor location not only on a genomic but also on a transcriptomic and morphological level, it would be possible to overcome the poor genotype-phenotype correlation described in the CMS paper and define molecular/morphological signatures of left- and right-sided colorectal cancers. After validating these signatures, they could be used to aid in the discovery of molecular weak-points in right-sided colorectal cancers which could represent potential therapeutic targets.

5 Methods

For detailed explanations of the steps taken and complete source code, please refer to the development repository of this work, available via Gitlab (**???**), or to the print copy of the jupyter notebook at the end of this thesis.

5.1 Overview of Software

The analysis was largely done using computer algorithms written in python3 (17). Python was chosen as it conatins a rich, well-established suite of core modules, many of which were used in this work.

5.1.1 Core Modules

`os` (18) ~ facilitates use of operating system dependent tasks, such as opening and reading files, building and retrieving paths, and creating, moving and renaming files and folders.

`glob` (19) ~ facilitates searching for files and folders using Unix style pathname pattern expansion.

`csv` (20) ~ facilitates the reading and writing of data in "comma-separated value" format, the most common interchange format for tabular data. It is capable of handling arbitrary delimiters.

`shutil` (21) ~ facilitates high-level operations on files and collections of files which go beyond those implemented in the `os` module.

`collections` (22) ~ extends the default Python datatypes to include specialized data containers. In particular, this work used the `defaultdict` constructor which calls a factory function to supply missing values.

`json` (23) ~ facilitates the encoding and decoding of Python data structures into and out of JavaScript Object Notation (JSON), a data-interchange format most commonly used in REST API-based client-server communication.

`gzip` (24) ~ a wrapper for the zlib (25) implementation of the gzip algorithm, which facilitates the compression of data and is particularly well-suited to the data types used in this work.

`xml` (26) ~ facilitates the reading and writing of data stored in Extensible Markup

Language (XML).

`pickle` (27) ~ To facilitate development, a disk storage and retrieval function was implemented using the python pickle library, which allows for the serialization and deserialization of python objects to and from a binary file. This allows a user to store and retrieve the panda dataframes as well as use the final, trained classifiers to verify results or use the classification system in another context.

5.1.2 External Modules

The use of the Python programming language also made it possible to leverage a number of third-party data-retrieval, data-management, statistics, and machine-learning modules already implemented, well-documented, and (with the exception of plotly) freely available to the public. They are listed and described below.

`Requests` ~ The Requests module for Python implements HTTP/1.1 `GET`, `OPTIONS`, `HEAD`, `POST`, `PUT`, `PATCH`, and `DELETE` requests. These are commonly used when communicating with servers over the internet via exposed REST APIs.

`iPython` (28) and `Jupyter` (29) ~ iPython is a command shell for interactive computing using the python programming language. It implements a number of features that go beyond the standard python interpreter, which facilitate rapid prototyping and development. The programming-agnostic utilities have since been moved to a separate project called "Jupyter". Jupyter is a web application which functions as a wrapper for iPython, but can be similarly used for many other programming languages, provided that the language implementation offers some form of a Read–Evaluate–Print Loop (REPL) environment (29).

`NumPy` (30) ~ NumPy is a python module which adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays (31). It is the basis upon which most of the other SciPy packages are built. Its implementation of N-dimensional data arrays (`ndarray`) in particular is useful when dealing with large quantities of data, as was the case with this work.

`Pandas` (32) ~ Pandas is a module for python which extends the functionality of Python + NumPy to include data manipulation and analyses for numerical and time series data. While most operations using pandas can be executed using NumPy alone, its two fundamental data structures, `Series` (1-dimensional)

and `DataFrame` (2-dimensional) greatly simplify tasks dealing with labeled or relational data with heterogeneous data types, as was the case in this work.

`scikit-learn` (33) ~ `scikit-learn` is the defacto standard machine-learning module for Python. It features efficient, flexible, and reusable data analysis tools for python with an emphasis on machine learning algorithms, such as the logistic regression and random forests classifier used in this work, which are described in detail below.

`t-SNE` (34) ~ Initial exploration was done using t-distributed stochastic neighbor embedding (t-SNE), implemented using the `scikit-learn` module. Using this probabilistic (non-deterministic) model, it is possible to explore the ultra-high-dimensional dataset in 3 spacial dimensions, thereby facilitating fast and effective data-exploration, which can lend important first insights when approaching an unknown set of data.

`matplotlib` (35) ~ `matplotlib` is the defacto standard 2D plotting library for Python. It provides a MATLAB-style interface and integrates well into other SciPy libraries such as iPython and Jupyter Notebook. While it supplies full flexibility for creating publication quality graphs, this proved more of a burden while rapidly integrating workflow changes, which is why most of the graphing was done using the plotly library (see below).

`plotly` (36) ~ `plotly` is a non-free javascript library for creating browser-based graphs in Jupyter with it's associated python API, plotly.py. It features a simple interface which is not based on MATLAB as is the case with matplotlib, and generates interactive plots for quick, intuitive data analysis.

`Anaconda` (37) ~ Anaconda is a package manager using development environments, which allows for the management of dependencies on a project-by-project basis. Anaconda, in comparison to other environment managers, not only manages the Python module dependencies, but also the Python executable itself, which allows for the existance of multiple Python versions on a system in parallel. This enables any user to install the analysis workflow created in this work without disturbing any existing software on the system.

5.1.2.1 Classifiers

Since logistic regression and random forest classification constitute a key part of the analysis in this thesis, they are explained below in further detail.

5.1.2.1.1 Logistic Regression

First described by David Cox in 1958 (38), logistic regression can be used to predict the probability of an outcome using an arbitrary number of predictors. While it is not technically a classifier, it can be used to construct one by defining a cutoff-value for the probability outcome. In the binary classification case this trivially implemented using a simple majority vote.

The implementation in this work is a nested cross-validated logistic regression, with 10 outer folds and 12 inner folds, based on the `LogisticRegressionCV()` function in `scikit-learn`. Cross-validation is done by splitting the dataset into k subsets (in this case stratified to ensure an even distribution of left and right samples) and then iteratively taking one of these subsets out for testing. A portion of this process is shown in Fig. 4. The results of all iterations are then combined, forming a mean quality of the regression over the entire dataset.

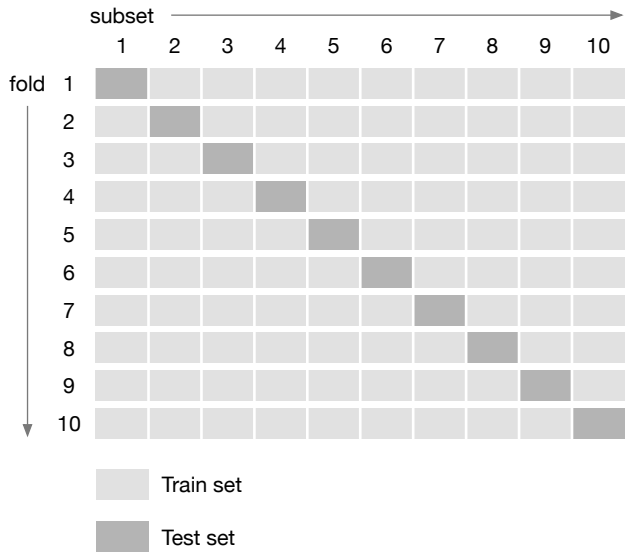

Figure 4: k-fold cross validation for k=10. Each row represents the complete dataset, split into 10 subsets, represented by the columns. For each fold, represented by the rows, a different subset of the data is selected as the test set. This process is repeated until all subsets have been used in both the training and testing case.

The choice of k is somewhat arbitrary. While choosing a larger k is less likely to overestimate the true expected error, this results in smaller testing sets, which

in turn leads to greater variability in the results, and a near-linear increase in computational costs. The limit case is so-called "leave-one-out cross validation", where $k = n - 1$.

The nested approach additionally separates the training set of each iteration into further subsets, enabling the tuning of hyperparameters of the model without "leaking" information into the results, at the expense of computational effort.

5.1.2.1.2 Random Forest

In general, random forest classification builds upon the concept of decision trees. The random forest algorithm, an extension of Tin Kam Ho's "random decision forests" (39) using Leo Breiman's concept of bootstrap aggregation (40), is an ensemble-type machine learning method, which operates by constructing a large number of decision trees with random subsets of the feature space, training each tree individually, and making a majority vote with all decisions. Random forests are robust classifiers which avoid the tendency for overfitting of standard decision tree classifiers, at the expense of computational complexity and some loss of direct interpretability. Importantly in contrast to many other classification methods, the classification error ϵ of a random forest algorithm C_{RF} is proven to converge to a constant c as the number of observations n approaches infinity (41), or

$$\lim_{n \to \infty} \epsilon(C_{RF}) = c.$$

This, in combination with its resistance to over-fitting due to its random nature (40) allows it to be used in proofs and the validity of the model can be assessed using a receiver operating characteristic (ROC) curve as is common in the life sciences. Interestingly, the rate of convergence is independent of the dimensionality of the data d and depends only on the number of "strong features", which explains how it is possible for RF classifiers to readily handle data where $d \gg n$ (42). A general explanation of random forests follows by way of example.

Consider the dataset X, where f_{a1} is the a^{th} feature of sample 1, etc.

$$X = \begin{bmatrix} f_{a1} & f_{b1} & f_{c1} & \cdots & f_{m1} & C_1 \\ f_{a2} & f_{b2} & f_{c2} & \cdots & f_{m2} & C_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ f_{an} & f_{bn} & f_{cn} & \cdots & f_{mn} & C_n \end{bmatrix}$$

In the first step, called "bootstrap aggregation" or "bagging", a subset

$$X_S \subseteq X$$

of *samples* is taken at random for each of n trees.

$$X_S = \begin{bmatrix} f_{a3} & f_{b3} & f_{c3} & \dots & f_{m3} & C_3 \\ f_{a7} & f_{b7} & f_{c7} & \dots & f_{m7} & C_7 \\ f_{a9} & f_{b9} & f_{c9} & \dots & f_{m9} & C_9 \end{bmatrix} \subseteq X.$$

Then, for each node in the decision tree, a subset

$$X_{Sf} \subseteq X_S$$

of *features* is chosen at random.

$$X_{Sf} = \begin{bmatrix} f_{b3} & f_{c3} & f_{g3} & C_3 \\ f_{b7} & f_{c7} & f_{g7} & C_7 \\ f_{b9} & f_{c9} & f_{g9} & C_9 \end{bmatrix} \subseteq X_S$$

Finally the optimal feature (based on gini purity) is chosen for the split as shown in Fig. 5.

This process is continued iteratively on each node until the tree is "pure", meaning that each leaf contains samples of only one class. Due to this iterative process, the random forest classifier is capable of recovering information about compounding effects, where the effect E_{ab} of two simultaneous variants a and b is greater than $E_a + E_b$. This differs from logistic regression, where the coefficient of an independent variable is tuned by the effect of that variable only.

The choice of forest size, similar to the choice of k for cross-validation, is largely up to the user (43). However, it can be said that generally, the larger the forest, the more stable the result.

The implementation in this work uses the `RandomForestClassifier()` function from `scikit-learn`.

5.1.3 TCGA Database Tools

A number of tools specific to the TCGA Dataset were used in this work. These facilitate access to the data using web-based interfaces. They are described in detail below.

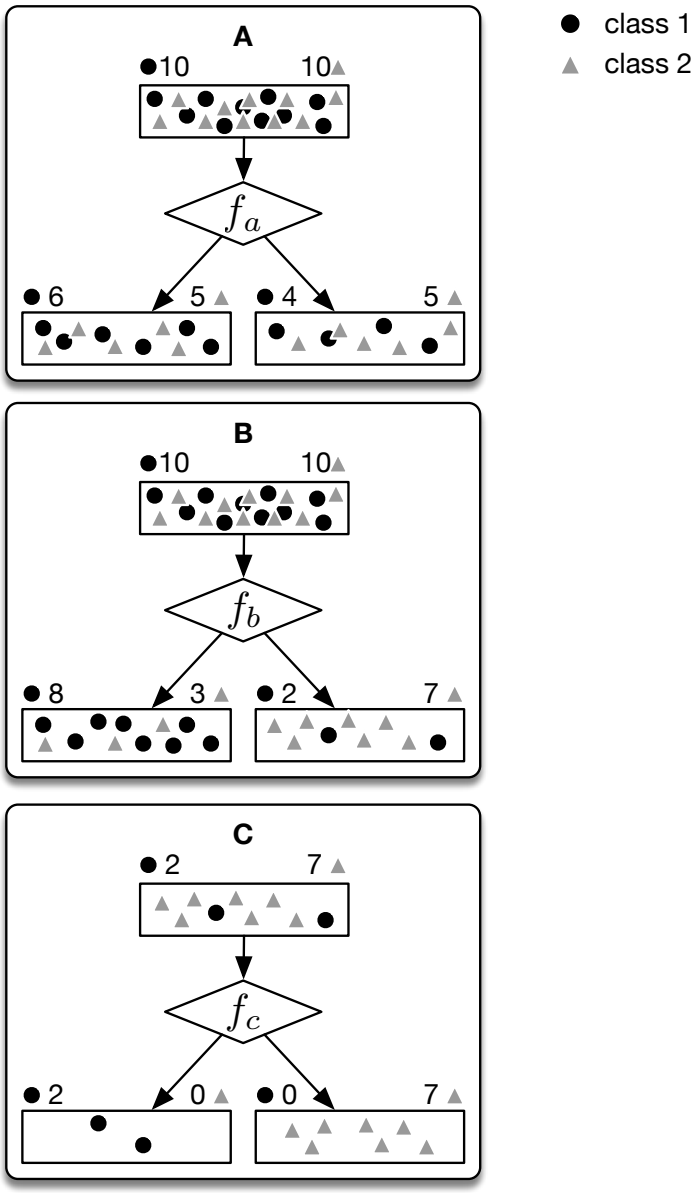

Figure 5: **A:** An example node, starting with a branch containing 10 samples of each class. The node results in a non-optimal split using feature a. **B:** Using the same starting branch as shown in **A**, the node now splits the samples using feature b, which results in the optimal separation amongst possible features in the set $\{a,\ b\}$. **C:** Continuing with the second (right side) branch from **B**, another node is created using feature c, which results in leaves containing samples of only one class. This branch is now considered "pure".

`GDC Data Portal` (44) ~ The GDC Data Portal is the main webpage for accessing TCGA databases. It features powerful search functionality and supplies some rudimentary analysis tools. The most useful feature in the context of this work was the "repository", where data can be filtered and selected ("checked-out") for download. This yields a so-called "manifest" file which can then be passed through the `gdc-client`.

`gdc-client` (45) ~ The `gdc-client` is a command line utility for downloading data files from the TCGA database. It accepts manifest files generated via the GDC Data Portal and downloads the files listed therein serially. It yields not only clinical data files, but also transmission logs and annotation information.

`GDC REST API` ~ The GDC Representational state transfer (REST) Application Programming Interface (API) represents the back-end functionality of the GDC Data Portal. All information served via the Data Portal is available via the API and vise versa. Use of the API is necessary when dealing with large amounts of data in a programmatic fashion. For this work it was particularly useful for retrieving metadata about the files downloaded via the `gdc-client`.

5.1.4 TCGA Data Format

The TCGA Dataset (46) (which has since been integrated into the **G**enomic **D**ata **C**ommons) is organized into top-level "projects" corresponding to primary tumor sites, i.e. kidney, lung, ovary, etc. These projects contain some number of cases, each with raw sequencing data, transcriptome profiling, single nucleotide variation, copy number variation, methylation information, and biospecimen supplement. The data model used by the GDC is explained in detail on the GDC Data Model Components page (47). Legacy versions of the dataset contain additional data, such as digital tumor specimen slides (44), which can be used to assess tumor morphology. Each file also possesses metadata, which contains association information between files. Finally, there are some project-level aggregate files including the somatic Mutation Annotation Format files, which is where collected genomics data is stored (44). These files are created using a pipeline described here: https://docs.gdc.cancer.gov/Data/File_Formats/MAF_Format/

5.1.4.1 Clinical Data File Format

Clinical data associated with cases in the TCGA is supplied in XML format, with each file being approximately 30 KiB in size and containing approximately 130 fields. Each file belongs to a specific case in the dataset. These files optionally contain information on tumor location, which was essential for this work. The available anatomic location information was available as one of eight classes: `Cecum`, `Ascending Colon`, `Hepatic Flexure`, `Transverse Colon`, `Splenic Flexure`, `Descending Colon`, `Sigmoid Colon`, and `Rectosigmoid Junction` as shown in Fig. 6.

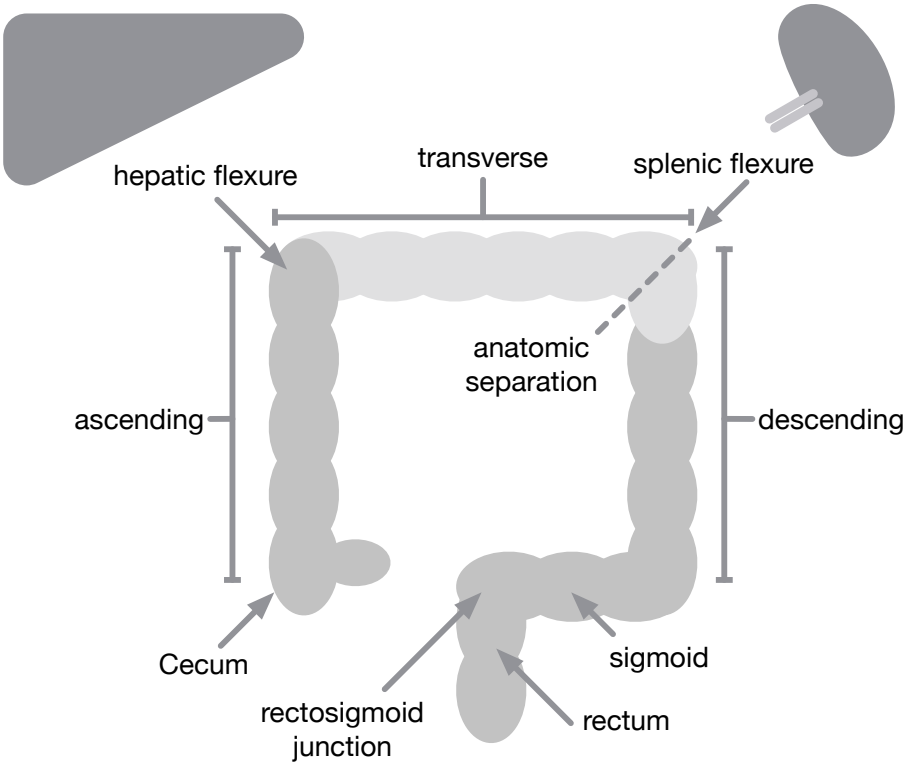

Figure 6: Visualisation of the anatomic location information available via the TCGA. The transverse colon and splenic flexure were not included in the analysis. This is denoted by the lighter shade of gray.

5.1.4.2 Genomic Data Format

The genomic data in the TCGA database is available in multiple formats, the result of three separate so-called "variant calling pipelines" using the variant callers `MuSE`, `VarScan` and `SomaticSniper`, or `MuTect`. The detection of variants in a tumor tissue sample is a complicated process and there is no consensus on

the best variant calling algorithm (48).

`Somatic Sniper` has been largely superceded by next-generation variant callers due to its high number of false-positive calls (49), and `VarScan` displays low agreement with other variant callers (49).

`MuTect` is a variant calling algorithm developed by the Broad Institute which has a well-established status as being highly-sensitive and highly-specific (50). It is particularly good at detecting point substitutions in samples with low frequency of mutations, which occur when there is great heterogeneity in the tumor sample or the sample contains a measurable amount of normal cells. Moreover, it is amongst the recommended variant callers when compared with others (49). Since colorectal cancer is heterogeneous in nature and since it was neither possible to control the tumor sample selection methods nor the preparation of the DNA and thereby the relative amount of tumor cell DNA, the `MuTect` was chosen for the analysis. Ideally, an intelligent ensemble method utilizing a combination of callers would be ideal, however this was beyond the scope of this thesis.

5.1.4.3 Transcriptome Data Format

The transcriptome is the set of the RNA molecules in cells or a tissue. In a narrower sense, "transcriptome" can refer specifically to mRNA. A number of aspects have to be taken into account when dealing with trascriptomic data.

For one, the transcriptomic data is continuous, indicated by its representation using floating point (decimal) values. Secondly, when generating expression data using next-generation sequencing techniques, the length of the mRNA influences the expected number of reads. As such, a higher number of reads may be due either to a higher expression or simply a longer transcript. Thirdly, the mRNA in cells are not always complete, but rather exist in fragments in various stages of degradation, the rate of which is dependent on further extraneous factors.

It follows that the *absolute* amount of a transcript cannot be used to directly infer expression levels. Instead, relative expression levels are generated by normalizing the read counts to those of reference genes which are commonly referred to as "housekeeping genes". There are a number of ways to achieve this, but the TCGA uses either `FPKM` (**F**ragments **P**er **K**ilobase of transcript per **M**illion mapped reads) (equation 1) or `FPKM-UQ` (**FPKM**-**U**pper **Q**uartile) (equation 2).

$$FPKM = \frac{RC_g * 10^9}{RC_{g\Omega} * L} \tag{1}$$

$$FPKM - UQ = \frac{RC_g * 10^9}{RC_{g75} * L} \tag{2}$$

- RC_g: Number of reads mapped to the gene
- $RC_{g\Omega}$: Number of reads mapped to all protein-coding genes
- RC_{g75}: The 75th percentile read count value for genes in the sample
- L: Length of the gene in base pairs, calculated as the sum of all exons in a gene

In both cases, the 10^9 reads per gene are normalized to all reads multiplied by the length of the gene in question. Normalizing by the length of the gene ensures that large genes do not erroneously lead to high measured expression levels and vice versa. Then, in FPKM, the read counts are normalized over all reads mapped to all protein-coding genes, whereas in FPKM-UQ, the read counts are normalized over the upper quartile, that is, the upper quarter of read counts for the given batch of genes (51). This makes it easier to compare values generated using different experimental conditions. It is worth noting that this normalization procedure depends on the specific batch of genes it was normalized against. This means that in order for the scaled values to be useful, they have to be analyzed in the context of the entire gene set.

6 Results

A large portion of the work involved modifying the format of the incoming data from the TCGA dataset for further processing and final analysis.

Care was taken not to modify the original data in-place to ensure that the workflow would be reproducible. In order to maintain consistency in naming conventions, files and cases —once read— are referred to strictly by their universally unique identifier (UUID)

6.1 Retrieval

A large number of files from the TCGA portal were required in order to conduct the analyses, including clinical data, mutation data, and normalized transcriptome data. The GDC uses a relational database to represent the data, however the client-side implementation is file-system-based. This greatly reduced upfront implementation complexity and allowed for the utilization of native python functions and iteration methods to build the data frames.

The files were selected from the GDC Data Portal, then a manifest file was downloaded via the browser, and finally the manifest file was fed into the `gdc-client` program which conducted the download of the actual files. The download via the `gdc-client` yielded transmission logs and annotation information in addition to the actual data.

The transmission logs are files containing information about the transfer of data from the server to the local machine to control the quality of the data. In this case the data transmission concluded without errors so that the veracity of the download was assumed. However, the transmission logs needed to be removed before further analysis could be conducted. Removal was done by an iterating procedure over the file hierarchy in search of files matching a `regular expression`. Regular expressions (also known as *regex*) in formal language theory are search patterns for use in string searching algorithms and are ubiquitously implemented in word processors.

The relationships between the files and their corresponding cases are necessary for associating different files with one another. However, this information is stored in separate server-side metadata and so had to be queried separately using the GDC REST API. This was done by iterating through the manifest file, extracting

the UUID for each downloaded file, querying the REST API for that UUID using the python `request` library, and dumping the resulting `json` data to a file on disk. Then the metadata was searched for the `case_id` and this association was stored in a dictionary, with each file having an entry containing its associated case in the TCGA-COAD project. After this database was constructed, the association of the matrix was reversed, creating a one-to-many association of cases and their respective files.

6.2 Filtering

The TCGA COAD Project contains 459 cases. Having retrieved the data (n=459, 100%), the parameters needed for the analysis had to be extracted and filtered. Processing the data to get it into a state in which it can be analyzed is sometimes called "data munging". This takes up a large portion of the work and each step requires careful planning to avoid both introducing unwanted complexity and reducing the data set more than necessary.

6.2.1 Annotations

The downloaded files were searched for annotations, which is how possible confounding features such as preceding or additional cancers are noted in the database (44). Though using this information is theoretically possible, the decision was made to ignore these cases, 69 in total, as correcting for the confounders on a case-by-case basis would have immensely increased the complexity of the analysis (n=390, 85%).

6.2.2 Localization

Since right-sided CRC tend not to respond to anti-EGFR therapy despite KRAS and NRAS wild-type configuration, it was hypothesized that the discovery of genomic markers which predict left- and right-sidedness could also help to mechanistically explain the difference in anti-EGFR response. As such, the goal was to compare only two classes, "left" and "right" tumors, so these groups had to first be rigorously defined. This was done using the clinical data from the TCGA. The tumor location information was extracted from each of the 459 clinical data files. 16 cases had no such information and so had to be removed from the analysis. This was done by iterating over the clinical data files of each case in the dataset and comparing the available tumor location data with the

sets defined below. The 16 cases containing files containing no such data where flagged for removal and removed en bloc at the end of the iteration (n=374, 82%).

The anatomic location information was available from the GDC as a set of 8 classes as shown in Fig. 6.

The 23 `Transverse Colon` and `Splenic Flexure` cases where also removed from the analysis (n=351, 77%). It was suspected that this would achieve a better separation of the two groups. This point was chosen because the Arc of Riolan, an arterio-arterial anastomosis between the superior and inferior mesenterial arteries, and the Cannon-Böhm point, which marks the transition from vagal to sacral innervation, are both found in this section of the colon, and as such it represents the natural junction between mid- and hindgut (52). The final grouping was:

Right Colon ~ `Cecum`, `Ascending Colon`, `Hepatic Flexure`

Left Colon ~ `Descending Colon`, `Sigmoid Colon`, `Rectosigmoid Junction`

Now after having clearly separated groups of cases representing left and right, the next requirement was to retrieve the genomic and transcriptomic data and then to find a set of cases for which all data was available. This amounts to a definition and intersection of three sets: the clinical, genomic, and transcriptomic data. This was done serially in three major steps, as shown in Fig. 7 and described below.

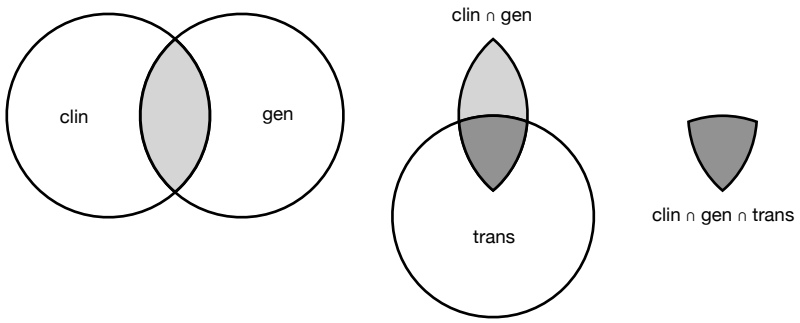

Figure 7: Diagram showing the set operations performed on the data.

6.2.3 Genomic Data

The mutation data of the TCGA is stored in mutation annotation format (MAF) files. These files contain all mutations for all cases of a project in a long list of variant calls. This list was first filtered to remove information which was irrelevant to the analysis.

6.2.3.1 Missing Data

Ideally, TCGA MAF files contain variant information for all cases in a project. However, 42 of the cases for which clinical data was available had no entry in the MAF file. Thus, these cases had to be removed (n=293, 64%). Conversely, 64 cases referenced in the MAF file had no associated clinical data and were also removed, though this did not impact the total number of cases. After filtering out this incomplete data, the remaining cases were comprised of both clinical and variant data.

6.2.3.2 KRAS and NRAS Mutations

Since KRAS and NRAS mutations have already been established as highly correlated with anti-EGFR resistance (9), the cases where these mutations were present were removed. This was done by first generating a list of the associated `case_id's` of variants with the `Hugo_Symbol` "KRAS" and "NRAS". Then, all variant calls with a `case_id` matching one of the `case_id's` in the previously generated list where removed from the matrix. This reduced n by another 130 cases (n=163, 36%).

6.2.3.3 Synonymous Variants

Synonymous variants are those variants which do not change the amino acid in the final protein. For example the triplets `CGA` and `CGC` differ in their final base (A vs. C), yet they both encode the same amino acid (Arginin). This is due to the redundancy of the genetic code. Under the assumption that known synonymous variants have no discernible effect on the pathogenesis of tumors and are most likely polymorphisms or artifacts of hyper-mutated tumor genomes —which needed to be controlled for— these 21,751 variants were removed from the feature set. This was done by using the `Consequence` column in the MAF file. All calls where the `Consequence` value was equal to `synonymous_variant`

where removed. This did not reduce the number of cases but was an important feature reduction step.

6.2.3.4 Transposition of the Matrix

Finally, the mutation information had to be associated with each corresponding case. Since the MAF file does not offer this information directly, as is the case with the clinical data, where each case has an associated clinical data file, it was necessary to transpose the matrix so that each row corresponds to a particular case and each column denotes a gene with a potential mutation. This is shown by example in Fig. 8.

Hugo_Symbol	Entrez_Gene_Id	Start_Position	End_Position	case_id	...
gene1	...	...	...	7	...
gene2	...	...	...	9	...
gene3	...	...	...	3	...
gene4	...	...	...	7	...
gene5	...	...	...	6	...
gene6	...	...	...	8	...
gene7	...	...	...	7	...
gene8	...	...	...	4	...
gene9	...	...	...	4	...

case_id	gene1	gene2	gene3	gene4	gene5	gene6	gene7	gene8	gene9
1	0	0	1	0	0	0	1	0	0
2	0	0	1	0	0	1	0	1	1
3	1	1	1	0	1	0	0	0	0
4	1	1	1	1	1	0	1	1	1
5	1	0	1	1	0	0	0	1	1
6	0	1	0	1	0	0	0	1	0
7	0	1	1	0	0	1	0	0	0
8	1	1	0	0	0	0	0	0	1
9	0	1	0	0	0	1	0	1	0

Figure 8: Transposing the database matrix over the value `case_id` retaining only mutation status of `Hugo_Symbol`. "1" and "0" in the bottom table represent the binary mutation status, where a "1" denotes the existence of a particular mutation (column) in a given case (row) and a "0" denotes its absence.

6.2.3.5 Hypermutation

Looking at the variant count distribution (in Fig. 9), it can be seen that the number of variants per sample differs greatly between left and right tumors, with a large number of right-sided samples having more than 250 variants. This hyper-mutated state is described as CMS1 (16) and is highly correlated with right-sidedness in the CMS classification. Training the random forest classifier on this data yielded a ROC curve (Fig. 11), which denotes a bias in the classifier. This means that the classifier relies on the discrepant variant counts instead of the variants themselves for its prediction. Since the variants in the MSI (microsatellite instability) cases are unlikely to be mechanistically involved in anti-EGFR resistance but are nonetheless asymmetrically represented in the classes, the predictive power (in this case for right-sidedness) of these variants would be unrealistically high. Additionally, specific therapy is available for MSI high CRC via anti-PDL1 (programmed death ligand 1) chemotherapy (53). Therefore, microsatellite-instable cases had to be removed from the analysis if novel markers where to be discovered. A variant count of 250 was used as a cutoff to exclude cases with high frequency of mutations, as this was an easy-to-implement surrogate for MSI status. This resulted in an additional 40 samples with higher variant counts which were removed from the final analysis (Fig. 10) (n=123, 27%).

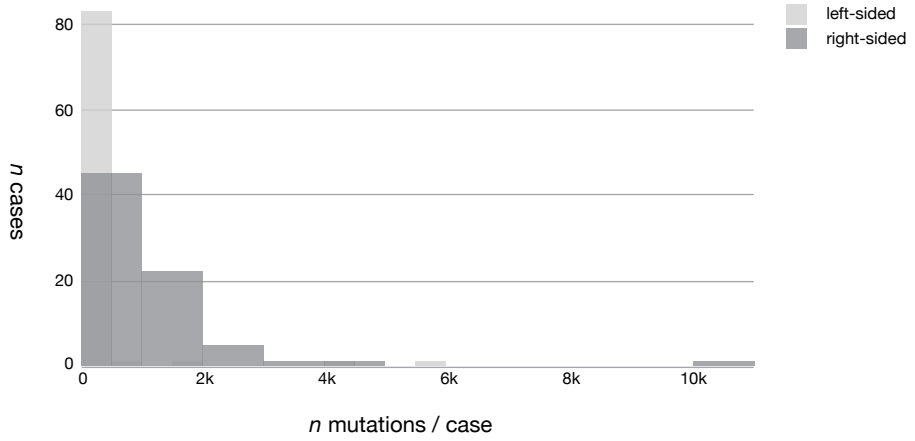

Figure 9: Native Variant Frequency Distribution. In the set of right-sided tumors, there exist some cases with an extreme number of mutations, represented by data points on the far right of the graph.

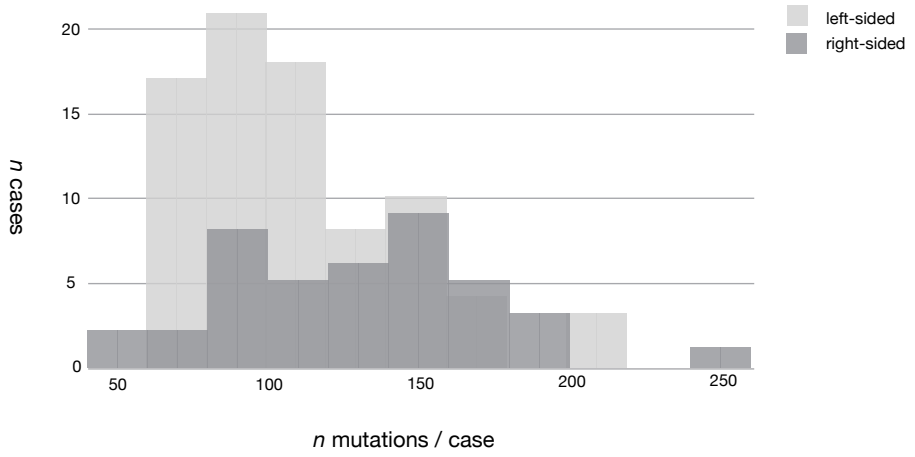

Figure 10: Corrected Variant Frequency Distribution. After removing the extreme cases, the two groups are more similar, thereby largely correcting the selection bias.

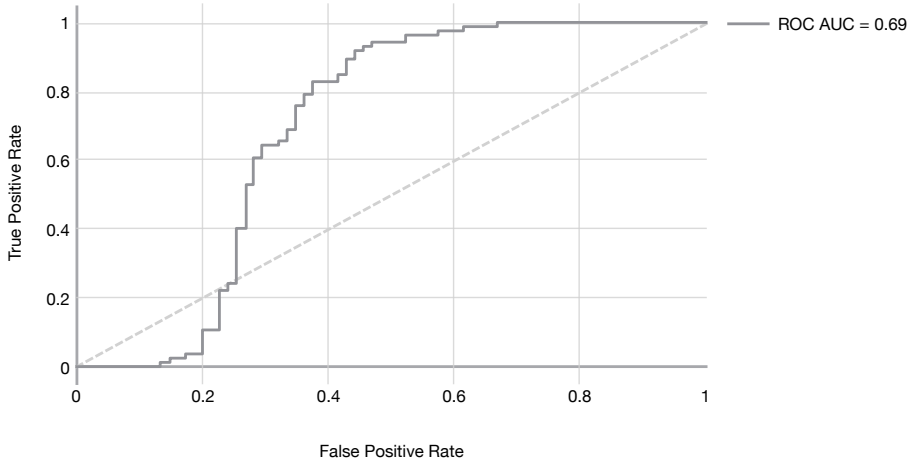

Figure 11: The sigmoid ROC due to discrepant mutation distributions in the classification groups.

6.2.4 Transcriptomic Data

The initial consideration of combining genomic and transcriptomic data to train a single set of classifiers would have made analysis practically impossible due to the differing dimensionality of the data. This is due to the fact that the classifying algorithms predict a dependent variable (in our case tumor location) per unit change of an independent variable. In order to compare the feature "importance" over the combined feature space, either the boolean mutation values would

need to be normalized over the range of possible transcriptome values, thereby artificially inflating the dimensionality and reducing the predictive power per unit change by at least three orders of magnitude, or the range of the transcriptome values would have had to be reduced to boolean, thereby artificially reducing the entropy of the data. Both of these choices would have required a much greater number of cases, which were unavailable. As neither of these possibilities were feasible, the alternative chosen was to train separate classifiers on the two datasets.

FPKM-UQ data was chosen as the format for the transcriptomic data as it facilitates the comparison of the generated values with other results which may have been generated using different experimental conditions. The transcriptome data in the TCGA database is organized analogously to the mutation annotations in that the relationship is inverse to that of the clinical data. As such, the association in the data had to be transposed in the same way as the mutation data.

After removing the cases unsuitable for analysis, 27% of the original dataset was left, a total of 123 cases.

6.3 Analysis

The reduced dimension visualization using t-SNE showed some potential –albeit modest– clustering of left- and right-sided tumors(54)[1].

For the final analysis, two binary classification techniques were chosen, cross-validated logistic regression and random forest. Logistic regression is widely-used and well-understood and serves as a baseline result for better comparison with other research, while Random Forest is a relatively new ensemble classification technique and potentially yields better results. Both models where implemented such that it was possible to extract the n most important features (in terms of coefficient values and gini importance for logit and rf, respectively[2]) and n "most sided" (where the calculated probability for the class was highest amongst all cases) which was the means of selection for potential further histomorphology.

The classification target was tumor location, which was encoded as a boolean `tumor left`. In other words, the algorithms classified cases as being left (`true`)

[1] To see this interactive analysis, please refer to the source code of this thesis (Section 9.8.2)
[2] for definitions of these terms, please refer to the respective section in the methods

or not-left (`false`).

6.3.1 Logistic Regression

The logistic regression was tuned using nested cross-validation. The benefit of cross-validation is its use of the entire dataset for training, thereby maximizing the available data, while being resilient against over-fitting. The choice of 12 for k enabled an optimal use of hardware, as it was exactly twice the number of processor cores available (which was 6). This allowed for parallel computation of 6 inner folds at a time, which was repeated twice for each of the default 10 outer folds (55), resulting in 120 unique iterations. If the choice of k had not been a multiple of the processor cores then the cores would not have been able to paralellize the load efficiently. For example, had 13 inner folds been chosen, then the computer would have run 6 parallel threads twice, followed by a single thread, thereby increasing the computation time by 50% for a single additional inner fold.

The variance between the folds[3] was used to calculate a standard deviation of the modal over the input data, allowing for an evaluation of the reliability of the results.

6.3.2 Random Forest

A liberal one fourth of total features (20,000) was chosen as the number of trees for the random forest classifier (this was the recommendation as discussed in a consultation with a statistics institute in Munich). The size of X_{Sf}[4] was set to $\sqrt{features}$, a commonly chosen value to encourage tree variability, and the default value for `RandomForestClassifier()` (56).

A measure of variance in the random forest model similar to the one implemented in the logistic regression model was taken into consideration but ultimately not implemented, as this would have greatly increased the computation required. The results from the random forest classifier could be sufficiently interpreted by comparing them with the results from the logistic classifier.

[3] refer to the methods section for an explanation of cross-validation "folds"

[4] the subset X of features f of a subset of samples S. For a more detailed explanation, please refer to the methods.

7 Analysis Results

It was hypothesized that it would be possible to generate molecular (genomic or transcriptomic) predictors of tumor location by training classifiers, using the classification target `tumor left` versus `non tumor left` (=right). As analysis was done on two datasets using two algorithms, a total of four results was yielded. The ROC curves of all classifications are shown in Fig. 12.

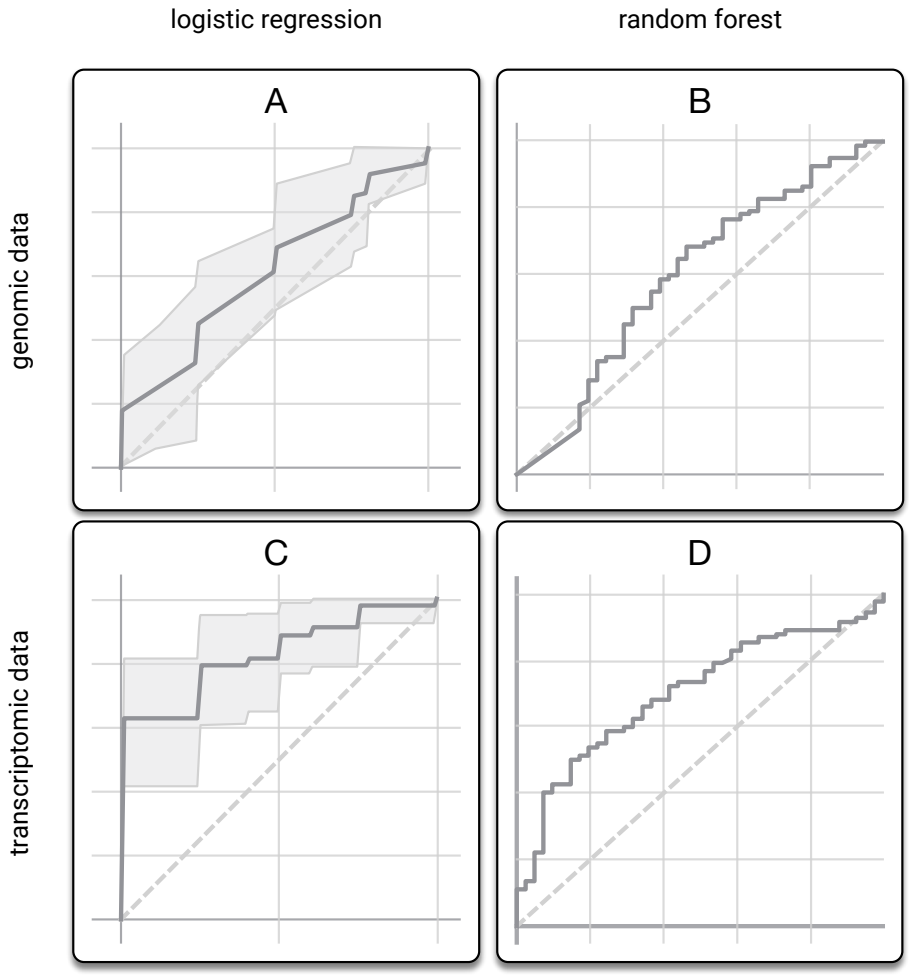

Figure 12: Overview of the receiver operating characteristics of both classifiers on both data sets. **A**: Logistic regression on genomic data. The line of non-discrimination is included within one standard deviation of the result. (AUC = 0.61 ± 0.11). **B**: Random forest classification on genomic data (AUC = 0.6). **C**: Logistic regression on transcriptomic data (AUC = 0.82 ± 0.1). **D**: Random forest classification on transcriptomic data (AUC = 0.7).

Since the logit regression estimates correlate with the dependent variable (left-sidedness), the coefficients of the logit regression are positive when the variable correlates positively with left-sidedness, and negative when positively correlated with right-sidedness. The random forest classifier does not yield coefficients but rather `gini importance`, which is a measure of classification power irrespective of outcome. Therefore the results of the random forest classifiers are all positive.

The functions of the genes found are discussed in further detail in the Discussion.

7.1 Genomic Data

7.1.1 Logit

The logit analysis of the genomic data yielded a ROC (receiver operating characteristic) with an AUC (area under the curve) of 0.61 $\pm$ 0.11 (Fig. 12), a non-significant result as it contains 0.5. The 20 genes with the largest coefficients (10 largest left and 10 largest right) show that the classification power was mostly influenced by right-sidedness-predicting variables (Fig. 13).

The genes of the variants (57) of the 10 largest coefficients in descending order were

- MEIS3 - Meis Homeobox 3
- APC - Adenomatous polyposis coli
- SSH2 - Slingshot Protein Phosphatase 2
- AMER1 - APC Membrane Recruitment Protein 1
- PLCZ1 - Phospholipase C Zeta 1
- BRAF - B-Raf proto-oncogene, serine/threonine kinase
- RYR3 - ryanodine receptor 3
- SMAD4 - Mothers against decapentaplegic
- PCDHGA7 - Protocadherin Gamma Subfamily A, 7
- OR52L1 - Olfactory Receptor Family 52 Subfamily L Member 1

7.1.2 Random Forest

The results from the logic analysis were corroborated in the random forest analysis. The random forest analysis yields an ROC without a measure of accuracy (while possible, the implementation is considerably more complex and so was decided to be beyond the scope of this work). However, considering the

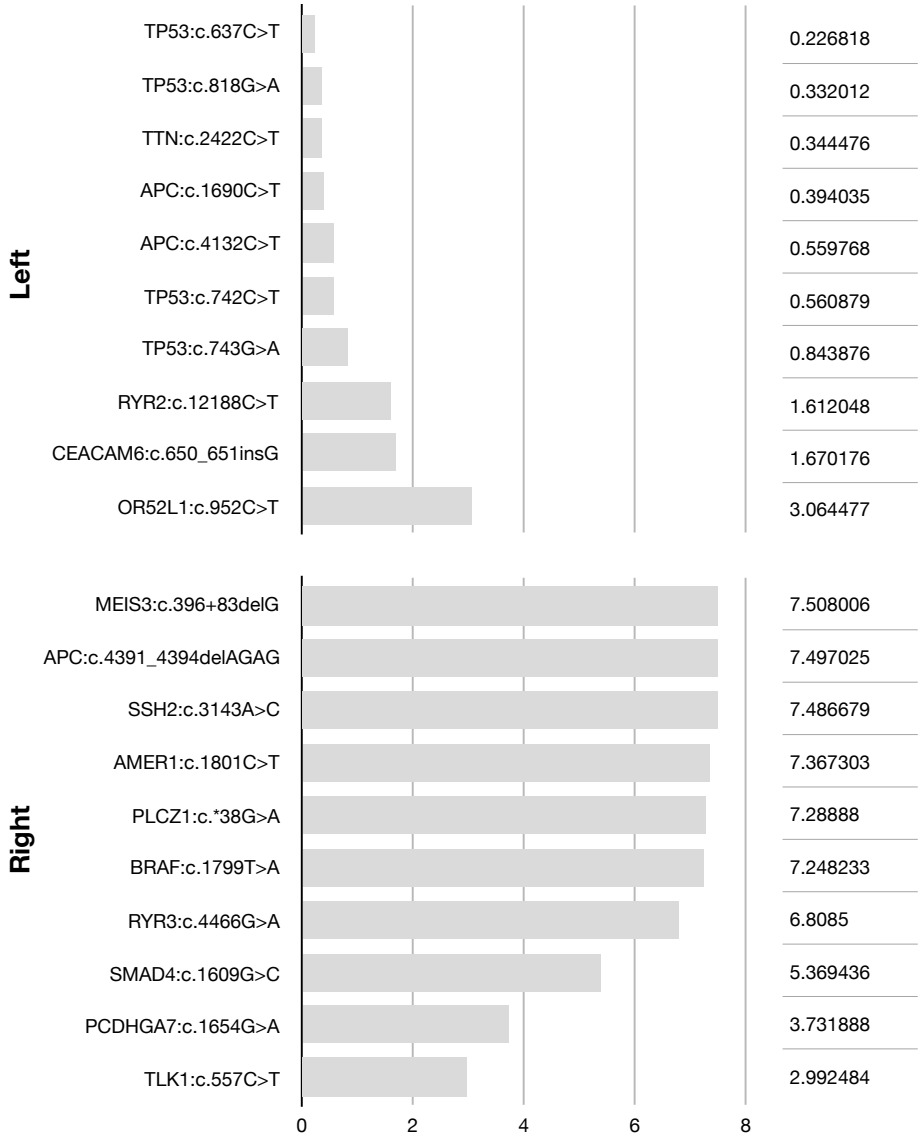

Figure 13: The 20 genomic variants with largest regression coefficients (10 largest left and 10 largest right) as identified by the logit classifier.

small AUC of 0.6 (Fig. 12), which is in consensus with the logit analysis, it can be assumed that the results of this analysis (Fig. 14) are similarly non-significant.

It is reassuring however that there is a large overlap in best predictors (7 of 10). AMER1, APC, BRAF, PLCZ1, BRAF, PCDHGA7, and RYR3 are all represented in the most important features for the random forest and logit classifiers.

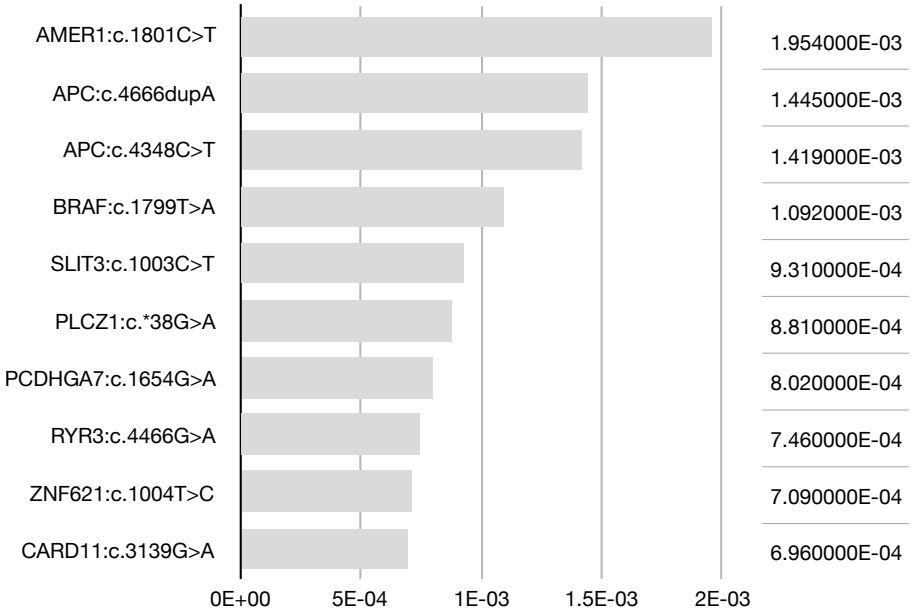

Figure 14: 10 most important features of RF classifier trained on genomic data.

In light of the significance, a further interpretation of the results is ineffectual. The results from the transcriptomic data is discussed in much further detail below.

7.2 Transcriptomic Data

7.2.1 Logit

The logit analysis of the transcriptomic data yielded a ROC with an AUC of 0.82 $\pm$ 0.1 (Fig. 12). Unlike the genomics results, the transcriptomics logit classifier used more left-sidedness predictors for its classification.

Transcript	Gene	Description (57)(58)	Predicts
ENSG00000159182.4	PRAC1	PRAC1 small nuclear protein	left
ENSG00000130303.11	BST2	bone marrow stromal cell antigen 2	right
ENSG00000100979.13	PLTP	phospholipid transfer protein	left
ENSG00000115414.17	FN1	fibronectin 1	left
ENSG00000179914.4	ITLN1	intelectin 1	left
ENSG00000109321.9	AREG	amphiregulin	left
ENSG00000189060.5	H1F0	H1 histone family member 0	left

The eighth and further coefficients have absolute values $< 0.5\mu$ and taper off very slowly, strongly suggesting that these coefficients are not significantly discernible from statistical noise.

The transcript ENSG00000159182.4, corresponding to the gene PRAC1 (PRAC1 small nuclear protein), is highly predictive of left-sidedness, having a coefficient larger than the next biggest coefficient by a factor of over 3.5 (Fig. 15).

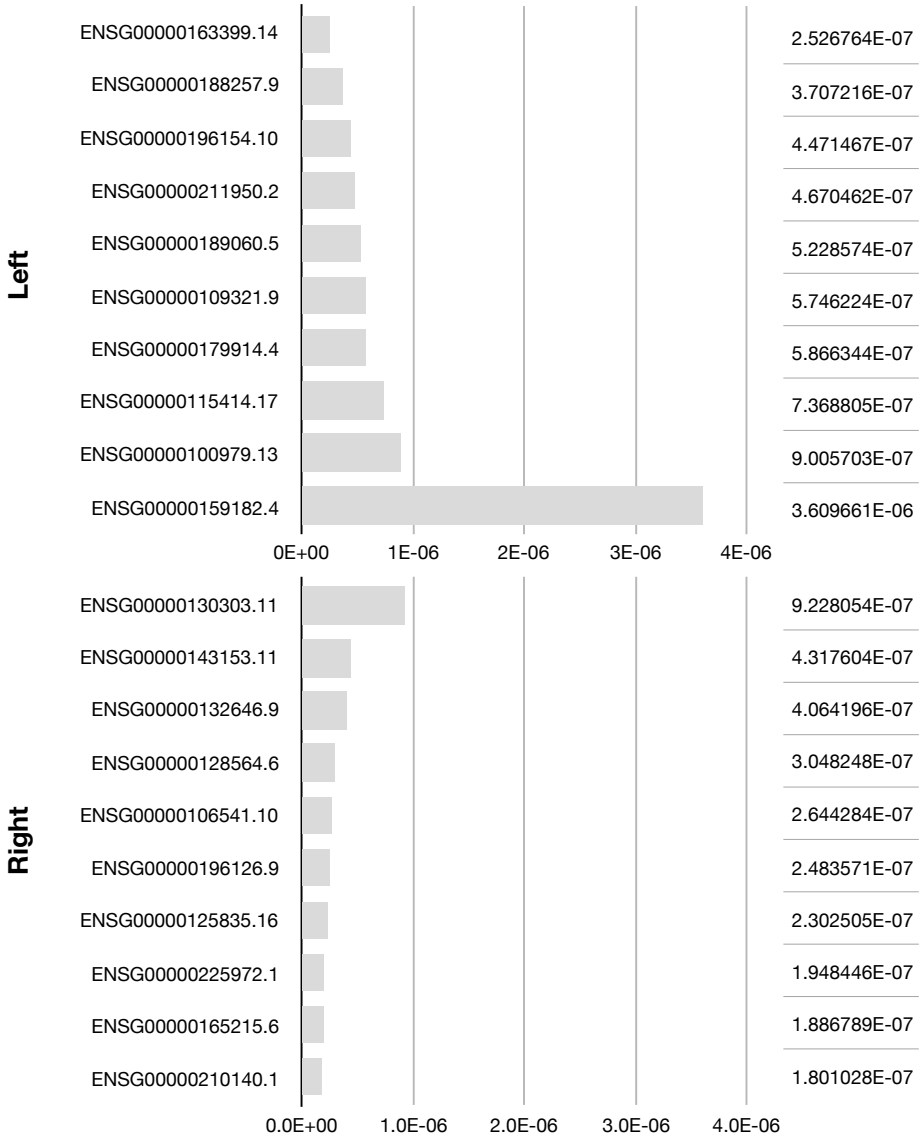

Figure 15: The 20 transcripts with largest regression coefficients (10 largest left and 10 largest right) as identified by the logit classifier.

Transcript	Gene	Description (57)(58)
ENSG00000159184.7	HOXB13	homeobox B13
ENSG00000180806.4	HOXC9	homeobox C9
ENSG00000159182.4	PRAC1	PRAC1 small nuclear protein
ENSG00000197757.7	HOXC6	homeobox C6
ENSG00000169676.5	DRD5	dopamine receptor D5

Transcript	Gene	Description (57)(58)
ENSG00000228630.4	HOTAIR	HOX transcript antisense RNA
ENSG00000128610.10	FEZF1	FEZ family zinc finger 1
ENSG00000230316.5	FEZF1-AS1	FEZF1 antisense RNA 1
ENSG00000229637.3	PRAC2	PRAC2 small nuclear protein
ENSG00000037965.5	HOXC8	homeobox C8

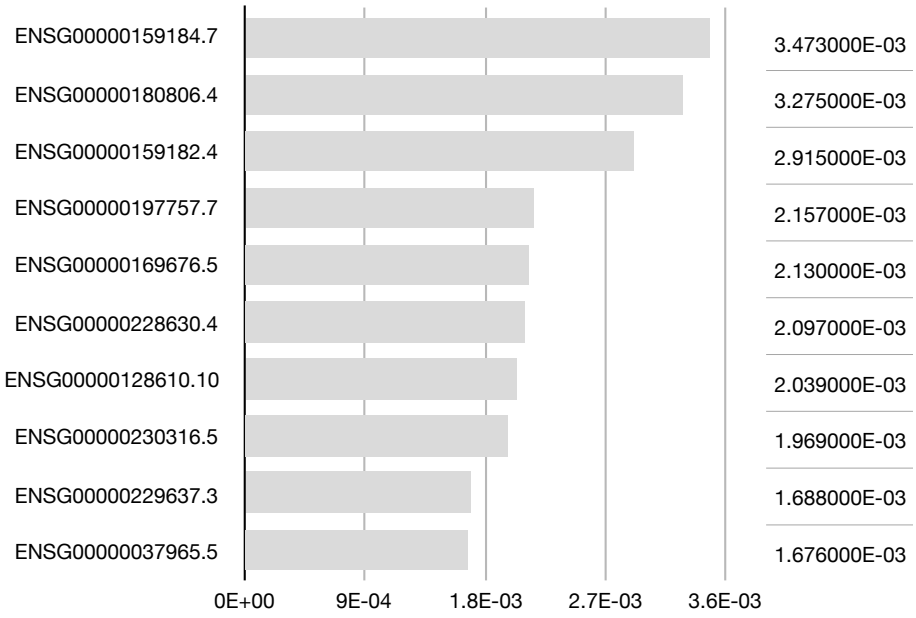

Figure 16: Ten most important features of RF classifier trained on transcriptomic data.

8 Discussion

It was hypothesized that it would be possible to find molecular markers which could help mechanistically explain the tumor-biological differences, in particular the differing response to anti-EGFR therapy, between left- and right-sided colorectal cancers by using machine learning classifiers to predict tumor location based on genomic and transcriptomic data from the TCGA database. The genomic and transcriptomic datasets differ significantly in their information density. While a genomic datum is boolean and therefore dichotomous, a transcriptomic datum encodes information on a much wider range of possibilities, allowing for a more subtle analysis. This explains why, although the `gini importance` per unit change is less for the transcriptomic data, the result is nonetheless more statistically significant.

By consistently reproducing findings from previously published work by other authors, the validity of the classifier as implemented in this thesis was demonstrated as a means of analyzing genomic and transcriptomic data. Unfortunately it was not possible to define further novel predictors of tumor location, as the statistical significance of the lesser predictors is questionable in light of the final sample size of $n = 123$ after adjusting the data.

The results are organized into a genomic and transcriptomic part, as these analyses were conducted separately from one another. The genomics results are discussed first to keep with the structure of the rest of this thesis.

8.1 Genomics

The genomics analysis did not yield statistically significant results. As such, no conclusions can be drawn from the results. For example, the gene of the top predictor, PLCZ1, is expressed exclusively in the acrosome of spermatozoa where it generates Ca^{2+} oscillations (59). It has no known connection to colorectal cancer and its usefulness in this classification setup is likely due to chance. It follows that the lesser predictors are even more likely to be no more than statistical noise.

8.2 Transcriptomics

The transcriptomics dataset yielded better results. This is likely due to the higher dimensionality of the data. Although there where less features in total compared to the genomics data, each feature carried more information than the dichotomous values in the genomics dataset.

Transcript ENSG00000159182.4, corresponding to the gene PRAC1, which was identified by the logit and RF classifiers as being highly predictive of left-sidedness, was identified recently by researchers in China as being significantly downregulated in right-sided coloadenocarcinoma (60) which is consistent also with previous findings (61). The positive correlation with left-sidedness fits well into this established framework.

ENSG00000130303.11 or BST2 was the second-best predictor and the only predictor of right-sidedness in the logit model. The exact function of the BST2 protein remains unclear, but it is likely involved in the restriction of infectivity of various viruses as part of the BST-2/CD317/HM1.24/tetherin complex (62). Additionally, BST2 has been shown to be overexpressed in colorectal cancers (63) and while the localization of the cases was not considered in the cited analysis (63), the overall post-operative survival of BST2-positive cases was worse than BST2-negative cases (63), which is consistent with the prediction of right-sidedness in the logit model.

Transcript ENSG00000100979.13 was the second-strongest predictor of left-sidedness in the logit regression model. It corresponds to the gene PLTP which plays an important role in lipoprotein metabolism and has been shown to be upregulated in obese men (64). Additionally, high HDL and hypercholesterolemia have been shown to be protective factors in CRC in mouse models (65). As such, if the PLTP protein is upregulated in left-sided coloadenocarcinomas and has a protective function, this could be a potential explanation for the better overall survival of patients with left-sided coloadenocarcinomas. However more research is needed to say for certain.

HOXB13 and HOXC9, as identified by the transcriptomic logit and random forest classifiers, belong to the homeobox (HOX) gene family, which is highly-conserved among vertebrates (66). HOXB13 is expressed exclusively in colon and prostate, indicating a potential association with the different levels of PRAC1 expression,

a gene similarly expressed in prostate, colon, and rectum. It has also been shown to be downregulated in left-sided colon cancers (67). HOXC9 has been associated with colorectal carcinogenesis (68). The high gini importance of this gene family in the random forest classification also fits well into this established framework.

PRAC1, HOXB13, HOXC9, HOXC6, HOTAIR, PRAC2, and HOXC8 all belong to the homebox gene family and their association with one another was uncovered by the random forest classifier. The fact that PRAC1 had the largest logit coefficient but was only third-best in terms of random forest gini importance is likely due to the ability of the random forest algorithm to assess compounding effects. This means that there are probably combinations of genes containing PRAC1, HOXB13, HOXC9, etc. in various combinations which predict tumor location better than any one transcript alone. PRAC1 belongs to the posterior HOXB gene cluster (69) and a group of researchers in Japan noted a significant difference in the expression of the homeobox gene family in the morphological domains of the digestive tract in a chick-model (70), which may indicate a neoplasia-independent marker of sidedness. However, while the classifier may have picked up on this physiological difference in expression, the fact that these genes are additionally down-regulated in left-sided colon cancers (67) may hint toward the progenitor tissue as a determining factor in CRC development. It remains unclear what role the various homeobox genes play in oncogenesis and regulation, but it appears to be very heterogeneous with some promoting and others inhibiting tumor growth (71). As such, more research is needed for a mechanistic explanation to be gleaned from these results.

Transcript ENSG00000115414.17 corresponds to the gene FN1 which is involved, among other things, in cell adhesion and metastasis (72). Fibronectin (along with collagen) was shown by the CMS research group to be associated with CMS4 (16), which has poor overall survival and relapse-free survival, compared to the other CMS. CMS4 is slightly more prevalent in left-sided CRC than in right-sided CRC (see Fig. 3). The fact that the logit regression identified this transcript as the fourth most important predictor fits into this established framework.

Transcript ENSG00000179914.4 corresponds to the gene ITLN1. ITLN1 plays a potential role in modulating insulin action and therefore glucose uptake (73). It additionally modulates the phosphorylation of Akt (73), thereby posing a

potential mechanism of anti-EGFR therapy resistance. More research is needed to say for certain if intelectin plays a mechanistic role in this context.

Transcript ENSG00000109321.9 corresponds to the gene AREG. Amphiregulin is a EGFR ligand and stimulates cell proliferation, apoptosis, and migration and has been shown to be upregulated in colon cancer (74). The fact that AREG was associated with left-sidedness may hint toward a more complex association, since left-sided CRC tend to have better overall survival. Perhaps the lack of AREG upregulation in right-sided CRC promotes the development of other, more detrimental tumor-biological changes. More research is needed to say for certain.

8.3 Closing Remarks

By framing the hypothesis around the notion that there is a discernible difference between left- and right-sided CRC on the genomic and transcriptomic level and training machine learning classifiers on the data, results were produced which show the differing impact the features have on predicting the tumor location. The results do not allow for the interpretation of the absolute impact of a feature, be it a genetic mutation or a relative difference in amounts of a transcript, on the biology of the cancer.

The classification analysis resulted in a number of genes which have been previously validated by other researchers and fit in the established framework both of oncogenic/tumor-biological (PRAC1, homeobox genes, FN1, and Amphiregulin) and metabolic (PLTP and ITLN1) factors. The tumor-biological factors represent potential but as-of-yet undeveloped chemotherapeutic targets such as cell-adhesion pathways and modulators of differentiation. However while these genes help to explain the tumor-biological differences between left- and right-sided CRC, it remains unknown if these molecular markers are perhaps only the symptom of some other, yet-to-be-determined mechanism. On the other hand, the metabolic factors underscore the importance of dietary and lifestyle factors influencing the development of colorectal cancer; CRC does not need to be treated via chemotherapy if it can be prevented by helping individuals regulate their cholesterol and blood sugar levels. As such, prevention will continue to play an important role in the fight against colorectal cancers for the foreseeable future.

The analysis was hampered by a lack of usable input data. Unfortunately, this meant that it was not possible to identify molecular predictors of tumor location beyond those previously documented. The fact that the transcriptomic data yielded significant results while the genomic data did not is likely due to the more precise nature of the data. It also reinforces the notion as stated in the CMS paper that transcriptomics allow for a more refined subclassification of CRC compared to genomics alone (16). Ideally, both genomic and transcriptomic data should be used in an intelligent ensemble approach in order to maximize the size of the incoming dataset, as this was the limiting factor in this thesis and continues to be a major obstacle for multi-omics-based analyses. The CMS paper for example used an intelligent ensemble containing data from 18 separate data sets, including the TCGA, normalized into a single matrix (16). However, the statistical expertise required for this approach was not available and went above and beyond the scope of this thesis. Another potential approach would be to disregard the genomics data altogether, thereby avoiding the loss of data via integration and maximizing the amount of data available to the transcriptomics classifier. This should be taken into account when approaching further research topics. Furthermore, in an attempt to increase the separation of the target classes, 23 intermediate cases (transverse colon, splenic flexure) from the TCGA dataset were ignored. Integrating these into the data and training not only for a binary but a multi-class outcome may also provide clues about the mechanism behind the tumor-biological differences seen clinically in CRC. For example, a gradual progression of transcript amount may hint towards gut milieu as an oncogenic factor, while an abrupt change around the splenic flexure would be more indicative of a embryological, progenitor-tissue dependent difference.

The choice of alternate classification algorithms may also lead to better results. While logistic regression is a robust algorithm with a proven track-record especially in the medical field, it can be outperformed by next-generation classifiers such as random forest or support vector machines. The random forest algorithm in particular is well-suited to the analysis of genomic and transcriptomic data where there are many more dimensions than there are samples, since the rate of convergence is proportional to only the subset of `important` features. The tradeoff is the potentially more complex interpretation of results since they may not be framed in the traditional units of measure many researchers have become accustomed to. Nevertheless, analysis using algorithms other than the ones used

here may potentially lead to further insights while avoiding the need to harvest new input data. Support vector machines (SVM) in particular show potential in genomics research due to their intrinsically high-dimensional nature. The computational complexity of SVM classification is also largely independent of the dimensionality of the data, rather on the sample size n (in the form $\frac{n(n-1)}{2}$), which tends to be rather small in research scenarios similar to this one. However, the limiting factor remains the amount of data. Regardless, the classifiers already trained in the context of this thesis can also be validated and/or further trained on new input data, should this data be collected some time in the future via projects similar to the TCGA Project.

It is encouraging to see that even with relatively simple means, using completely open-source and freely-available software packages, robust methods can be implemented and utilized to analyze the overwhelmingly large quantities of data available in the medical field today, in a way that is accessible and reproducible for anyone with a laptop and an internet connection. In addition, the methods put forth in this work and available online in the spirit of open research can be easily –even directly— applied to further analysis of the TCGA dataset.

The retrieval steps in particular are useful outside the scope of this project. While some packages already exist for accessing the TCGA dataset using the R programming language (75)(76), `RTCGA` has been failing its own build tests since September 2017, and `FirebrowseR` is a simple R wrapper for the Firebrowse REST API provided by the Broad Institute. So far, no packages are available for the python programming language. While this work does not constitute a plug-and-play solution, a point was made to use best-practices for the original work and common, well-maintained packages where ones already existed. As such, a user with basic understanding of programming and data structures could easily modify the codebase to suit their specific needs.

9 Jupyter Notebook

9.1 Notes

- In order to maintain consistency in naming conventions, files and cases are referred to strictly by their UUID.
- The original plan was to use a single `case_to_file` dictionary to store all files for a given case, and output all data into a common directory. It has remained in place since it most naturally represents the data as a file belonging to a case, but its use creates more work. May be refactored at a later time.

9.2 Set Flags

```
first_run = False
verbose = True

verboseprint = print if verbose else lambda *a, **k: None
```

9.3 Dependencies

```
import os
import glob
import requests
import csv
import shutil
from collections import defaultdict
try:
    import simplejson as json
except ImportError:
    import json
import gzip

import pandas as pd
import numpy as np
import xml.etree.ElementTree as ET
```

```
from scipy import interp
from matplotlib.pyplot import get_cmap

from plotly.offline import download_plotlyjs
from plotly.offline import init_notebook_mode
from plotly.offline import iplot
import plotly.graph_objs as go
init_notebook_mode()

from sklearn.manifold import TSNE
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold

from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

input_dir = os.path.join(os.getcwd(), 'in')
output_dir = os.path.join(os.getcwd(), 'out')

# initialize input directory
in_clin_manifest = os.path.join(
    input_dir,
    'clinical',
    'clinical_manifest.tsv')
in_clin_data = os.path.join(
    input_dir,
    'clinical',
    'data')
in_maf_file = os.path.join(input_dir,
  'maf',
  'TCGA.COAD.mutect.853e5584-b8c3-4836-9bda-6e7e84a64d97.DR-7.0.somatic.maf')
```

```
varsome_api_key_file = os.path.join(input_dir, 'varsome_api_key')

# initialize output directory
out_clin_data = os.path.join(output_dir, 'clinical', 'data')
out_clin_meta = os.path.join(output_dir, 'clinical', 'meta')
mut_df_file = os.path.join(output_dir, 'mut_df.gz')
trans_df_file = os.path.join(output_dir, 'trans_df.gz')

if first_run:
    for path in [out_clin_data, out_clin_meta]:
        os.mkdirs(path)

class tcgaFileQuery:
    def __init__(self, file_id):
        _cases_endpt = 'https://gdc-api.nci.nih.gov/files/ids'
        self._payload = {'query': file_id}
        self._response = requests.get(_cases_endpt, params=self._payload)
        self.data = self._response.json()

        self.case_id = self.data['data']['hits'][0]['cases'][0]['case_id']
        self.data_type = self.data['data']['hits'][0]['data_type']

def dump_json(json_data, path):
    with open(path, 'w') as jsonFile:
        json.dump(json_data, jsonFile, separators=(',', ': '), indent=4)

def fetch_metadata(manifest, output_location):
    file_to_case = {}
    UUID_to_filename = {}

    # metadata from server to file_to_case
    verboseprint('loading ' + manifest + '...')
    with open(manifest) as tsvFile:
        # get lenth of manifest file
        for line_count, value in enumerate(tsvFile):
            pass
```

```
tsvFile.seek(0)  # return to top of file
tsvData = csv.reader(tsvFile, delimiter='\t')

next(tsvData, None)  # skip header row of manifest file
for row in tsvData:
    file_id = row[0]
    UUID_to_filename[file_id] = row[1]

    verboseprint(
        str(line_count)
        + '\tfetching metadata for '
        + str(file_id)
        + '...',
        end=''
        )

    json_file = os.path.join(
        output_location,
        str(file_id) + '_meta.json')

    if first_run:
        queryResult = tcgaFileQuery(file_id)
        dump_json(queryResult.data, json_file)

       # store important metadata in databse
        file_to_case[file_id] = (queryResult.case_id,
                                 queryResult.data_type)

    else:
        with open(json_file, 'rt') as jsonFile:
            reader = jsonFile.read()
            jsonData = json.loads(reader)

        json_case_id = jsonData['data'][
            'hits'][0]['cases'][0]['case_id']
        json_data_type = jsonData['data'][
```

```
                    'hits'][0]['data_type']

                # store important metadata in databse
                file_to_case[file_id] = json_case_id, json_data_type

            verboseprint('done')
            line_count -= 1

    verboseprint('all metadata retrieved successfully', '\n')

    verboseprint('flipping association...', end=' ')
    # flip association, since one case potentially
    # contains multiple files
    case_to_file = {}
    for file_id in file_to_case:
        case_id = file_to_case[file_id][0]
        file_type = file_to_case[file_id][1]
        case_to_file[case_id] = file_id, file_type
    verboseprint('done')

    return case_to_file, UUID_to_filename

def remove_annotated(input_data, case_to_file, UUID_to_filename):

    # by using a defaultdict, even if multiple files
    # of a case are annotated, no duplicate entries are
    # created (as with a list) and no key errors result
    is_annotated = defaultdict(lambda: False)

    for case_id, f in case_to_file.items():
        file_id = f[0]
        path_to_file = os.path.join(input_data,
                                    file_id,
                                    UUID_to_filename[file_id])

#         verboseprint('looking at file', file_id, '... ')
```

```
        if os.listdir(os.path.split(path_to_file)[
            0]).count('annotations.txt') > 0:
            verboseprint('annotation found for case', case_id)
            is_annotated[case_id] = True

    verboseprint(len(is_annotated), 'annotated cases found')

    verboseprint('ignoring annotated cases')
    for case in is_annotated.keys():
        del case_to_file[case]

    return case_to_file
```

9.4 Retrieve from Disk

if the workflow has been executed before, the assembled DataFrame can be extracted from disk. Use the link below to jump to the Data Analysis section.

```
case_to_mut_df = pd.read_pickle(mut_df_file, compression='gzip')
```

```
case_to_trans_df = pd.read_pickle(trans_df_file, compression='gzip')
```

Jump to Exploration and Analysis

9.5 Genomics

9.5.1 Fetch Genomic Metadata

```
clin_c_to_f, clin_id_to_fn = fetch_metadata(in_clin_manifest, out_clin_meta)
```

9.5.2 Remove annotated cases

cases are annotated mostly due to some confounders and would require more complicated analysis to compensate for, so we will ignore them for the time being

```
clin_c_to_f = remove_annotated(in_clin_data, clin_c_to_f, clin_id_to_fn)
verboseprint('after removing annotated cases, our n =', len(clin_c_to_f))
```

9.5.3 Copy Clinical Data to Output

this way we retain the input as-is and can work on the output without worrying too much.

```
def clin_to_output():
    verboseprint('copying/extracting to output folder...')
    for case, file in clin_c_to_f.items():
        in_file = os.path.join(in_clin_data, file[0], clin_id_to_fn[file[0]])
        out_location = os.path.join(out_clin_data, case)
        out_file = os.path.join(out_clin_data, case, file[0] + '.xml')

        os.mkdir(out_location)
        verboseprint('copying file', file[0], 'to output...', end='')
        shutil.copyfile(in_file, out_file)
        verboseprint('done')

    verboseprint('extraction complete\n')
if first_run: clin_to_output()
```

9.5.4 Define "Left" & "Right"

```
anatomic_right = ('Cecum',
                  'Ascending Colon',
                  'Hepatic Flexure')
                    # 'Transverse Colon'
anatomic_left = ('Descending Colon',
                 'Sigmoid Colon',
                 'Rectosigmoid Junction')
                    #  'Splenic Flexure'
```

9.5.5 Extract Tumor Location

retrieve tumor location information from clinical data. some cases do not contain anatomic location information, so we cannot use them for our analysis

```
def extract_tumor_location_from_clin():
    tumor_location = {}
    cases_without_location = []
```

```
    for case, file in clin_c_to_f.items():
        # only for clinical xml data
        file_loc = os.path.join(out_clin_data, case, file[0] + '.xml')
        with open(file_loc) as xmlFile:
            tree = ET.parse(xmlFile)
            root = tree.getroot()

            if root[1][34].text == None:
                cases_without_location.append(case)
            elif root[1][34].text in anatomic_right:
                tumor_location[case] = False
            elif root[1][34].text in anatomic_left:
                tumor_location[case] = True

    verboseprint(len(cases_without_location),
                 'cases do not contain anatomic location information')

    for case in cases_without_location:
        verboseprint('ignoring case', case)
        del clin_c_to_f[case]

    # return as pandas Series
    return pd.Series(tumor_location)
tumor_location = extract_tumor_location_from_clin()
```

9.6 The MAF File

9.6.1 Format

- Format specification
- TCGA search result for all COAD MAF files
- Legacy Tumor-Normal Pair

9.6.2 The Problem

cells are defined by their genes. cancers carry variations in those genes which cause aberrant behavior. however, this causal relationship is not one-to-one, and additionally these variations do not need to be somatic mutations. they can be part of the individuals genome before the cancer develops. we require some function which is capable of separating irrelevant 'polymorphisms' from cancer-causing 'mutations'.

9.6.3 The Solution

The GDCs approach is the MAF file. It compares the tumor genome to the individuals normal genome and a reference sequence. The choice of reference sequence can impact the results. There are multiple workflows which achieve this result

- varscan
- mutect
- muse
- somaticsniper

9.6.4 HGVSc

- stands for "Human Genome Variation Society, coding". the full nomenclature definition can be found here. there are following possible suffixes.
  - "g" genomic
  - "m" mitochondrial
  - "c" coding DNA
  - "n" non-coding DNA
  - "r" RNA reference sequence (transcript)
  - "p" protein reference sequence
- the HGVSc variant information is based off of reference sequences. these can be found in `Gene`

```
# columns of maf file (without loading entire file)
! echo $in_maf_file
! head -6 $in_maf_file | tail -1 | tr '\t' '\n'
```

9.6.5 TODO: Reference_Allele vs HGVSc

in synonymous variant entries, the `Reference_Allele` vs `Allele` **differs** from the `HGVSc` notation! * for example, for `PERM1` the alleles are `G` and `A`, but the `HGVSc` notation specifies `C>T`

```
# mutect and somaticsniper share columns
maf_df = pd.read_csv(in_maf_file, sep='\t',
            usecols=[
                'Hugo_Symbol',
#                 'Reference_Allele',
                'HGVSc',
                'HGVSp',
#                 'Allele',
#                 'Gene',
                'Consequence',
                'case_id'
            ],
           header=5)

maf_df.head()
```

9.6.6 Intersection of Sets

some cases in the data portal do not have entries in the MAF file. analogously, some entries in the MAF file do not seem to be associated with any case in the current data portal version. we may only use the intersection of these two sets.

`tumor_location` is derived from `clin_c_to_f - clin_is_annotated - cases_without_location`.
some of these cases are not present in the MAF file. therefore we cannot use them.

```
bad_cases = []
for case in tumor_location.keys():
    if case not in maf_df['case_id'].unique():
        verboseprint('case', case, 'has no entry in MAF file')
        bad_cases.append(case)
```

```
tumor_location.drop(bad_cases, inplace=True)
del bad_cases

# trimmed of cases that are not in MAF
tumor_location.shape[0]

# maf not yet trimmed
len(maf_df['case_id'].unique())

# for some cases in MAF we do not have adequate data.
# therefore we must remove them.
# drop entries where the case_id is NOT (~) in tumor_location.
entries_of_cases_without_location = maf_df.loc[~maf_df[
    'case_id'].isin(tumor_location.keys())]['case_id'].index

maf_df.iloc[entries_of_cases_without_location].case_id.unique().shape[0]

maf_df.drop(entries_of_cases_without_location, inplace=True)

# number of cases matches trimmed location cases if == 0
len(maf_df['case_id'].unique()) - tumor_location.shape[0]

maf_df['case_id'].unique().shape[0]
```

9.6.7 Remove k- and n-RAS

- Mutations in these genes correlate strongly with resistance to anti-IgF therapy.
- Hugo Symbols are `KRAS` and `NRAS`
- unfortunately, this reduces our n by about 50%

```
kRAS_cases = maf_df.loc[maf_df['Hugo_Symbol'] == 'KRAS']['case_id']
nRAS_cases = maf_df.loc[maf_df['Hugo_Symbol'] == 'NRAS']['case_id']

knRAS_cases_set = set(kRAS_cases.tolist() + nRAS_cases.tolist())

len(knRAS_cases_set)

entries_of_cases_with_knRAS_mutations = maf_df.loc[maf_df[
    'case_id'].isin(knRAS_cases_set)].index

maf_df.drop(entries_of_cases_with_knRAS_mutations, inplace=True)

maf_df['case_id'].unique().shape[0]
```

9.6.8 Remove Synonymous Variants

using the `Consequence` column, value: `synonymous_variant`

```
synonymous_variants = maf_df.loc[
    maf_df['Consequence'] == 'synonymous_variant'].index
maf_df.drop(synonymous_variants, inplace=True)

synonymous_variants.shape[0]
```

9.6.9 Unique Variant Identifier

In order to completely define our mutations, we must combine multiple columns. A complete definition can be formatted as `Gene:HGVSc` * i.e. `PERM1` is `ENSG00000187642:c.1827C>T`

```
# create unique variant ID, store in column 'uvi'
# note, there are some NaN in the data set
maf_df.loc[:,'uvi'] = pd.Series(maf_df[
    'Hugo_Symbol'] + ':' + maf_df['HGVSc'], index=maf_df.index)

case_to_mut_df = pd.DataFrame(False, index=maf_df[
    'case_id'].unique(), columns=maf_df.uvi.unique())

# add column to end of DataFrame, left = True
case_to_mut_df.loc[:, 'tumor_loc_left'] = tumor_location

for case, mutations in maf_df.groupby('case_id')['uvi']:
    for mut in mutations:
        case_to_mut_df.at[case, mut] = True

def mut_freq_histogram():
    df_left = case_to_mut_df.loc[case_to_mut_df['tumor_loc_left'] == True]
    df_right = case_to_mut_df.loc[case_to_mut_df['tumor_loc_left'] == False]

    df_left.drop('tumor_loc_left', axis=1)
    df_right.drop('tumor_loc_left', axis=1)

    count_left = []
    for case in df_left.index:
        count_left.append(df_left.loc[case].value_counts()[1])
```

```
    count_right = []
    for case in df_right.index:
        count_right.append(df_right.loc[case].value_counts()[1])

    trace1 = go.Histogram(
        x=count_left,
        name='left-sided',
        opacity=0.8
        )

    trace2 = go.Histogram(
        x=count_right,
        name='right-sided',
        opacity=0.8
        )

    data = [trace1, trace2]
    layout = go.Layout(barmode='overlay')
    fig = go.Figure(data=data, layout=layout)

    iplot(fig)
mut_freq_histogram()
```

9.6.9.1 Remove hypermutated cases

cases with more than 250 mutations are removed to approximately account for MSI status

```
count = 0
for case in case_to_mut_df.index:
    if case_to_mut_df.loc[case].value_counts()[1] > 250:
        count += 1
        case_to_mut_df.drop(case, inplace=True)
print('removed', count, 'cases')
```

```
case_to_mut_df.shape[0]
```

9.6.10 Save to Disk

to save us the hassle of reevaluating every time, store finished, assembled DataFrame to disk.

```
case_to_mut_df.to_pickle(mut_df_file, compression='gzip')
```

9.7 Transcriptome

using transcriptome information from the TCGA Database, we can expand the amount of information per case.

possibilities include * adding transcriptome information to the MAF Dataframe, thereby expanding the dimensionality of our data set * This could however be potentially problematic, because we would introduce continuous values into our previously purely dichotomous data. This could make the analysis more complicated. * creating a separate database, running a logistic regression on that set, and comparing it to the genomic regression. In theory it should be possible to combine the results of both regressions to get a combined result. * associating the variants with their corresponding transcript, and using some combination of this data to train our classifier. * It is unclear to me how the combined value should be calculated, and what it would represent. For example, if there are `20` variants in gene `x1` and the corresonding transcipt `x2` has a FPKM-UQ value of `1000` what would the function f(x1, x2), used to combine these two bits of information, look like?

```
in_trans_manifest = os.path.join(
    input_dir,
    'transcriptome',
    'transcriptome_manifest.tsv')
in_trans_data = os.path.join(input_dir, 'transcriptome', 'data')


out_trans_data = os.path.join(output_dir, 'transcriptome', 'data')
out_trans_meta = os.path.join(output_dir, 'transcriptome', 'meta')

trans_c_to_f, trans_id_to_fn = fetch_metadata(
    in_trans_manifest,
    out_trans_meta)

trans_c_to_f = remove_annotated(
```

```
    in_trans_data,
    trans_c_to_f,
    trans_id_to_fn)

len(trans_c_to_f)

bad_cases = []
for case in trans_c_to_f.keys():
    if case not in case_to_mut_df.index:
        verboseprint('case', case, 'has no entry in MAF file')
        bad_cases.append(case)


verboseprint(len(bad_cases), 'cases will be removed from analysis')
for case in bad_cases:
    trans_c_to_f.pop(case)
del bad_cases

len(trans_c_to_f)

case_to_mut_df.shape

def trans_extract_to_output():
    verboseprint('copying/extracting to output folder...')
    for case, file in trans_c_to_f.items():
        in_file = os.path.join(in_trans_data, file[0], trans_id_to_fn[file[0]])
        out_location = os.path.join(out_trans_data, case)
        out_file = os.path.join(out_trans_data, case, file[0] + '.txt')

        os.mkdir(out_location)
        verboseprint('copying file', file[0], 'to output...', end='')
        with gzip.open(in_file, 'rb') as f:
            file_content = f.read()
        with open(out_file, 'wb') as f:
            f.write(file_content)
        verboseprint('done')


    verboseprint('extraction complete\n')

if first_run: trans_extract_to_output()
```

```
for case in case_to_mut_df.index:
    try: trans_c_to_f[case]
    except KeyError:
        verboseprint(case)
        case_to_mut_df.drop(case, inplace=True)

def make_transcriptome_DataFrame():
    transcriptome = defaultdict(dict)

    for case, file in trans_c_to_f.items():
        file_loc = os.path.join(out_trans_data, case, file[0] + '.txt')
        with open(file_loc, 'r') as tsvFile:
            tsvData = csv.reader(tsvFile, delimiter='\t')
            for row in tsvData:
                transcriptome[case][row[0]] = row[1]

    return pd.DataFrame.from_dict(transcriptome, orient='index', dtype='float')

case_to_trans_df = make_transcriptome_DataFrame()

case_to_trans_df.loc[:, 'tumor_loc_left'] = tumor_location
```

9.7.1 Save to Disk

to save us the hassle of reevaluating every time, store finished, assembled DataFrame to disk.

```
case_to_trans_df.to_pickle(trans_df_file, compression='gzip')
```

9.8 Exploration and Analysis

9.8.1 Naming Convention

`X` is the information we give the regression model, `y` is the correct result.

9.8.2 t-Distributed Stochastic Neighbor Embedding

We can use t-SNE to visualize our dataset in 2 or 3 dimensions, to try to gain further insight into our data set. It is useful for quickly reducing the dimensionality of a dataset without any prior insight. In this example, 3 dimensions are used. this can be chosen using the `n_components` variable.

The model reduces high-dimensional datasets to something that humans are capable of comprehending. May yield different results on repeated runs.

9.8.3 Logistic Regression

A binary logistic regressor, using nested cross-validation. * Outer CV * Sample Selection: Stratified k-fold * Scoring: **R**eceiver **O**perating **C**haracteristic **A**rea **U**nder the **C**urve * Inner CV * Hyperparameter `C` is assessed for `n_inner_folds` (default: 10) logarithmically spaced values between 10-4 and 104 * Sample Selection: Stratified k-fold * Scoring: Accuracy

9.8.4 Random Forest

A binary classifier based on random subsampling and decision trees.

```
def get_n_colors(n):
    """return list of colors from viridis colorspace for use with plotly"""

    cmap = get_cmap('viridis')
    colors_01 = cmap(np.linspace(0, 1, n))
    colors_255 = []

    for row in colors_01:
        colors_255.append(
            'rgba({}, {}, {}, {}'.format(
                row[0] * 255,
                row[1] * 255,
                row[2] * 255,
                row[3]
                )
            )

    return colors_255

def tsne(df):

    tsne = TSNE(n_components=3, perplexity=20, n_iter=5000)
```

```
X = df.drop('tumor_loc_left', axis=1)
y = df['tumor_loc_left']

X_3d = tsne.fit_transform(X)

verboseprint('t-SNE')
verboseprint('-----')
verboseprint('')
verboseprint('reduced dimensionality from {} to {} dimensions'.format(
    X.shape[1], X_3d.shape[1]
    )
)

trace_true = go.Scatter3d(
    x = X_3d[y==True, 0],
    y = X_3d[y==True, 1],
    z = X_3d[y==True, 2],
    name = 'left',
    mode = 'markers',
    marker = dict(
        color = 'rgba(37, 146, 34, 0.8)'
        ),
    text = X[y==True].index.tolist()
    )
trace_false = go.Scatter3d(
    x = X_3d[y==False, 0],
    y = X_3d[y==False, 1],
    z = X_3d[y==False, 2],
    name = 'right',
    mode = 'markers',
    marker = dict(
        color = 'rgba(209, 28, 36, 0.8)'
        ),
    text = X[y==False].index.tolist()
    )
layout = go.Layout(title='t-SNE Representation')
```

```
    fig = go.Figure(data=[trace_true, trace_false], layout=layout)

    iplot(fig)

class logit_clf:
    def __init__(self, df, n_inner_folds=12, n_outer_folds=10, v=0):

        self.df = df
        self.n_outer_folds = n_outer_folds

        self.X = df.drop('tumor_loc_left', axis=1)
        self.y = df['tumor_loc_left']

        self.classifier = LogisticRegressionCV(
                                Cs=10,
                                fit_intercept=True,
                                cv=n_inner_folds,
                                dual=False,
                                penalty='l1',
                                solver='liblinear',
                                tol=0.0001,
                                max_iter=100,
                                class_weight=None,
                                n_jobs=-1,
                                verbose=v,
                                refit=True,
                                intercept_scaling=1.0,
                                random_state=3257
                                )
        self.cv = StratifiedKFold(
            n_splits=self.n_outer_folds,
            random_state=5235)

    def fit_and_print_roc(self):
        tprs = []
```

```
aucs = []
mean_fpr = np.linspace(0, 1, 100)

plotly_data = []

colors = get_n_colors(self.n_outer_folds)

i = 0
for train, test in self.cv.split(self.X, self.y):
    probabilities_ = self.classifier.fit(
                        self.X.iloc[train],
                        self.y.iloc[train]
                        ).predict_proba(self.X.iloc[test])

    # Compute ROC curve and area under the curve
    fpr, tpr, thresholds = roc_curve(
        self.y.iloc[test],
        probabilities_[:, 1])
    tprs.append(interp(mean_fpr, fpr, tpr))
    tprs[-1][0] = 0.0
    roc_auc = auc(fpr, tpr)
    aucs.append(roc_auc)
    plotly_data.append(
        go.Scatter(
            x=fpr,
            y=tpr,
            name='ROC fold {} (AUC = {})'.format(
                i, round(roc_auc,2)),
            line=dict(color=colors[i], width=1)
            )
        )
    i += 1

# add ROC reference line
plotly_data.append(
    go.Scatter(
```

```
            x=[0, 1],
            y=[0, 1],
            mode='lines',
            line=dict(
                color='navy',
                width=2,
                dash='dash'
                ),
            showlegend=False
            )
        )

            # mean
mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)

# Standard Deviation
std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
plotly_data.append(
    go.Scatter(
        x=mean_fpr,
        y=tprs_upper,
        name='upper bound',
        line=dict(color='grey'),
        opacity=0.1,
        showlegend=False
        )
    )
# plot mean above std deviation
plotly_data.append(
    go.Scatter(
        x=mean_fpr,
```

```
                y=tprs_lower,
                name='± 1 std. dev.',
                fill='tonexty',
                line=dict(color='grey'),
                opacity=0.1
                )
            )

        plotly_data.append(
            go.Scatter(
                x=mean_fpr,
                y=mean_tpr,
                name='Mean ROC (AUC = {} ± {})'.format(
                    round(mean_auc, 2),
                    round(std_auc, 2)
                    ),
                line=dict(color='darkorange', width=3),
                )
            )

        layout = go.Layout(title='Receiver operating characteristic',
                   xaxis=dict(title='False Positive Rate'),
                   yaxis=dict(title='True Positive Rate')
                   )
        fig = go.Figure(data=plotly_data, layout=layout)

        iplot(fig)

    def get_n_pos_most_important(self, r_min, r_max, n=10, plot=True):
        """Return Variants whose Regression Coefficients are largest.

        assumes that the target class is `left`.
        """
        coefs = self.classifier.fit(self.X, self.y).coef_[0]
        indices = np.argsort(coefs)[::-1]
```

```
        n_indices = []
        for i in range(n):
            n_indices.append(indices[i])

        if plot:
            trace1 = go.Bar(x=self.df.columns[n_indices],
                            y=coefs[n_indices],
                            text = self.df.columns[n_indices],
                            marker=dict(color='green'),
                            opacity=0.5
                           )

            layout = go.Layout(
                title="Positive Regression Coefficients",
                yaxis=dict(range=[r_min, r_max])
            )
            fig = go.Figure(data=[trace1], layout=layout)

            iplot(fig)

    def get_n_neg_most_important(self, r_min, r_max, n=10, plot=True):
        """Return Variants whose Regression Coefficients are largest.

        assumes that the target class is `left`.
        """
        coefs = self.classifier.fit(self.X, self.y).coef_[0]
        indices = np.argsort(coefs)

        n_indices = []
        for i in range(n):
            n_indices.append(indices[i])

        if plot:
            trace1 = go.Bar(x=self.df.columns[n_indices],
                            y=coefs[n_indices],
                            text = self.df.columns[n_indices],
```

```
                        marker=dict(color='green'),
                        opacity=0.5
                       )

        layout = go.Layout(
            title="Negative Regression Coefficients",
            yaxis=dict(range=[r_min, r_max])
        )
        fig = go.Figure(data=[trace1], layout=layout)

        iplot(fig)

def get_n_most_sided(self, n=10):
    """return two dicts, right and left, each with n case UUIDs
    and respective P(right-sidedness) - P(left-sidedness), where
    this value is minimal in the classifier.

    the classifier must be fit() before using this function.
    """

    # The order of the classes in probas corresponds to that
    # in the attribute classes_, in this case [False, True]

    probas = self.classifier.predict_proba(self.X)
    proba_diffs = np.array(probas[:,0] - probas[:,1])

    indices = np.argsort(proba_diffs)

    left_indices = []
    right_indices = []
    for i in range(n):
        left_indices.append(indices[i])
        right_indices.append(np.flip(indices, 0)[i])

    right = {}
    for i in right_indices:
```

```
            case_id = self.df.index[i]
            proba_diff = round(proba_diffs[i], 4)
            right[case_id] = proba_diff

        left = {}
        for i in left_indices:
            case_id = self.df.index[i]
            proba_diff = round(proba_diffs[i], 4)
            left[case_id] = proba_diff

        return right, left

class rf_clf:
    def __init__(self, df, n_trees=1000, v=0):
        self.df = df
        self.X = df.drop('tumor_loc_left', axis=1)
        self.y = df['tumor_loc_left']
        self.classifier = RandomForestClassifier(
                                n_estimators=n_trees,
                                criterion='gini',
                                max_features='sqrt',
                                max_depth=None,
                                min_samples_split=2,
                                min_samples_leaf=1,
                                min_weight_fraction_leaf=0.0,
                                max_leaf_nodes=None,
                                min_impurity_decrease=0.0,
                                min_impurity_split=None,
                                bootstrap=True,
                                oob_score=True,
                                n_jobs=-1,
                                random_state=3576,
                                verbose=v,
                                warm_start=False,
                                class_weight=None
                                )
```

```
def fit(self):
    self.classifier.fit(self.X, self.y)

def print_roc(self):
    tprs = []
    aucs = []
    mean_fpr = np.linspace(0, 1, 100)

    y_decision = self.classifier.oob_decision_function_[:,1]
    fpr, tpr, thresholds = roc_curve(self.y, y_decision)
    roc_auc = auc(fpr, tpr)

    trace0 = go.Scatter(
        x=fpr,
        y=tpr,
        name='ROC AUC = {}'.format(round(roc_auc, 2)),
        line=dict(color='darkorange', width=2)
        )
    trace1 = go.Scatter(
        x=[0, 1],
        y=[0, 1],
        mode='lines',
        line=dict(color='navy', width=2, dash='dash'),
        showlegend=False
        )
    layout = go.Layout(
                title='Receiver operating characteristic',
                xaxis=dict(title='False Positive Rate'),
                yaxis=dict(title='True Positive Rate')
                )

    data = [trace0, trace1]
    fig = go.Figure(data=data, layout=layout)

    iplot(fig)
```

```
def get_n_most_important(self, n=10, plot=True):
    """Return Variants IDs deemed most important by the Classifier.

    Metric used by rf clf is (by default) `gini impurity`, or
    if specified, entropy gain.
    """
    def std_dev_of_features(n=8000):
        """use chunking to get std dev in memory-efficient way

        For large numbers of features and large numbers of trees,
        the memory required to store the input matrix for np.std becomes
        unfeasable. Here, the approach is to scan through all trees
        multiple times, selecting only a batch of features at a time
        and then calculating the std dev on that batch. This way we avoid
        loading the entire matrix of possibilities into memory at once.
        """

        std = np.zeros(self.X.shape[1])
        for i in range(0, self.X.shape[1], n):
            chunk_buffer = []
            for tree in self.classifier.estimators_:
                chunk_buffer.append(tree.feature_importances_[i:i+n])
            std[i:i+n] = np.std(chunk_buffer, axis=0)
            del chunk_buffer

        return std

    std = std_dev_of_features()
    importances = self.classifier.feature_importances_
    indices = np.flip(np.argsort(importances), 0)

    n_indices = []
    for i in range(n):
        n_indices.append(indices[i])

    if plot:
```

```
            # plotly
            trace = go.Bar(x=self.df.columns[n_indices],
                           y=importances[n_indices],
                           text = self.df.columns[n_indices],
                           marker=dict(color='green'),
                           error_y=dict(
                               visible=True,
                               arrayminus=std[n_indices]),
                           opacity=0.5
                          )

            layout = go.Layout(title="Feature importance")
            fig = go.Figure(data=[trace], layout=layout)

            iplot(fig)

        return self.df.columns[n_indices].tolist()

    def get_n_most_sided(self, n=10):
        """return two dicts, right and left, each with n case UUIDs
        and respective P(right-sidedness) - P(left-sidedness), where
        this value is minimal in the classifier.

        the classifier must be fit() before using this function.
        """

        # The order of the classes in probas corresponds to that
        # in the attribute classes_, in this case [False, True]

        probas = self.classifier.predict_proba(self.X)
        proba_diffs = np.array(probas[:,0] - probas[:,1])

        indices = np.argsort(proba_diffs)

        left_indices = []
        right_indices = []
```

```
        for i in range(n):
            left_indices.append(indices[i])
            right_indices.append(np.flip(indices, 0)[i])

        right = {}
        for i in right_indices:
            case_id = self.df.index[i]
            proba_diff = round(proba_diffs[i], 4)
            right[case_id] = proba_diff

        left = {}
        for i in left_indices:
            case_id = self.df.index[i]
            proba_diff = round(proba_diffs[i], 4)
            left[case_id] = proba_diff

        return right, left
tsne(case_to_trans_df)
```

```
t-SNE
-----

reduced dimensionality from 60483 to 3 dimensions
```

```
mut_logit_clf = logit_clf(case_to_mut_df)
mut_logit_clf.fit_and_print_roc()
mut_logit_important = mut_logit_clf.get_n_most_important()
mut_logit_sided = mut_logit_clf.get_n_most_sided()

forest_size = case_to_mut_df.shape[1] // 4
mut_rf_clf = rf_clf(case_to_mut_df, n_trees=forest_size, v=1)
mut_rf_clf.fit()
mut_rf_clf.print_roc()
mut_rf_important = mut_rf_clf.get_n_most_important()
mut_rf_sided = mut_rf_clf.get_n_most_sided()

trans_logit_clf = logit_clf(case_to_trans_df, v=1)
trans_logit_clf.fit_and_print_roc()
```

```
trans_logit_important = trans_logit_clf.get_n_most_important()
trans_logit_sided = trans_logit_clf.get_n_most_sided()

forest_size = case_to_trans_df.shape[1] // 4
trans_rf_clf = rf_clf(case_to_trans_df, v=1)
trans_rf_clf.fit()
trans_rf_clf.print_roc()
trans_rf_important = trans_rf_clf.get_n_most_important()
trans_rf_sided = trans_rf_clf.get_n_most_sided()
```

TODO we can get the `n` cases which best represent left- or right-sidedness by finding `P(left) - P(right)`. Then, we take the case UUIDs and download the slides from the legacy TCGA. Prof. Kirchner can then typify them.

```
import pickle
```

9.9 Save Classifiers to Disk

```
with open('./out/classifiers/mut_logit_clf.pickle', 'wb') as f:
    # Pickle the 'data' dictionary using the highest protocol available.
    pickle.dump(mut_logit_clf, f, pickle.HIGHEST_PROTOCOL)

with open('./out/classifiers/mut_rf_clf.pickle', 'wb') as f:
    # Pickle the 'data' dictionary using the highest protocol available.
    pickle.dump(mut_rf_clf, f, pickle.HIGHEST_PROTOCOL)

with open('./out/classifiers/trans_logit_clf.pickle', 'wb') as f:
    # Pickle the 'data' dictionary using the highest protocol available.
    pickle.dump(trans_logit_clf, f, pickle.HIGHEST_PROTOCOL)

with open('./out/classifiers/trans_rf_clf.pickle', 'wb') as f:
    # Pickle the 'data' dictionary using the highest protocol available.
    pickle.dump(trans_rf_clf, f, pickle.HIGHEST_PROTOCOL)
```

9.10 Retrieve Classifiers from Disk

```
# Mutation Logistic Regression
with open('./out/classifiers/mut_logit_clf.pickle', 'rb') as f:
    unpickler = pickle.Unpickler(f)
    mut_logit_clf = unpickler.load()
```

```
# Mutation RandomForest
with open('./out/classifiers/mut_rf_clf.pickle', 'rb') as f:
    unpickler = pickle.Unpickler(f)
    mut_rf_clf = unpickler.load()

# Transcriptome Logistic Regression
with open('./out/classifiers/trans_logit_clf.pickle', 'rb') as f:
    unpickler = pickle.Unpickler(f)
    trans_logit_clf = unpickler.load()

# Transcriptome RandomForest
with open('./out/classifiers/trans_rf_clf.pickle', 'rb') as f:
    unpickler = pickle.Unpickler(f)
    trans_rf_clf = unpickler.load()

mut_rf_clf.classifier.predict_proba(
    case_to_mut_df.drop(
        'tumor_loc_left', axis=1))[:,0]

mut_logit_clf.get_n_pos_most_important(r_min=0, r_max=8)

mut_logit_clf.get_n_neg_most_important(r_min=-8, r_max=0)

trans_logit_clf.get_n_pos_most_important(r_min=0, r_max=0.000004)

trans_logit_clf.get_n_neg_most_important(r_min=-0.000004, r_max=0)

np.array(
    mut_rf_clf.classifier.predict_proba(
        case_to_mut_df.drop(
            'tumor_loc_left', axis=1))[
        :,0] - mut_rf_clf.classifier.predict_proba(
        case_to_mut_df.drop('tumor_loc_left', axis=1))[:,1]
)
```

9.11 Variant Information

- using the VarSome search engine API, we can retrieve a large amount of information about our variants.
- Reference API Client written in Python, hosted on GitHub

9.11.0.1 Available GET Parameters

params are passed as dict with key value pairs, for example
`params={'expand-pubmed-articles': 1, 'add-source-databases': 'gerp,wustl-civic'}`

- `add-all-data` = 1 or 0
- `add-region-databases` = 1 or 0
- `expand-pubmed-articles` = 1 or 0
- `add-main-data-points` = 1 or 0
- `add-source-databases` = all or none or
  - gerp
  - wustl-civic
  - ncbi-dbsnp
  - ensembl-transcripts
  - broad-exac
  - dbnsfp-dbscsnv
  - gwas
  - ncbi-clinvar
  - gnomad-genomes
  - dbnsfp
  - sanger-cosmic
  - dann-snvs
  - sanger-cosmic-public
  - gnomad-exomes
  - ncbi-clinvar2
  - refseq-transcripts
  - uniprot-variants
  - gnomad-genomes-coverage
  - gnomad-exomes-coverage
  - thousand-genomes
  - isb-kaviar3
  - iarc-tp53-germline
  - sanger-cosmic-licensed
  - iarc-tp53-somatic
  - icgc-somatic
- `allele-frequency-threshold` = float

```
from variantapi.client import VariantAPIClient
varsome_api_key = with open(varsome_api_key_file) as f: f.read()
varsome_api = VariantAPIClient(varsome_api_key)

result = varsome_api.batch_lookup(
    list(significant_var_dict.keys()),
    params={'add-source-databases': 'gerp'},
    ref_genome='hg19')

result
```

9.11.1 various test snippets

```
missence_variants = maf_df.loc[
    maf_df['Consequence'] == 'missense_variant'].index
var_list = []
for entry in maf_df.loc[missence_variants][
    ['Hugo_Symbol', 'HGVSc']].values:
    # concatenate as HUGO:HGVSc
    var_def = entry.tolist()[0] + ':' + entry.tolist()[1]
    var_list.append(var_def)

# create batches of at most n variants
n = 10
q_chunks = [var_list[i:i+n] for i in range(
    0, len(var_list), n)]
```

TODO: above, n is set to 10 for testing, for final release, set to 10000 (ten thousand)

```
result = varsome_api.batch_lookup(q_chunks[0],
                    params={'add-source-databases': 'gerp'},
                    ref_genome='hg19')

interesting = ['alt', 'chromosome']

[result[0][x] for x in interesting]
```

10 Acknowledgements

Foremost, I would like to thank my advisor Prof. Andreas Jung and Dr. Jörg Kumbrink for their continued support of my research. Their feedback, motivation, and guidance have been an immeasurable help in completing this work.

I would also like to thank Jochen von Seckendorff for providing access to the required computational infrastructure and Vage Shirvanyan, Johanna von Seckendorff, and Marietta von Siemens for proofreading my drafts.

References

1. Ahmedin J, Freddie B, M. CM, Jacques F, Elizabeth W, David F. Global cancer statistics. CA: A Cancer Journal for Clinicians [Internet] 61(2):69–90. Available from: https://onlinelibrary.wiley.com/doi/abs/10.3322/caac.20107

2. Schmiegel W, Buchberger B, Follmann M, et al. S3-leitlinie–kolorektales karzinom. Zeitschrift für Gastroenterologie 2017;55(12):1344–498.

3. M. CM, Ahmedin J, A. SR, Elizabeth W. Worldwide variations in colorectal cancer. CA: A Cancer Journal for Clinicians [Internet] 59(6):366–78. Available from: https://onlinelibrary.wiley.com/doi/abs/10.3322/caac.20038

4. Haggar FA, Boushey RP. Colorectal cancer epidemiology: Incidence, mortality, survival, and risk factors. Clinics in colon and rectal surgery 2009;22(4):191.

5. Popkin BM. The nutrition transition: An overview of world patterns of change. Nutrition reviews 2004;62(suppl_2):S140–3.

6. Kono S. Secular trend of colon cancer incidence and mortality in relation to fat and meat intake in japan. European journal of cancer prevention 2004;13(2):127–32.

7. JA B. Colorectal cancer: Evidence for distinct genetic categories based on proximal or distal tumor location. Annals of Internal Medicine [Internet] 1990;113(10):779–88. Available from: http://dx.doi.org/10.7326/0003-4819-113-10-779

8. Missiaglia E, Jacobs B, D'Ario G, et al. Distal and proximal colon cancers differ in terms of molecular, pathological, and clinical features. Annals of Oncology [Internet] 2014;25(10):1995–2001. Available from: + http://dx.doi.org/10.1093/annonc/mdu275

9. Lièvre A, Bachet J-B, Le Corre D, et al. KRAS mutation status is predictive of response to cetuximab therapy in colorectal cancer. Cancer Research [Internet] 2006;66(8):3992–5. Available from: http://cancerres.aacrjournals.org/content/66/8/3992

10. Di Nicolantonio F, Martini M, Molinari F, et al. Wild-type braf is required for response to panitumumab or cetuximab in metastatic colorectal cancer. Journal of clinical oncology 2008;26(35):5705–12.

11. Bardelli A, Corso S, Bertotti A, et al. Amplification of the MET receptor drives resistance to anti-EGFR therapies in colorectal cancer. Cancer Discovery 2013;3(6):658–73.

12. Neuss S, Becher E, Waltje M, Tietze L, Jahnen-Dechent W. Functional expression of hgf and hgf receptor/c-met in adult human mesenchymal stem cells suggests a role in cell mobilization, tissue repair, and wound healing. STEM CELLS [Internet] 22(3):405–14. Available from: https://stemcellsjournals.onlinelibrary.wiley.com/doi/abs/10.1634/stemcells.22-3-405

13. Yamamoto I, Nosho K, Kanno S, et al. EZH2 expression is a prognostic biomarker in patients with colorectal cancer treated with anti-EGFR therapeutics. Oncotarget 2017;8(11):1–9.

14. Mizukami T, Togashi Y, Naruki S, et al. Significance of FGF9 gene in resistance to anti-EGFR therapies targeting colorectal cancer: A subset of colorectal cancer patients with FGF9upregulation may be resistant to anti-EGFR therapies. Mol Carcinog 2016;56(1):106–17.

15. Bertotti A, Papp E, Jones S, et al. The genomic landscape of response to egfr blockade in colorectal cancer. Nature 2015;526(7572):263.

16. Guinney J, Dienstmann R, Wang X, et al. The consensus molecular subtypes of colorectal cancer. Nature medicine 2015;21(11):1350–6.

17. Rossum G van. Python website [Internet]. 1995;Available from: https://www.python.org

18. contributors P. Os module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/os.html

19. contributors P. Glob module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/glob.html

20. contributors P. Csv module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/csv.html

21. contributors P. Shutil module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/shutil.html

22. contributors P. Collections module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/collections.html

23. contributors P. Json module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/json.html

24. contributors P. Gzip module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/gzip.html

25. contributors P. Zlib module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/zlib.html

26. contributors P. Xml module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/xml.html

27. contributors P. Pickle module documentation [Internet]. 2018;Available from: https://docs.python.org/3/library/pickle.html

28. Pérez F, Granger BE. IPython: A system for interactive scientific computing. Computing in Science & Engineering 2007;9(3).

29. Ragan-Kelley M, Perez F, Granger B, et al. The jupyter/ipython architecture: A unified view of computational research, from interactive exploration to communication and publication. In: AGU fall meeting abstracts. 2014.

30. Walt S van der, Colbert SC, Varoquaux G. The numpy array: A structure for efficient numerical computation. Computing in Science & Engineering 2011;13(2):22–30.

31. contributors W. NumPy — wikipedia, the free encyclopedia [Internet]. 2018;Available from: https://en.wikipedia.org/w/index.php?title=NumPy&oldid=819079342

32. McKinney W, others. Data structures for statistical computing in python. In: Proceedings of the 9th python in science conference. SciPy Austin, TX; 2010. p. 51–6.

33. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in python. Journal of Machine Learning Research 2011;12(Oct):2825–30.

34. Maaten L van der, Hinton G. Visualizing data using t-sne. Journal of machine learning research 2008;9(Nov):2579–605.

35. Hunter JD. Matplotlib: A 2D graphics environment. Computing In Science & Engineering 2007;9(3):90–5.

36. Sievert C, Parmer C, Hocking T, et al. Plotly: Create interactive web graphics via plotly's javascript graphing library [software]. 2016;

37. Inc. A. Anaconda documentation [Internet]. 2018;Available from: https://docs.anaconda.com/anaconda/

38. Cox DR. The regression analysis of binary sequences. Journal of the Royal Statistical Society Series B (Methodological) [Internet] 1958;20(2):215–42. Available from: http://www.jstor.org/stable/2983890

39. Ho TK. Random decision forests. In: Document analysis and recognition, 1995., proceedings of the third international conference on. IEEE; 1995. p. 278–82.

40. Breiman L. Random forests. Machine learning 2001;45(1):5–32.

41. Biau G, Devroye L, Lugosi G. Consistency of random forests and other averaging classifiers. Journal of Machine Learning Research 2008;9(Sep):2015–33.

42. Denil M, Matheson D, De Freitas N. Narrowing the gap: Random forests in theory and in practice. In: International conference on machine learning. 2014. p. 665–73.

43. Oshiro TM, Perez PS, Baranauskas JA. How many trees in a random forest? In: International workshop on machine learning and data mining in pattern recognition. Springer; 2012. p. 154–68.

44. Commons NGD. GDC data portal user's guide. 1.11.0 ed. 2017.

45. Commons GD. Gdc-client [Internet]. 2018;Available from: https://gdc.cancer.gov/access-data/gdc-data-transfer-tool

46. Grossman RL, Heath AP, Ferretti V, et al. Toward a shared vision for cancer genomic data. New England Journal of Medicine 2016;375(12):1109–12.

47. Commons GD. Gdc data model components [Internet]. 2018;Available from: https://gdc.cancer.gov/developers/gdc-data-model/gdc-data-model-components

48. Commons NGD. GDC dna-seq somatic variation [Internet]. 2017. Available from: https://gdc.cancer.gov/about-data/data-harmonization-and-generation/genomic-data-harmonization/high-level-data-generation/dna-seq-somatic-variation

49. Krøigård AB, Thomassen M, Lænkholm A-V, Kruse TA, Larsen MJ. Evaluation of nine somatic variant callers for detection of somatic mutations in exome and targeted deep sequencing data. PLoS One 2016;11(3):e0151664.

50. Cibulskis K, Lawrence MS, Carter SL, et al. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. Nature biotechnology 2013;31(3):213.

51. NIH. XML - confluence wiki page [Internet]. 2013;Available from: https://wiki.nci.nih.gov/display/TCGA/XML

52. Stein M, Stein M. Baucheingeweide. 5., komplett überarb. Aufl. Marburg: Medi-Learn Verl; 2012.

53. Lee V, Murphy A, Le DT, Diaz LA. Mismatch repair deficiency and response to immune checkpoint blockade. The oncologist 2016;theoncologist–2016.

54. Seckendorff L. TCGA coad dissertation repository [Internet]. 2021;Available from: https://github.com/poddus/COAD

55. contributors. LogisticRegressionCV documentation [Internet]. 2018;Available from: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html

56. contributors. Random forest classifier documentation [Internet]. 2018;Available from: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

57. (HGNC) HGNC. The hgnc database [Internet]. Available from: URL: http://www.genenames.org/

58. Zerbino DR, Achuthan P, Akanni W, et al. Ensembl 2018. Nucleic Acids Research [Internet] 2018;46(D1):D754–61. Available from: http://dx.doi.org/10.1093/nar/gkx1098

59. Nomikos M, Kashir J, Lai FA. The role and mechanism of action of sperm plc-zeta in mammalian fertilisation. Biochemical Journal [Internet] 2017;474(21):3659–73. Available from: http://www.biochemj.org/content/474/21/3659

60. Hu W, Yang Y, Li X, et al. Multi-omics approach reveals distinct differences in left-and right-sided colon cancer. Molecular Cancer Research 2017;molcanres–0483.

61. Bauer KM, Hummon AB, Buechler S. Right-side and left-side colon cancer follow different pathways to relapse. Molecular carcinogenesis 2012;51(5):411–21.

62. Gupta RK, Towers GJ. A tail of tetherin: How pandemic hiv-1 conquered the world. Cell host & microbe 2009;6(5):393–5.

63. Mukai S, Oue N, Oshima T, et al. Overexpression of transmembrane protein bst2 is associated with poor survival of patients with esophageal, gastric, or colorectal cancer. Annals of Surgical Oncology [Internet] 2017;24(2):594–602. Available from: https://doi.org/10.1245/s10434-016-5100-z

64. F. DRP, J. SW, D. DL, K. H, VAN TA. Effect of adiposity on plasma lipid transfer protein activities: A possible link between insulin resistance and high density lipoprotein metabolism. European Journal of Clinical Investigation [Internet] 24(3):188–94. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2362.1994.tb00987.x

65. Su F, Grijalva V, Navab K, et al. HDL mimetics inhibit tumor development in both induced and spontaneous mouse models of colon cancer. Molecular Cancer Therapeutics [Internet] 2012;11(6):1311–9. Available from: http://mct.aacrjournals.org/content/11/6/1311

66. Fagerberg L, Hallström BM, Oksvold P, et al. Analysis of the human tissue-specific expression by genome-wide integration of transcriptomics and antibody-based proteomics. Molecular & Cellular Proteomics [Internet] 2014;13(2):397–406. Available from: http://www.mcponline.org/content/13/2/397.abstract

67. Jung C, Kim R, Zhang H, et al. HOXB13 is downregulated in colorectal cancer to confer tcf4-mediated transactivation. British journal of cancer 2005;92(12):2233.

68. Vider BZ, Zimber A, Hirsch D, et al. Human colorectal carcinogenesis is associated with deregulation of homeobox gene expression. Biochemical and biophysical research communications 1997;232(3):742–8.

69. Nagel S, Burek C, Venturini L, et al. Comprehensive analysis of homeobox genes in hodgkin lymphoma cell lines identifies dysregulated expression of hoxb9 mediated via erk5 signaling and bmi1. Blood 2007;109(7):3015–23.

70. Sakiyama J-i, Yokouchi Y, Kuroiwa A. HoxA and hoxb cluster genes subdivide the digestive tract into morphological domains during chick develop-

ment. Mechanisms of Development [Internet] 2001;101(1):233–6. Available from: http://www.sciencedirect.com/science/article/pii/S0925477300005645

71. Abate-Shen C. Deregulated homeobox gene expression in cancer: Cause or consequence? Nature Reviews Cancer 2002;2(10):777.

72. Akiyama SK, Olden K, Yamada KM. Fibronectin and integrins in invasion and metastasis. Cancer and Metastasis Reviews [Internet] 1995;14(3):173–89. Available from: https://doi.org/10.1007/BF00690290

73. Yang R-Z, Lee M-J, Hu H, et al. Identification of omentin as a novel depot-specific adipokine in human adipose tissue: Possible role in modulating insulin action. American Journal of Physiology-Endocrinology and Metabolism [Internet] 2006;290(6):E1253–61. Available from: https://doi.org/10.1152/ajpendo.00572.2004

74. Berasain C, Avila MA. Amphiregulin. Seminars in Cell & Developmental Biology [Internet] 2014;28:31–41. Available from: http://www.sciencedirect.com/science/article/pii/S1084952114000068

75. Deng M, Brägelmann J, Kryukov I, Saraiva-Agostinho N, Perner S. FirebrowseR: An r client to the broad institute's firehose pipeline. Database 2017;2017.

76. Kosinski M, Biecek P, Kosinski MM. Package "rtcga". 2016;Available from: https://github.com/RTCGA/RTCGA