Machine Learning Engineer Nanodegree

Capstone Project

Ollie Graham
September 17th, 2017

1. Definition

1.1 Project Overview

A lot has been said during the past several years about how precision medicine and, more concretely, how genetic testing is going to disrupt the way diseases like cancer are treated.

But this is only partially happening due to the huge amount of manual work still required. Memorial Sloan Kettering Cancer Center (MSKCC) launched this competition, accepted by the NIPS 2017 Competition Track, because help is needed to take personalized medicine to its full potential.

Once sequenced, a cancer tumor can have thousands of genetic mutations. But the challenge is distinguishing the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers).

Currently this interpretation of genetic mutations is being done manually. This is a very time-consuming task where a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature.

References:

Classifying Clinically Actionable Genetic Mutations (https://www.kaggle.com/c/msk-redefining-cancer-treatment)
OncoKB (http://oncokb.org/)
Wikipedia: Mutations (https://en.wikipedia.org/wiki/Mutation)

1.2 Problem Statement

For this Kaggle competition MSKCC is making available an expert-annotated knowledge base where world-class researchers and oncologists have manually annotated thousands of mutations.

The aim of this project is to help develop a Machine Learning algorithm that, using this knowledge base as a baseline, automatically classifies genetic variations.

The solution required the use of a supervised classification algorithm, of which there are many options, each with its own particular best use cases. I am particularly interested in applying a deep learning algorithm to this problem, using the Keras Classifier model which acts as a wrapper around the Scikit learn library.

1.2.1 Deep Learning

Deep learning is a class of machine learning algorithms that:

- use a cascade of many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The algorithms may be supervised or unsupervised and applications include pattern analysis (unsupervised) and classification (supervised).
- are based on the (unsupervised) learning of multiple levels of features or representations of the data. Higher level features are derived from lower level features to form a hierarchical representation.
- are part of the broader machine learning field of learning representations of data.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.
- use some form of gradient descent for training

Deep learning algorithms are very powerful and when correctly modelled, calibrated and trained outperform many other type of machine learning algorithms in terms of their accuracy.

I used a supervised application of deep learning which means that the model was trained on a training set of data with labels on which it tried to fit that model to the data in order to predict those labels as best as possible.

1.2.2 Artificial Neural Networks

Artificial neural networks (ANNs) or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the analytic results to identify cats in other images. They have found most use in applications difficult to express in a traditional computer algorithm using rule-based programming.

An ANN is based on a collection of connected units called artificial neurons, (analogous to axons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. Further, they may have a threshold such that only if the aggregate signal is below (or above) that level is the downstream signal sent.

Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

A deep neural network (DNN) is an ANN with multiple hidden layers between the input and output layers. Similar to shallow ANNs, DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network.

1.2.2.1 Application to Natural Language Processing

Neural networks have been used for implementing language models since the early 2000s. Recurrent neural networks, especially Long Term Short Memory (LSTM), are most appropriate for sequential data such as language. LSTM helped to improve machine translation and language modeling.

Other key techniques in this field are negative sampling and word embedding. Word embedding, such as word2vec, can be thought of as a representational layer in a deep learning architecture that transforms an atomic word into a positional representation of the word relative to other words in the dataset; the position is represented as a point in a vector space. Using word embedding as an RNN input layer allows the network to parse sentences and phrases using an effective compositional vector grammar. A compositional vector grammar can be thought of as probabilistic context free grammar (PCFG) implemented by an RNN. Recursive auto-encoders built atop word embeddings can assess sentence similarity and detect paraphrasing. Deep neural architectures have achieved state-of-the-art results in natural language processing tasks such as constituency parsing, sentiment analysis, information retrieval, spoken language understanding, machine translation, contextual entity linking, writing style recognition and others.

Natural language processing using tools such as Word2Vec and Doc2Vec are particularly applicable to this classification task, given the need to classify records using a knowledge base composed of text.

In addition to these natural language tools the supervised learning classification task can also be facilitated by the use of a neural network. As mentioned above, I used the Keras library for this task and will discuss it in detail below.

Reference: Deep Learning - Wikipedia (https://en.wikipedia.org/wiki/Deep_learning)

1.3 Evaluation Metrics

The accuracy of the predictions in the competition shall be measured by the multiclass log loss (MLL).

Multi-class log loss is the multi class version of the Logarithmic Loss metric. Each observation is in one class and for each observation, I need to submit a predicted probability for each class. The metric is negative the log likelihood of the model that says each test observation is chosen independently from a distribution that places the submitted probability mass on the corresponding class, for each observation.

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

where N is the number of observations, M is the number of class labels, *log* is the natural logarithm, *yij*, is 1 if observation *i* is in class *j* and 0 otherwise, and *pij* is the predicted probability that observation *i* is in class *j*.

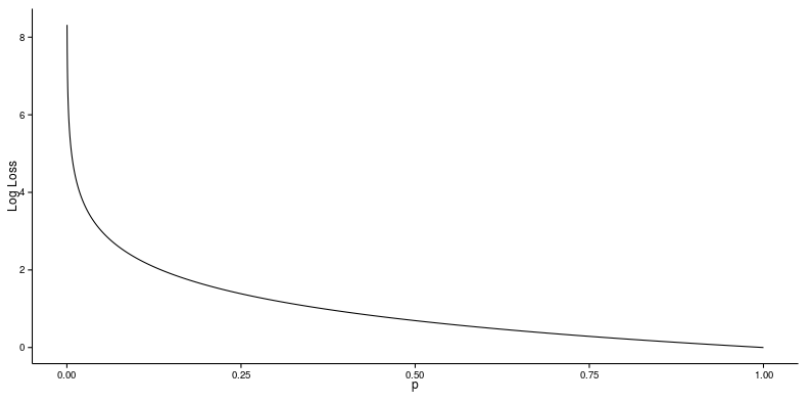
Both the solution file and the submission file are CSV's where each row corresponds to one observation, and each column corresponds to a class. The solution has 1's and 0's (exactly one "1" in each row), while the submission consists of predicted probabilities.

The submitted probabilities need not sum to 1, because they will be rescaled (each is divided by the sum) so that they do before evaluation.

1.3.1 Why is Log Loss an appropriate metric?

Log Loss quantifies the accuracy of a classifier by penalising false classifications. Minimising the Log Loss is basically equivalent to maximising the accuracy of the classifier.

In order to calculate Log Loss the classifier must assign a probability to each class rather than simply yielding the most likely class.



Log Loss heavily penalises classifiers that are confident about an incorrect classification. For example, if for a particular observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large indeed. This is going to have a significant impact on the overall Log Loss for the classifier.

Source: Log Loss Explainer (http://www.exegetic.biz/blog/2015/12/making-sense-logarithmic-loss/)

2. Analysis

2.1 Datasets and Inputs

In this project I will develop an algorithm to classify genetic mutations based on clinical evidence (text).

There are nine different classes a genetic mutation can be classified on.

This is not a trivial task since interpreting clinical evidence is very challenging even for human specialists. Therefore, modeling the clinical evidence (text) will be critical for the success of the approach.

Both, training and test, data sets are provided via two different files. One (training/test_variants) provides the information about the genetic mutations, whereas the other (training/test_text) provides the clinical evidence (text) that our human experts used to classify the genetic mutations. Both are linked via the ID field.

Therefore the genetic mutation (row) with ID=15 in the file training_variants, was classified using the clinical evidence (text) from the row with ID=15 in the file training_text.

Finally, to make it more exciting!! Some of the test data is machine-generated to prevent hand labeling.

2.1.1 File descriptions

training_variants - a comma separated file containing the description of the genetic mutations used for training. Fields are ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the aminoacid change for this mutations), Class (1-9 the class this genetic mutation has been classified on)

training_text - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)

test_variants - a comma separated file containing the description of the genetic mutations used for training. Fields are ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the aminoacid change for this mutations)

test_text - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)

submissionSample - a sample submission file in the correct format

2.2 Data Exploration

2.2.1 Overview

I'll begin by inspecting the format of the provided data.

train_variant

train_variant shape = 3,321 rows x 4 columns

	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4

test_variant holds the same format, but without the 'Class' field that we are attempting to identify.

test_variant shape = 5,668 rows x 4 columns

train_text

train_text shape = 3,321 rows, 4 columns

	ID	Text
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

Again the test_text file holds the same format of data.

test_text shape = 5,668 rows x 2 columns

The ID field is common to both the ..variant and ..text files and refers to the same record, which allows me to join them using pandas into a single dataframe that I can use for analysis.

It's also important to note that the test data is 70% larger than the training data.

2.2.2 Combined Data Exploration

This table describes the content of both training and test data combined:

	ID	Gene	Variation	Text
count	8989	8989	8989	8989
unique	5668	1507	8609	7390
top	2833	BRCA1	Truncating Mutations	The PTEN (phosphatase and tensin homolog) phos...
freq	2	293	111	59

In summary there are:

- 1,507 unique Genes
- 8,609 unique Variants
- 7,309 unique Text items

This indicates that the variant and complete text body as a whole for each record in the data might not hold much explanatory information about the data on their own, due to the extent of the variability in this data.

The fewer number of unqiue genes is more promising however.

2.2.3 Gene Feature

There are 264 unique genes in the test data and 1,397 unique genes in the test data.

Here are the top 10 genes in the both the training and test data:

Train Top 10 Genes

Gene	Count
BRCA1	264
TP53	163
EGFR	141
PTEN	126
BRCA2	125
KIT	99
BRAF	93
ERBB2	69
ALK	69
PDGFRA	60

Test Top 10 Genes

Gene	Count
F8	134
CFTR	57
F9	54
G6PD	46
GBA	39
PAH	38
AR	38
CASR	37
ARSA	30
BRCA1	29

The top 10 genes for training data are very different to that of the testing data, lowering the likelihood that this feature alone will be a good predictor of the test data.

It is also worth noting that there are 154 genes in common across the training and test data.

2.2.4 Variant Feature

Here are the top 10 variants in the both the training and test data:

Train Top 10 Variants

Variation	Count
Truncating Mutations	93
Deletion	74
Amplification	71
Fusions	34
Overexpression	6
G12V	4
Q61R	3
Q61L	3
Q61H	3
E17K	3

Test Top 10 Variants

Variation	Count
Truncating Mutations	18
Deletion	14
Amplification	8
Fusions	3
G44D	2
A1038V	1
A1035V	1
A1028V	1
A1020P	1
A101V	1

The top 4 variants in both training and testing data are the same, so this feature might be able to yield some predictive power.

2.2.5 Text Feature

Let's inspect a single text record:

train_text[text][0]

"Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development....

This continues for another 38,000 characters, so for brevity in this report I have cut it short.

Ok... That's a LOT of text in one record. I will investigate this feature further in the visualisation section below.

2.2.6 Data Distribution

Finally, lets take a look at the distribution of training data across the classes:

Class	Percent of Total
1	17.10
2	13.61
3	2.68
4	20.66
5	7.29
6	8.28
7	28.70
8	0.57
9	1.11

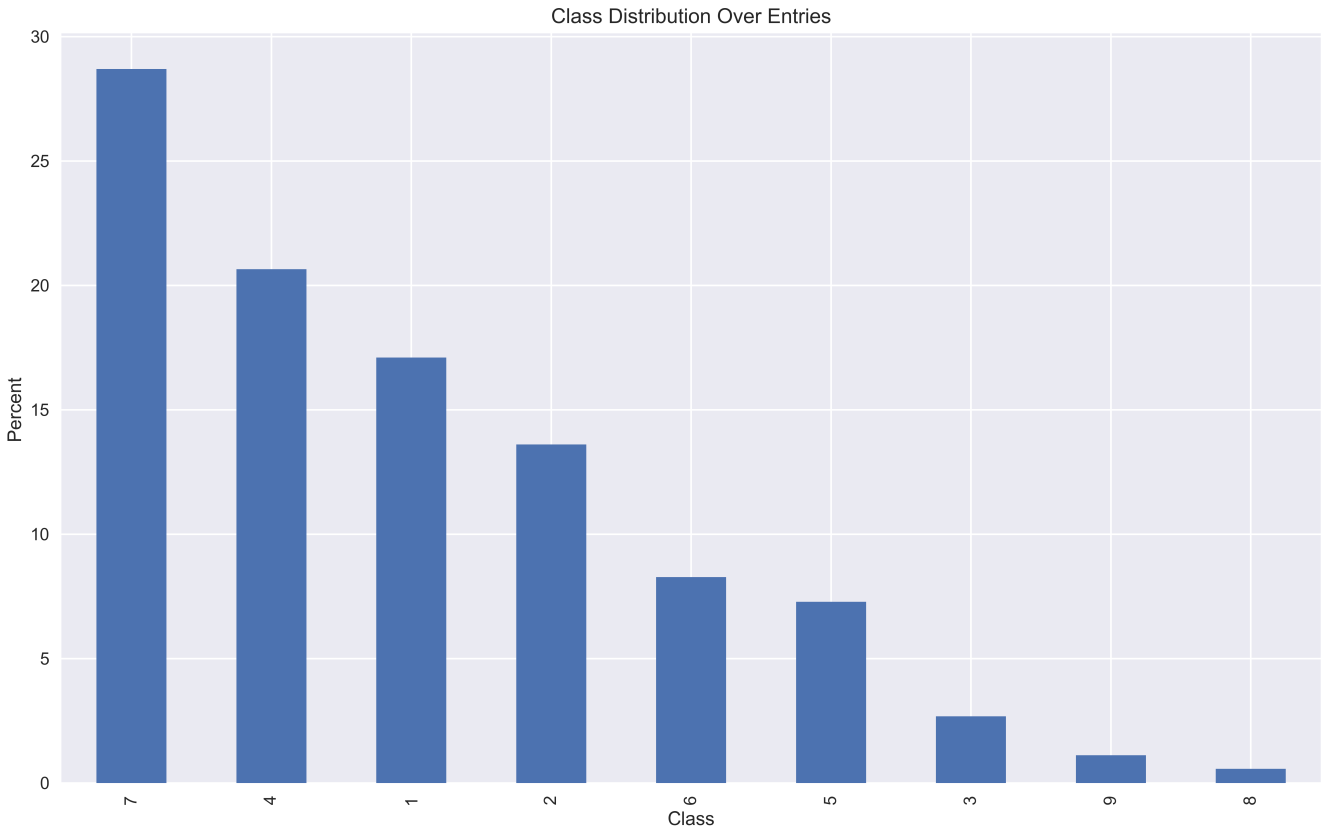
Class 7 is outweighted in the training data, containing nearly 30% of the records. Classes 7, 4 & 1 cumulatively contain ~66.5% of the training data which might make it difficult for our model to learn effectively on the remaining data in order to predict classes outside of this group.

This distribution can be used to inform our naive predictor, as discussed in the benchmark section below.

2.3 Exploratory Visualisation

2.3.1 Data Overview

Let's being with where we left off in the data exploration, with a look at the distribution of data across the classes:



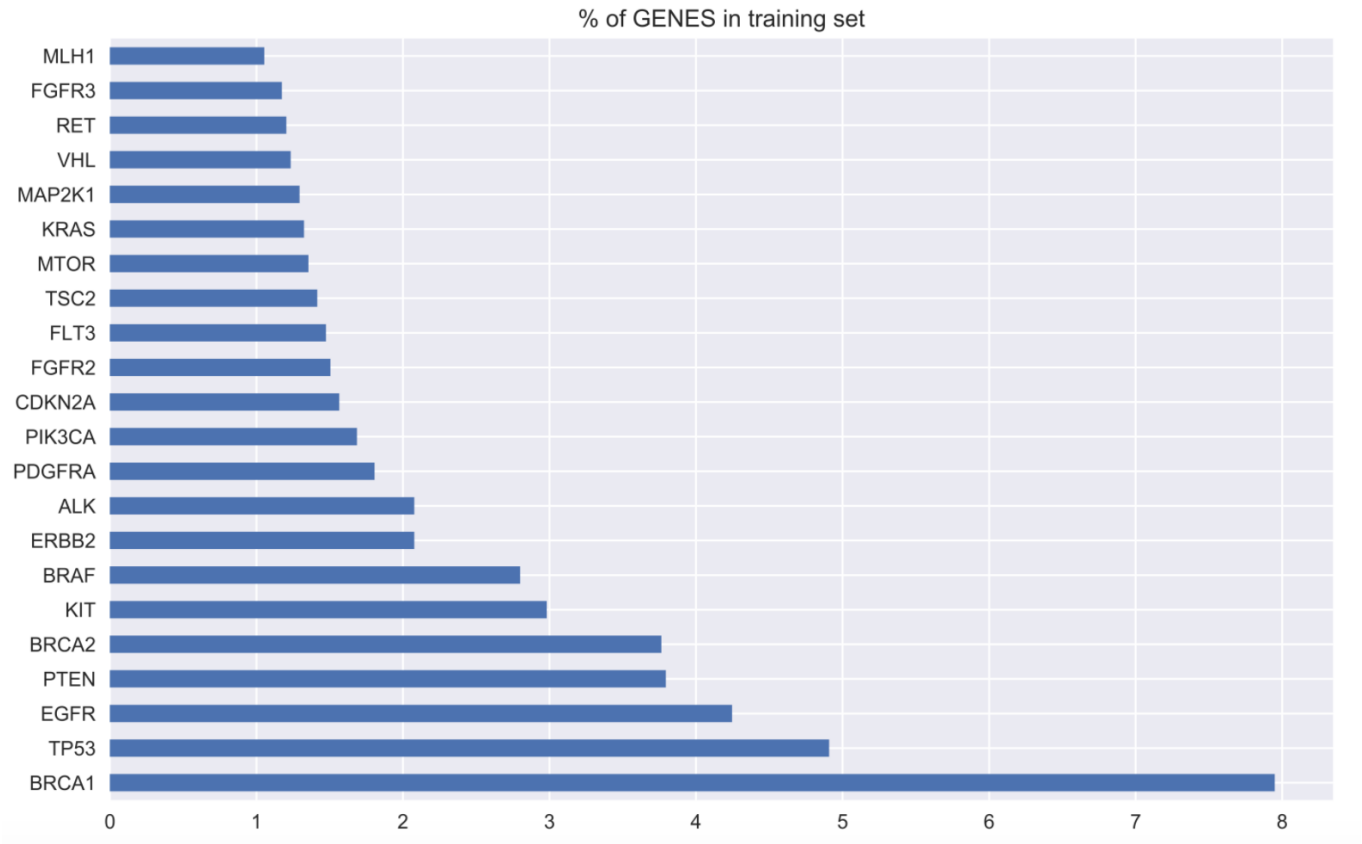
As described above the classes with the highest proportional representation in the training data are 7, 4 & 1.

Classes 3, 8 & 9 are very underrepresented in the data.

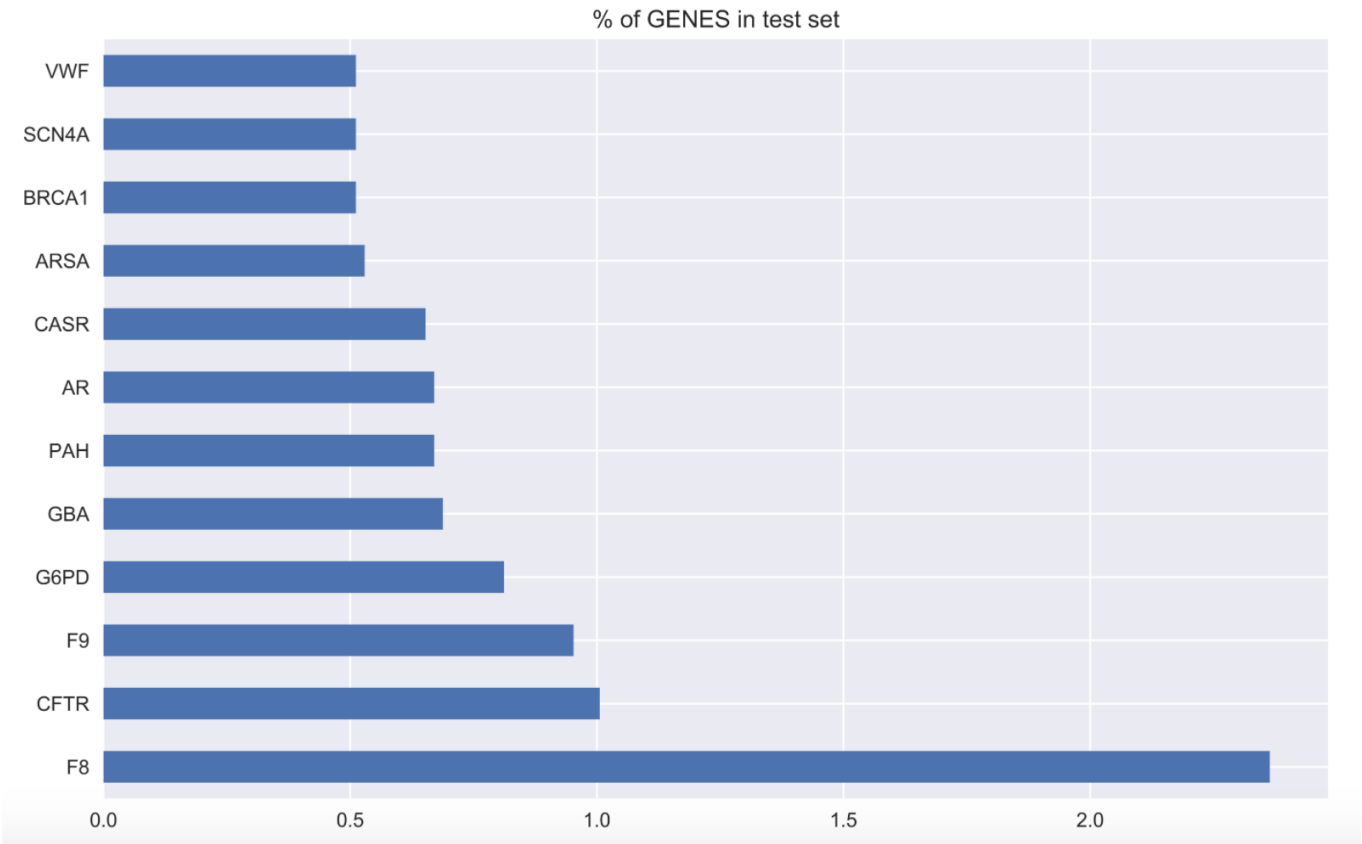
2.3.2 Gene Feature

2.3.2.1 Overall

This chart shows the relative proportion of the most populus genes in the training data:



By comparison here are the relative proportions of genes in the test data:



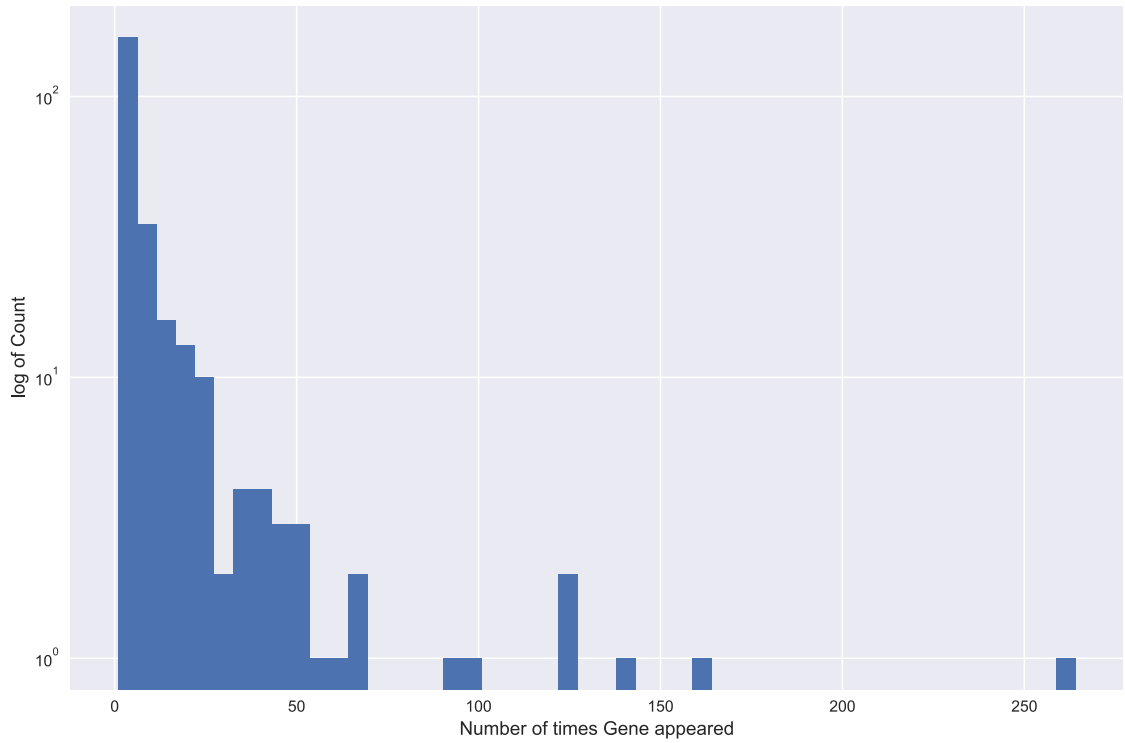
The test data appears to be far less dominated by a few genes, which backs up the statistic of number of unique genes in each data set (~8% of the training data and 25% of the test data).

The following table and chart show the frequency of appearances of a gene in the training data:

Genes that appear less than 2 times: 26.89%
Genes that appear less than 5 times: 53.03%
Genes that appear less than 10 times: 71.59%
Genes that appear less than 20 times: 82.95%
Genes that appear less than 50 times: 95.08%
Genes that appear less than 100 times: 98.11%
Genes that appear less than 300 times: 100.0%

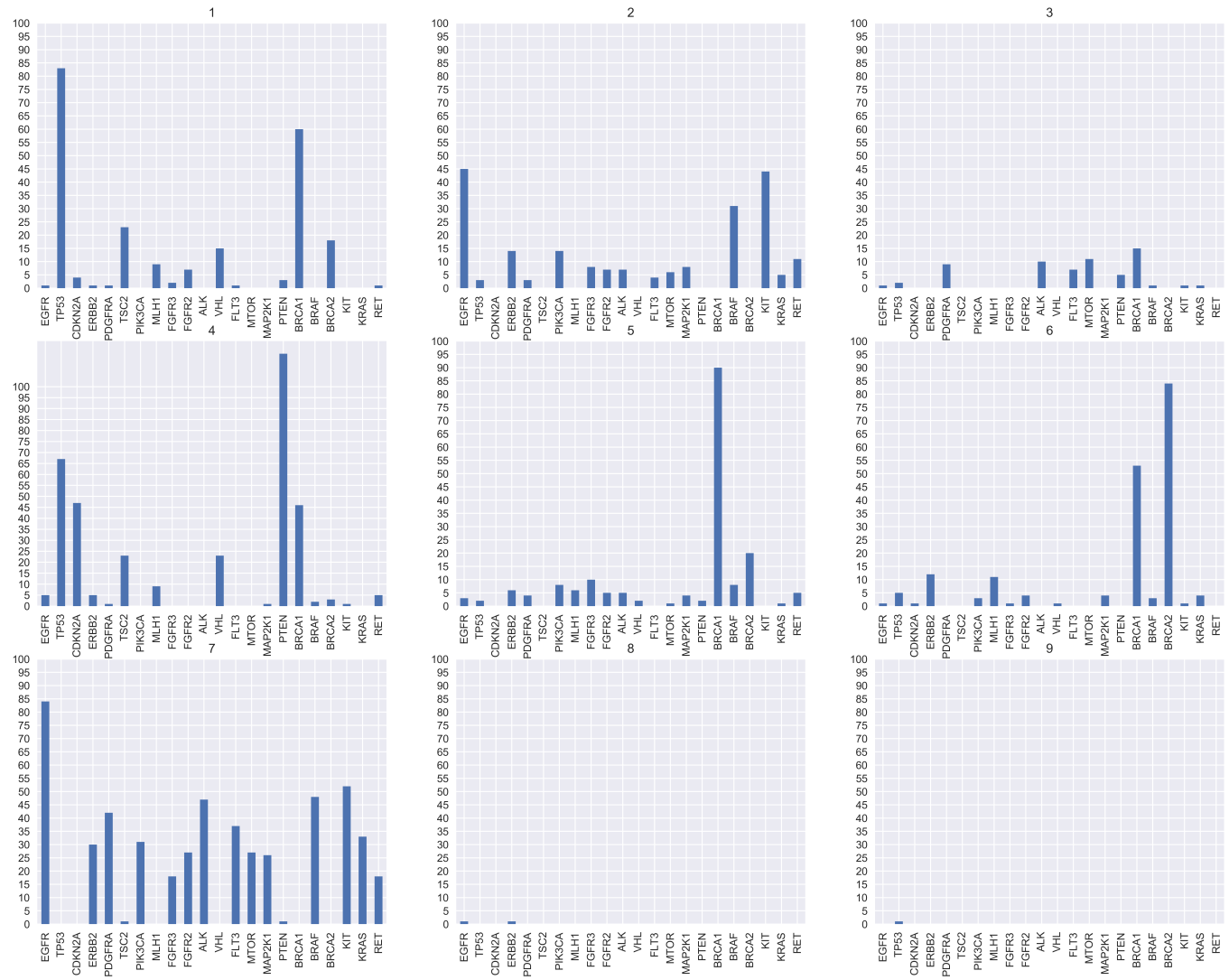
~70% of the Genes don't appear more than 10 times and 95% (250 out of 264) don't appear more than 50 times.

This distribution is shown in the chart below:



2.3.2.2 Genes by Class

The following charts show the relative proportion of the most populus genes by class (each class is a subplot) in the training data.



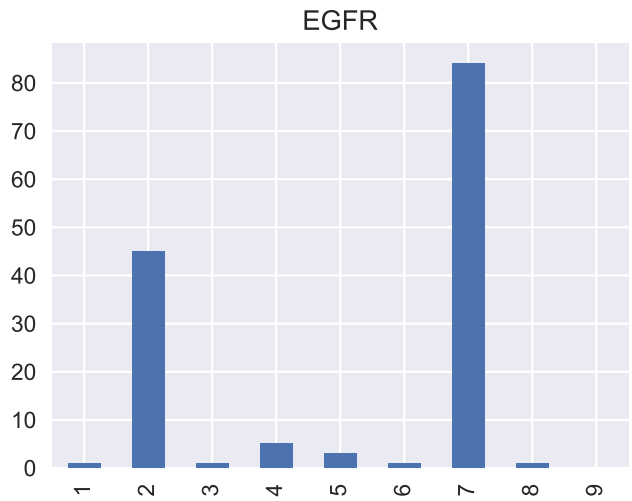
Class	Gene(s) with Higest Proportion
1	TP53, BRCA1
2	EGFR, BRAF, KIT
3	BRCA1, MTOR, ALK, PDGFRA
4	PTEN, TP53, CDKN2A, BRCA1
5	BRCA1, BRCA2
6	BRCA1, BRCA2
7	EGFR, ERBB2, PDGFRA, PIK3CA, ALK, FLT3, BRAF, KIT
8	EGFR, ERBB2
9	TP53

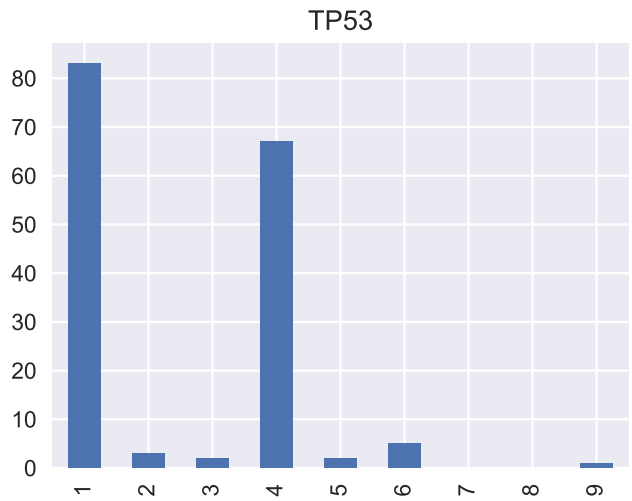
Note: The classes with clear leaders in terms of gene representation are: 1, 2, 4, 5, 6 & 7. Classes 3, 8 & 9 show little dominance with respect to a single gene. This could be becuase the type of mutations seen in those classes is common to many genes.

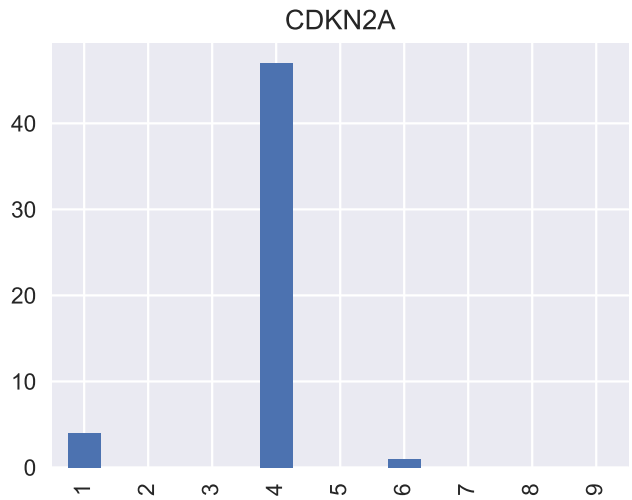
This also indicates that using Gene as a predictive feature for some classes might not yield much information.

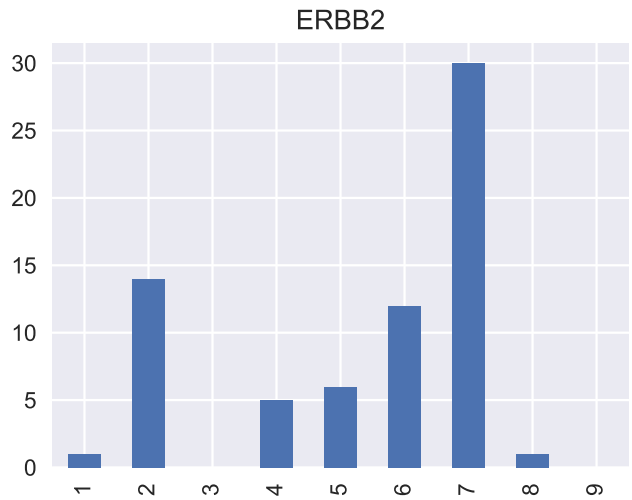
2.3.2.3 Most Common Genes

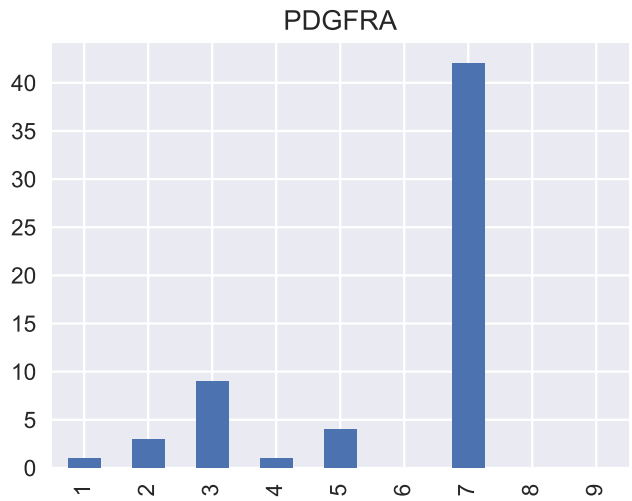
Now, lets inspect the class distribution for the most common genes in the training data.

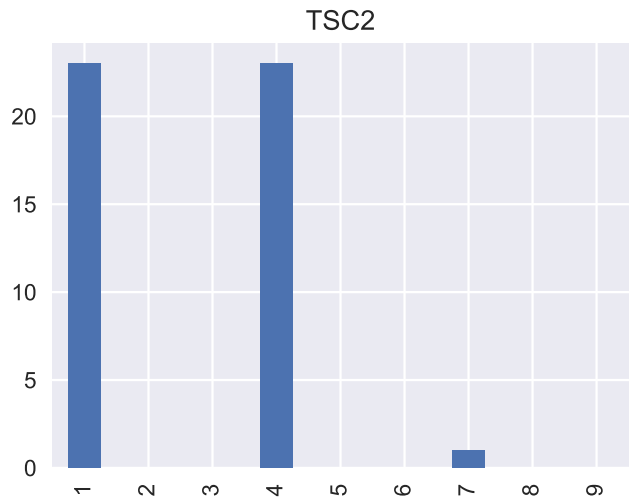

EGFR
0
10
20
30
40
50
60
70
80
1
2
3
4
5
6
7
8
9

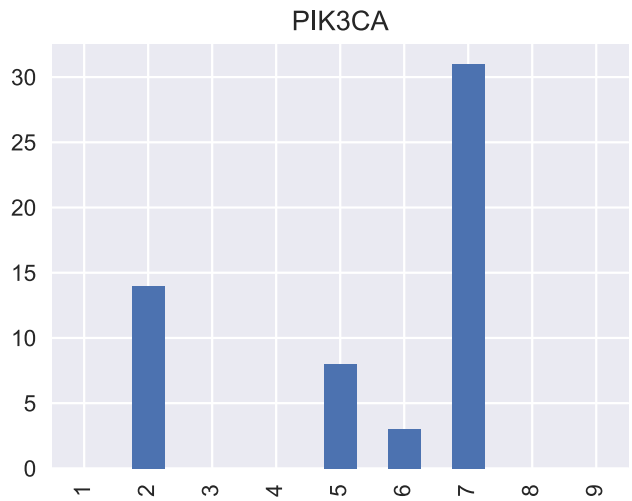

TP53
0
10
20
30
40
50
60
70
80
1
2
3
4
5
6
7
8
9

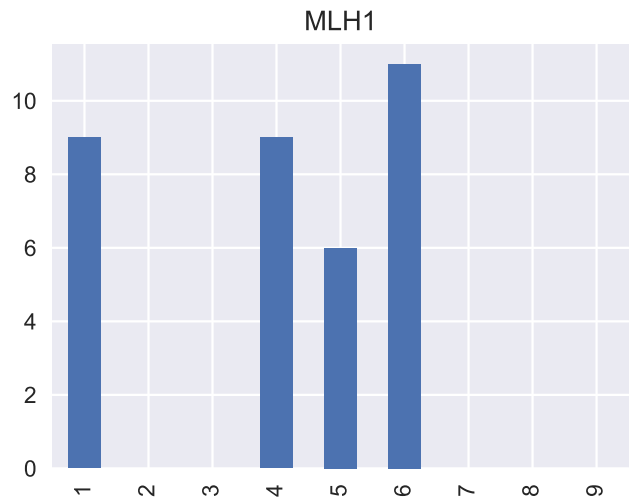

CDKN2A
0
10
20
30
40
1
2
3
4
5
6
7
8
9

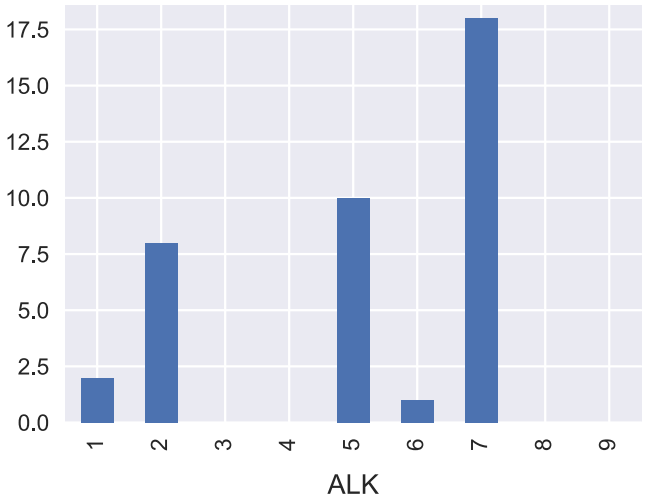

ERBB2
0
5
10
15
20
25
30
1
2
3
4
5
6
7
8
9

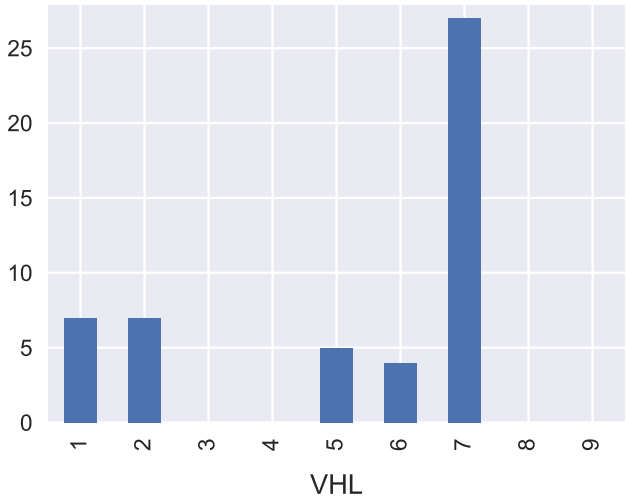

PDGFRA
0
5
10
15
20
25
30
35
40
1
2
3
4
5
6
7
8
9

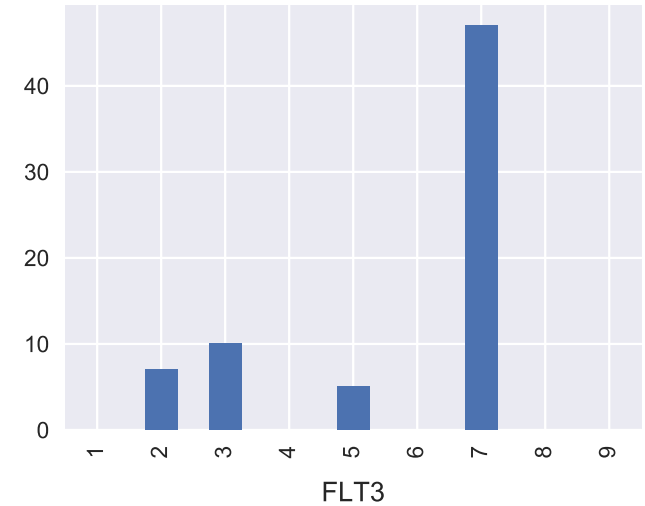

TSC2
0
5
10
15
20
1
2
3
4
5
6
7
8
9

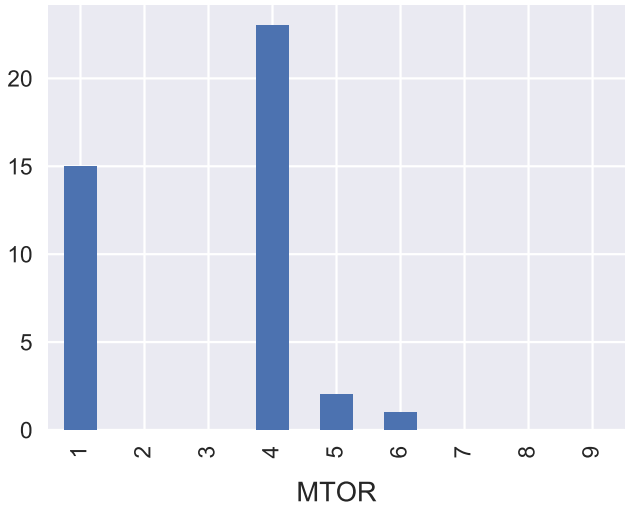

PIK3CA
0
5
10
15
20
25
30
1
2
3
4
5
6
7
8
9
FGFR3

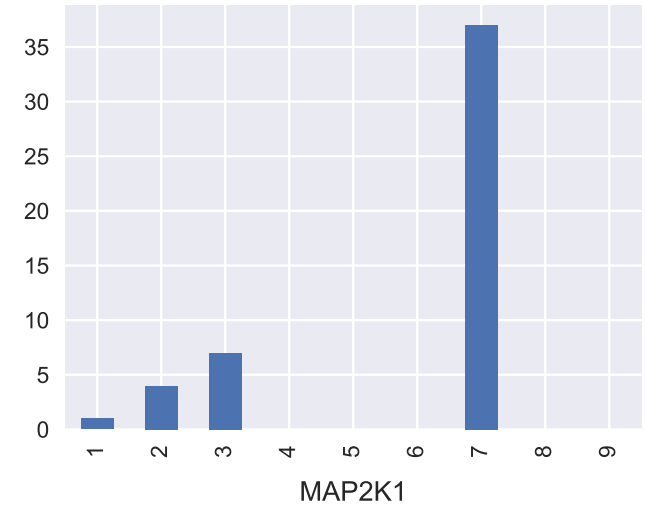

MLH1
0
2
4
6
8
10
1
2
3
4
5
6
7
8
9
FGFR2

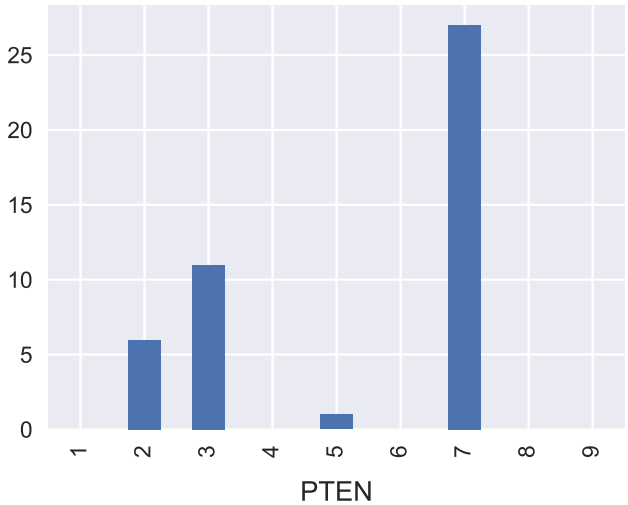

17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
1 2 3 4 5 6 7 8 9
ALK

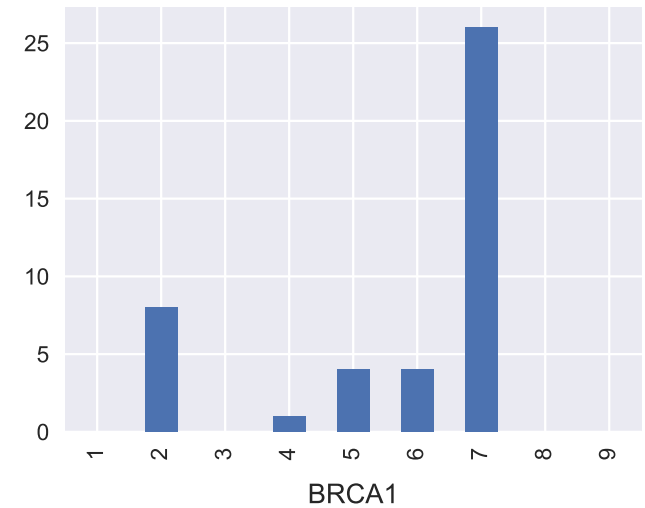

25
20
15
10
5
0
1 2 3 4 5 6 7 8 9
VHL

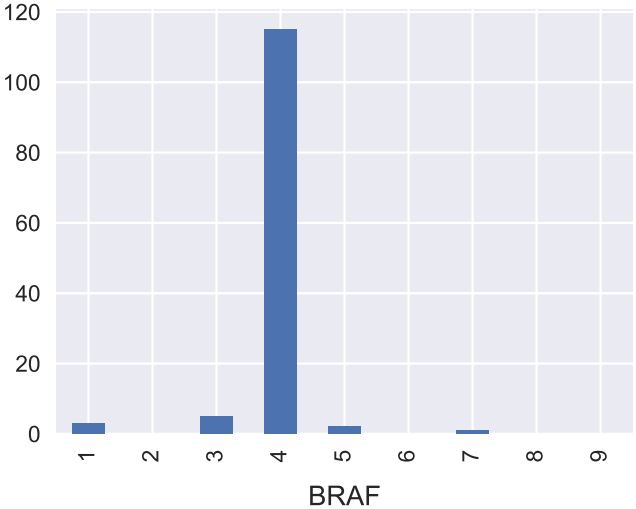

40
30
20
10
0
1 2 3 4 5 6 7 8 9
FLT3


20
15
10
5
0
1 2 3 4 5 6 7 8 9
MTOR

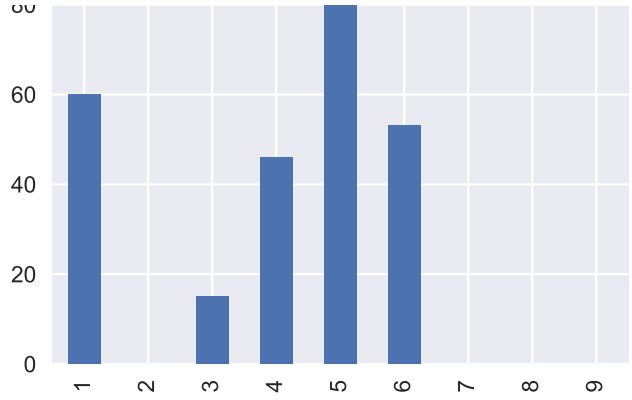

35
30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9
MAP2K1

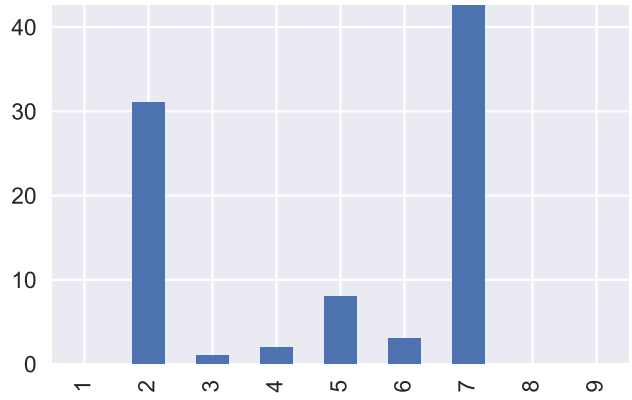

25
20
15
10
5
0
1 2 3 4 5 6 7 8 9
PTEN


25
20
15
10
5
0
1 2 3 4 5 6 7 8 9
BRCA1

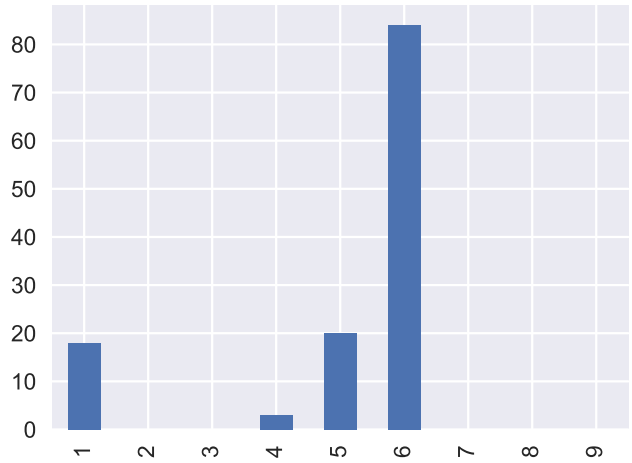

120
100
80
60
40
20
0
1 2 3 4 5 6 7 8 9
BRAF


50

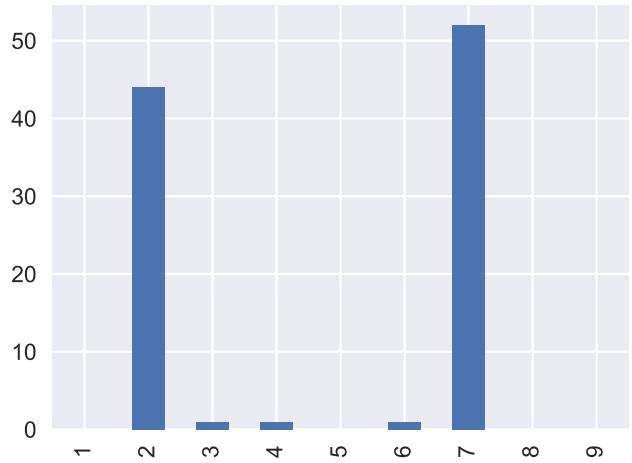




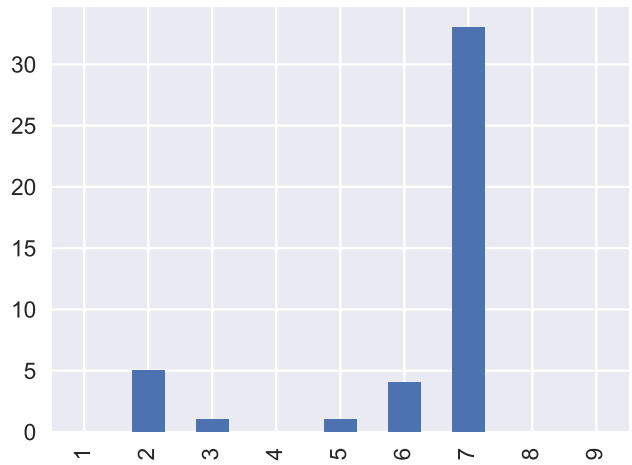
BRCA2



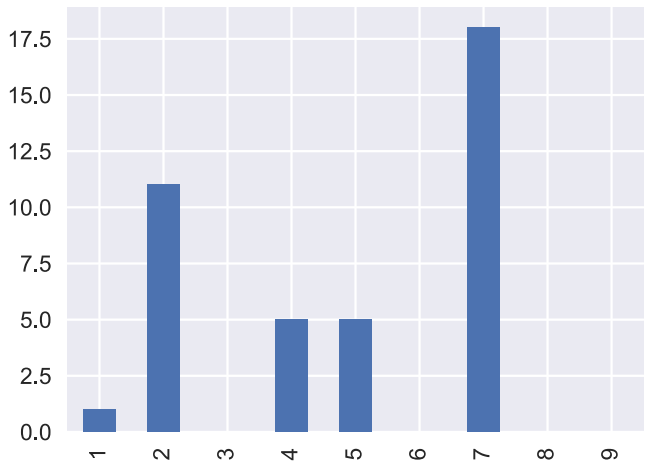
KIT



KRAS



RET



Representation of each of the genes by class where the gene is most prevalent (Count is > 10) is shown in the table the following table:

Gene	Classes	Gene	Classes
EGFR	2, 7	VHL	1, 4
TP53	1, 4	FLT3	7
CDKN2A	4	MTOR	3, 7
ERBB2	2, 6, 7	MAP2K1	7
PDGFRA	7	PTEN	4
TSC2	1, 4	BRCA1	1, 3, 4, 5, 6
PIK3CA	3, 2	BRAF	2, 7
MLH1	6	BRCA2	1, 5, 6
FGFR3	5, 7	KIT	2, 7
FGFR2	7	KRAS	7
ALK	7	RET	2, 7

Worth noting here is that class 7 is well represented across a number of genes and classes 3, 8, 9 are underrepresented as we observed previously.

2.3.2.4 Overall Summary

Overall, although its clear that certain genes in the training data are closely linked to certain classes of mutation, it seems as though the predictive power of this feature outside of the training data will be limited due to the completely different set of genes present in the test data.

2.3.3 Text Feature

2.3.3.1 Overall

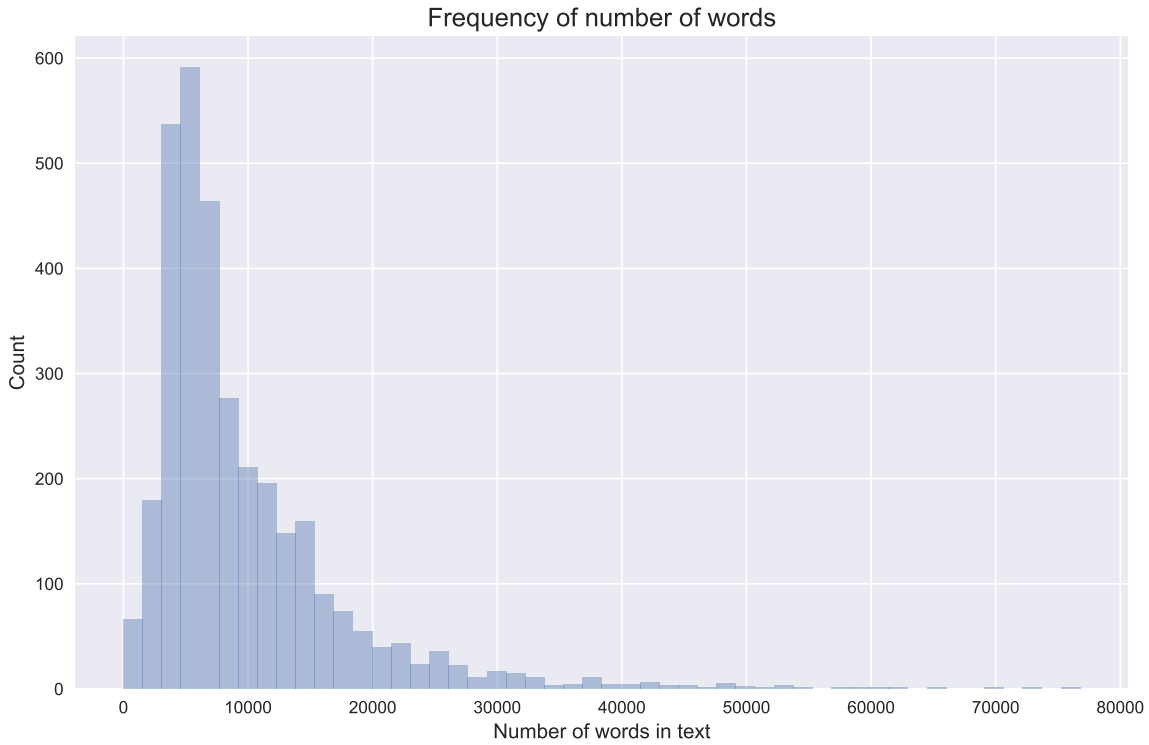
The text feature holds the most promise of any feature discussed thus far and seems to be the intended target of the proposers of this challenge.

It contains a huge corpus of text across both the training and test data as I will attempt to visualise below.

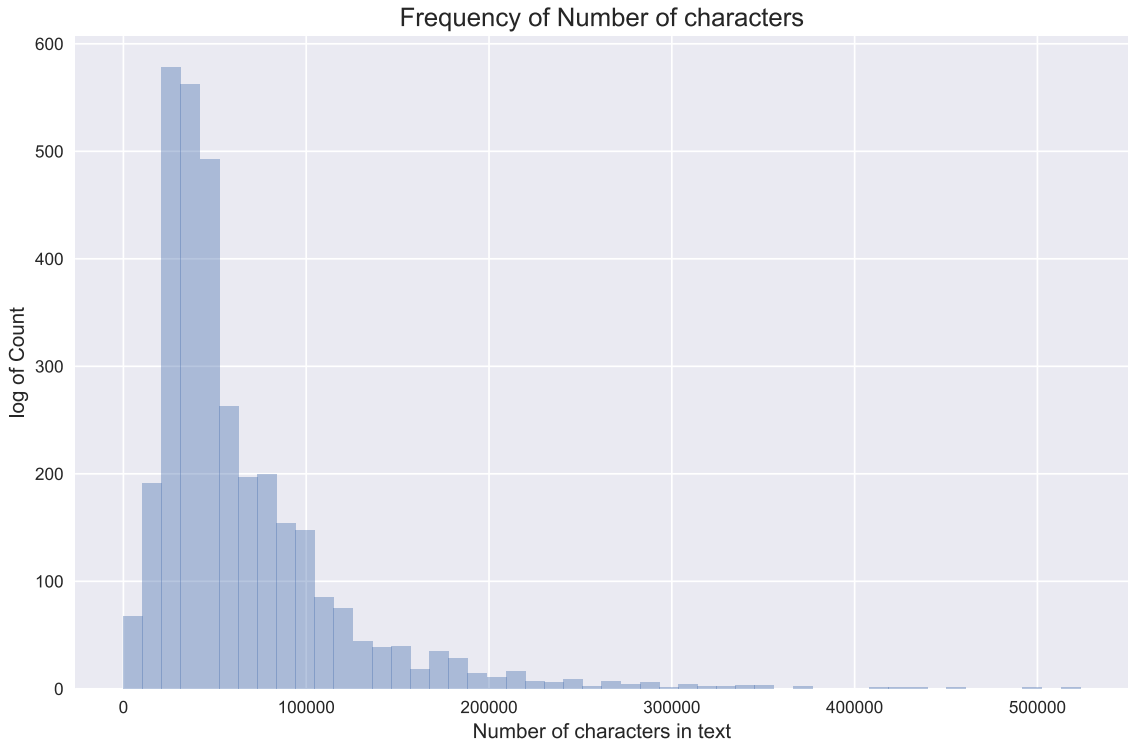
Overall across training and test data the number of words are distributed similarly, as follows:

Train		Test	
	Count		Count
count	3321.0	count	5668.0
mean	9551.0	mean	8473.0
std	7849.0	std	3555.0
min	1.0	min	1.0
25%	4733.0	25%	6382.0
50%	6871.0	50%	8346.0
75%	11996.0	75%	10213.0
max	76782.0	max	73607.0

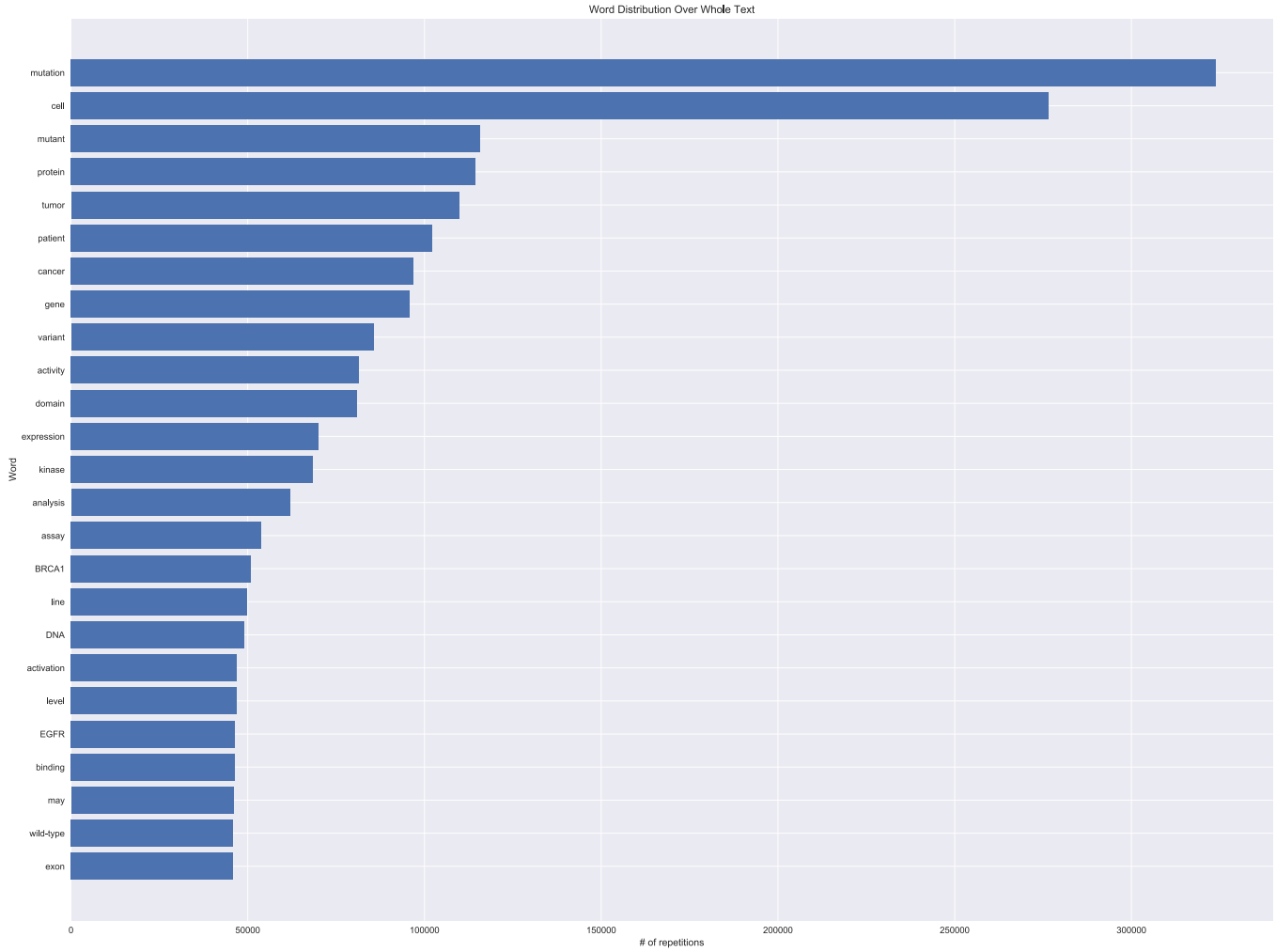
The count of number of words in the training data is skewed right, with the vast majority of records with a word count of < 10,000.



Similarly, so is the count of the number of characters, which is to be expected given the chart above.



Finally lets inspect the distribution of words over the training data set:



I am not surprised to see these cancer related terms being the most abundant across the text. Without such terms it would not be possible to describe the mutations, or classify them. These features should prove useful for classification if they can be related to specific classes.

2.3.3.2 Text by Class

Let's begin by looking at the statistics of the number of words in the training data by class:

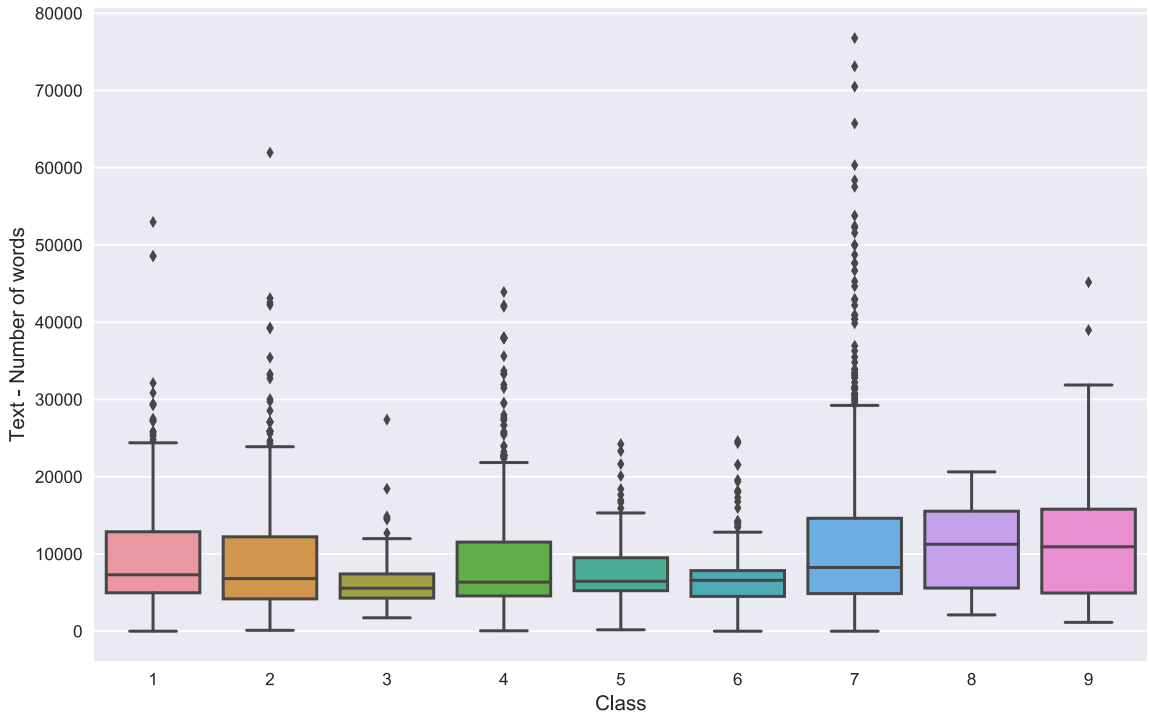
Class	count	mean	std	min	25%	50%	75%	max
1	568.0	9450.299296	6516.412101	1.0	4973.0	7305.0	12873.5	52972.0
2	452.0	9310.393805	7627.288722	116.0	4185.0	6810.0	12220.0	61957.0
3	89.0	6757.382022	3725.366918	1737.0	4283.0	5572.0	7415.0	27391.0
4	686.0	8983.390671	7280.220754	53.0	4566.0	6351.0	11537.0	43913.0
5	242.0	7517.049587	3902.868040	183.0	5245.0	6463.0	9513.5	24226.0
6	275.0	7184.120000	3836.912865	1.0	4505.5	6587.0	7847.0	24609.0
7	953.0	11442.867786	10111.940846	1.0	4871.0	8254.0	14620.0	76782.0
8	19.0	10814.421053	5648.714095	2111.0	5586.0	11253.0	15535.0	20626.0
9	37.0	12807.459459	10217.093429	1147.0	4942.0	10930.0	15797.0	45177.0

The number of words in the text used to describe a class of mutation varies greatly. The smallest number of words used is for class 6 (24,609) and the largest for class 7 (76,782). This suggests that some classes need much more evidence than others to classify correctly.

The mean number of words for a class range from 6,757 for class 3 to 12,807 for class 9.

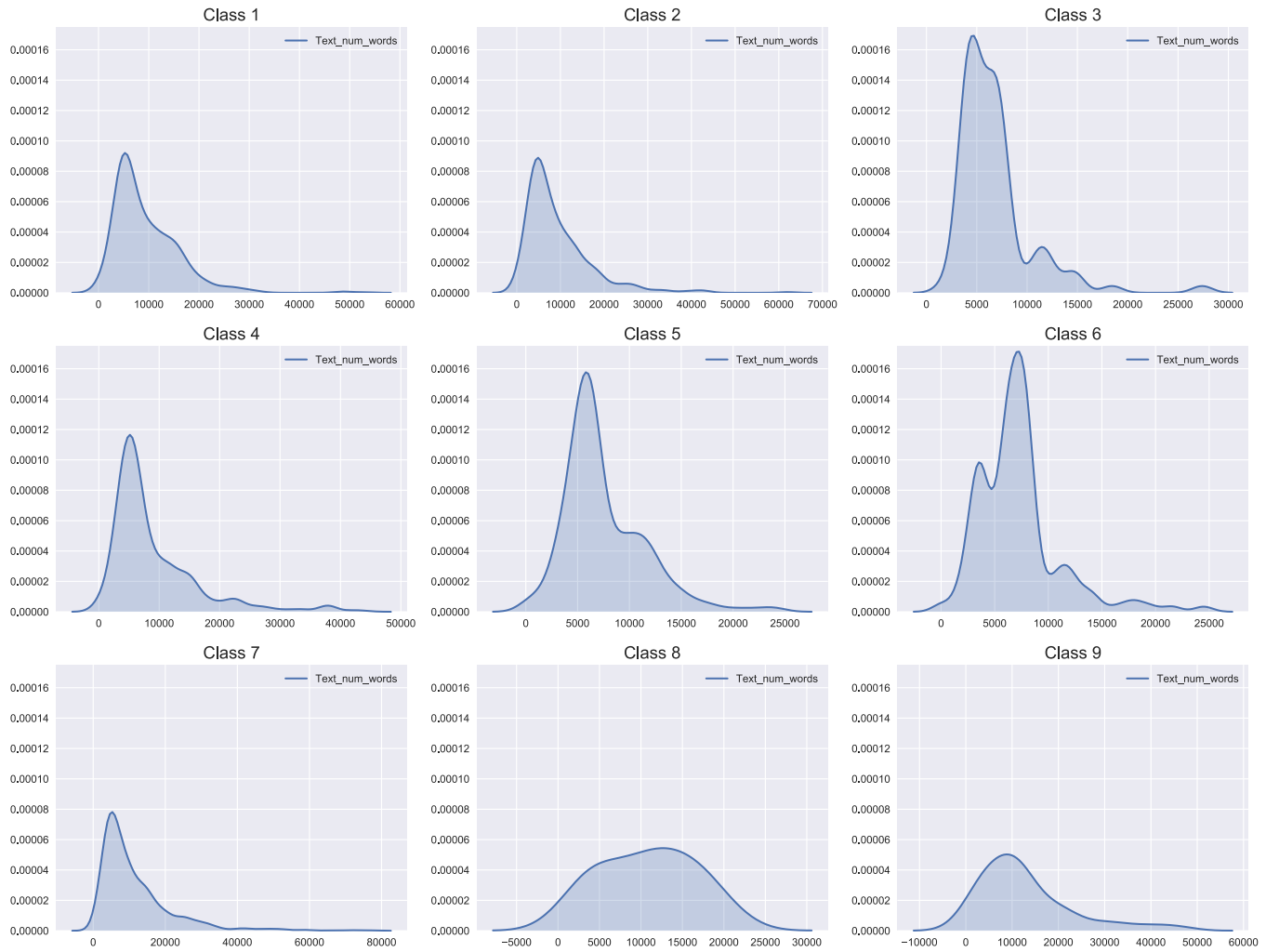
Classes 1, 2, 4, 5, 6, & 7 have a minimum word count of less than 500 words which suggests that there could be some erroneous records. This is becuase we would expect a research paper which is used as a reference to be significantly longer than this.

This violin plot also shows the distribution of word counts by class:

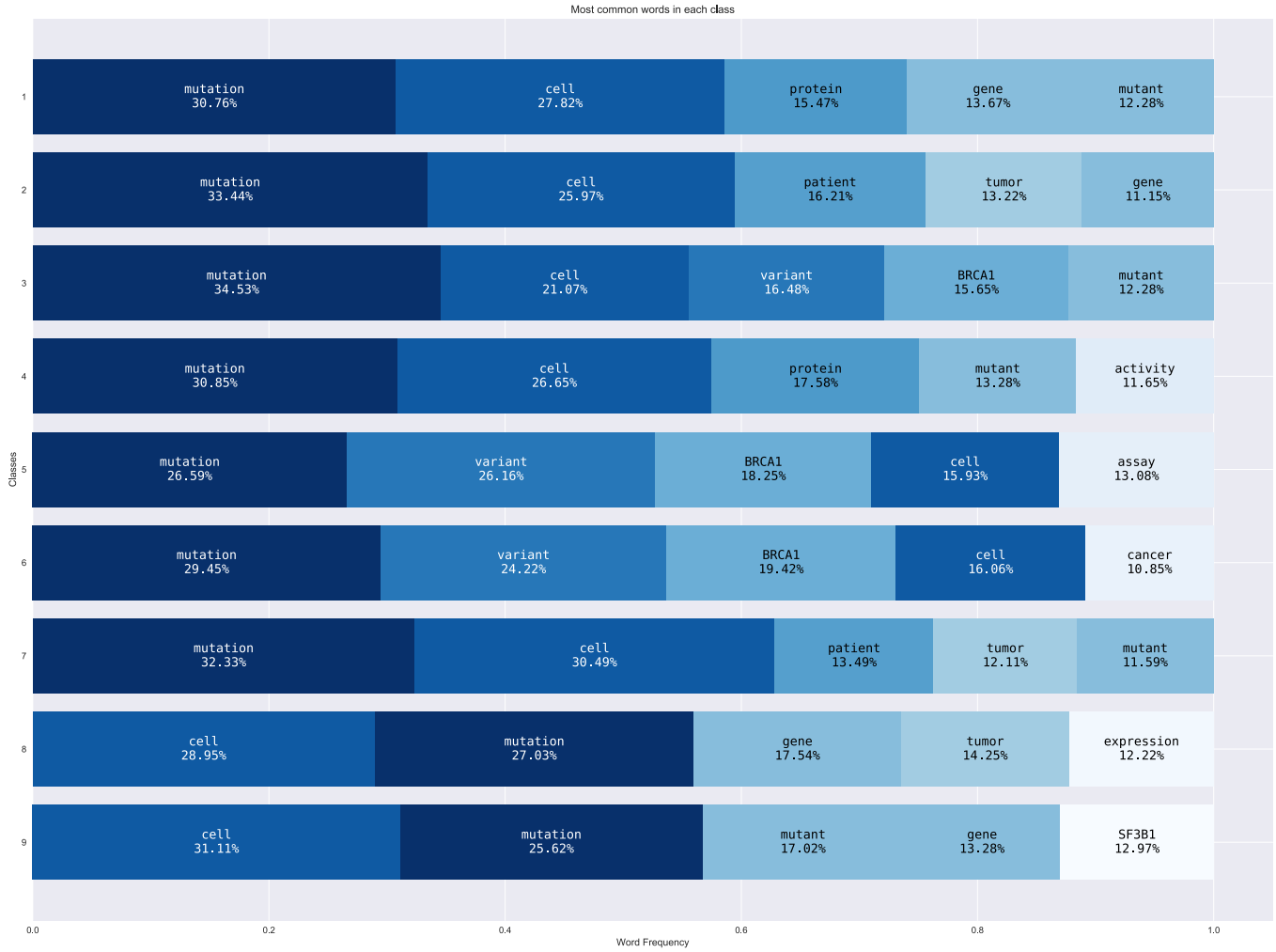


Classes 3, 5 & 6 seem to contain a lower number and smaller range of words than the other classes.

The following subplots show the distribution of the number of words by class, which echoes the violin plot above:


Class 1
Text_num_words
Class 2
Text_num_words
Class 3
Text_num_words
Class 4
Text_num_words
Class 5
Text_num_words
Class 6
Text_num_words
Class 7
Text_num_words
Class 8
Text_num_words
Class 9
Text_num_words

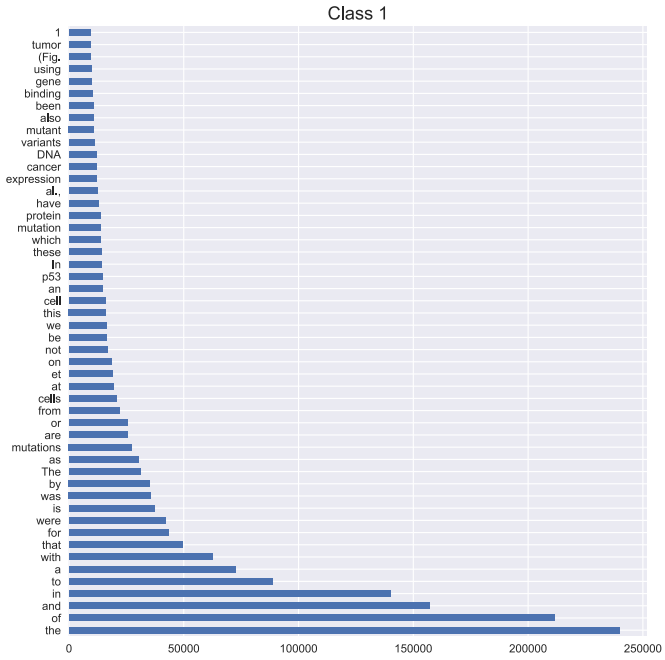
Next, I want to look at the 5 most popular words by class:

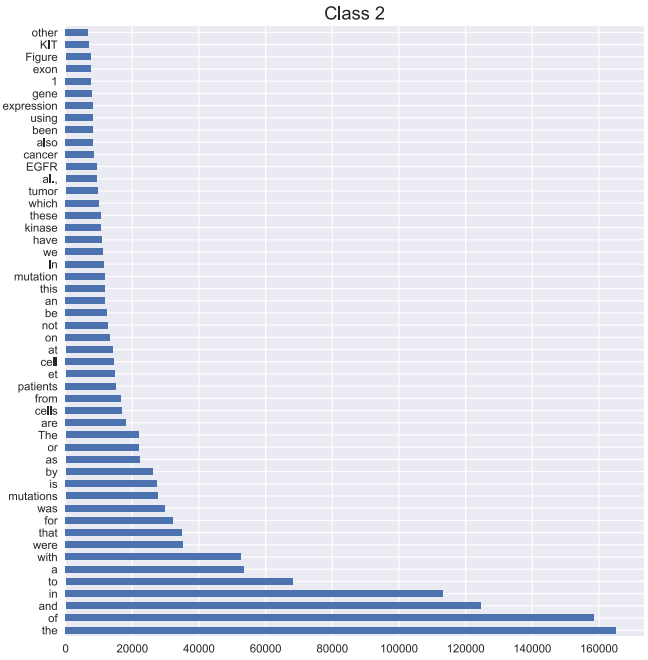


Its interesting to see here the somewhat expected terms: mutant/mutation, cell, variant, protein, tumor, gene are present across the majority of classes.

Classes 5, 6 again seem to be closely linked to gene BRCA1.

Lastly, lets expand the range of words. The following subplots show the top 50 words by frequency for each class:


Class 1
1
tumor
(Fig.
using
gene
binding
been
also
mutant
variants
DNA
cancer
expression
al.,
have
protein
mutation
which
these
In
p53
an
cell
this
we
be
not
on
et
at
cells
from
or
are
mutations
as
The
by
was
is
were
for
that
with
a
to
in
and
of
the
0
50000
100000
150000
200000
250000

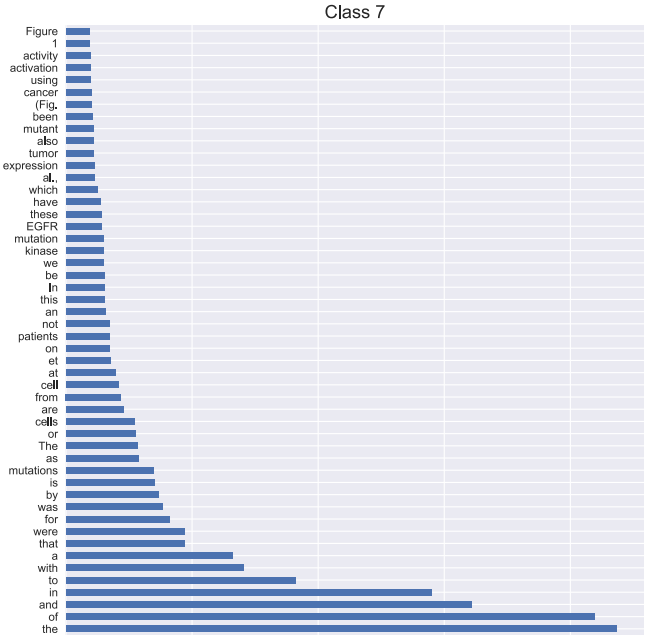

Class 2
other
KIT
Figure
exon
1
gene
expression
using
been
also
cancer
EGFR
al.,
tumor
which
these
kinase
have
we
In
mutation
this
an
be
not
on
at
cell
et
patients
from
cells
are
The
or
as
by
is
mutations
was
for
that
were
with
a
to
in
and
of
the
0
20000
40000
60000
80000
100000
120000
140000
160000

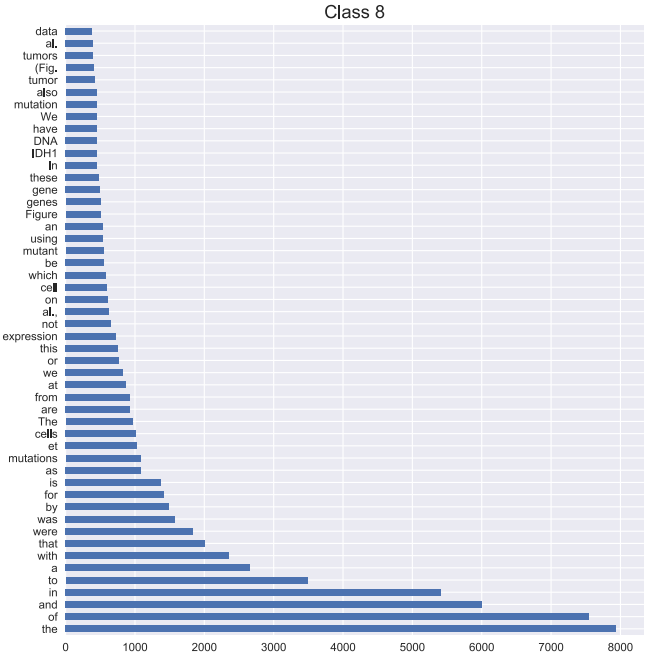

Class 3
1
mutant
expression
kinase
protein
functional
Figure
mutants
cell
FLT3
activity
an
In
which
cancer
been
these
mutation
ALK
we
have
at
be
on
this
variants
not
al.,
from
et
cells
or
are
BRCA1
The
by
was
as
is
mutations
for
were
that
with
a
to
in
and
of
the
0
5000
10000
15000
20000
25000

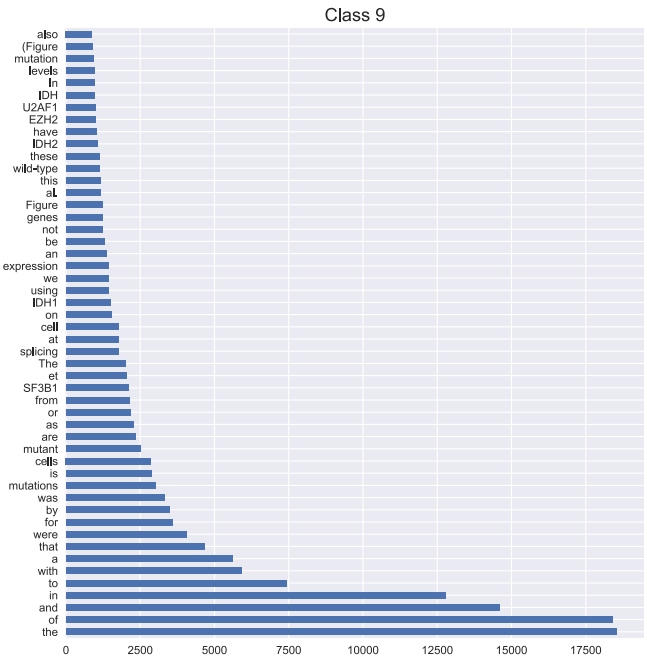

Class 4
1
binding
Figure
using
p53
(Fig.
mutant
also
expression
variants
been
BRCA1
mutation
which
an
In
activity
these
have
cell
al.,
we
this
protein
be
not
PTEN
et
on
at
from
cells
are
or
The
as
mutations
was
is
by
were
for
that
with
a
to
in
and
of
the
0
50000
100000
150000
200000
250000

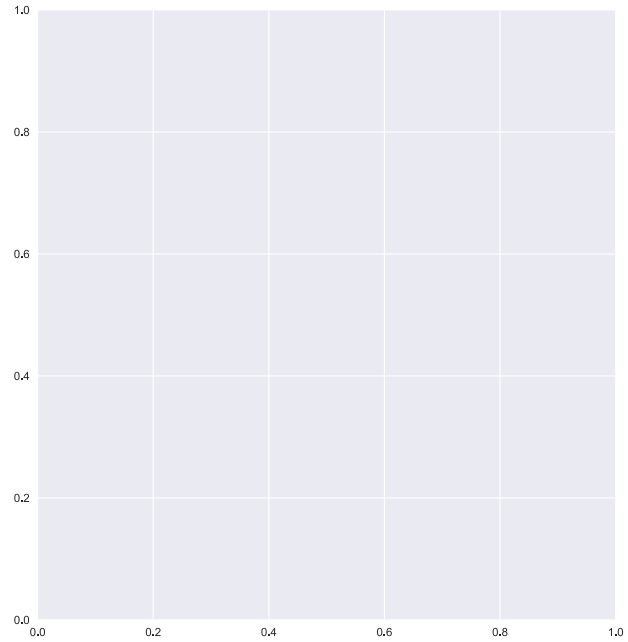

Class 5
domain
We
used
wild-type
using
BRCT
cell
also
activity
data
et
been
cancer
mutation
In
an
VUS
protein
which
functional
have
these
cells
at
not
we
this
from
be
on
or
are
The
by
was
mutations
variants
BRCA1
as
is
were
that
for
with
a
to
in
and
of


Class 6
0
domain
used
tumor
protein
(Fig.
kinase
activity
cell
using
been
also
data
et
deleterious
cancer
which
In
cells
have
an
mutation
these
not
we
this
at
on
from
be
variants
BRCA1
mutations
or
was
The
are
as
by
is
were
for
that
with
a
to
in
and
of









Note that becuase no pre-processing has been done so far that there are many 'stop words' such as 'the', 'in' 'of' which are largely noise and don't add to the meaning being conveyed in the text. I will attempt to remove these before passing the text to the model.

2.4 Algorithms and Techniques

2.4.1 Text pre-processing & Natural Language Processing

Based on the exploratory data analysis (EDA) and visualisation, the feature which seems to provide the most information regarding how to classify the data is the text field.

Before we can use this text as a predictive tool, or use it as an input to a classification model it needs to be pre-processed. This is essential to get the most singal from the text and discard the noise as discussed below.

Text pre - processing includes:

- Removal of basic English language stop words
- Identification of stop words related to scientific literature which contain little information to help classify the data
- Attempt to identify key terminology for classification of mutations in EDA

We need to remove 'stop words' (e.g. the, of, and etc.) from the text as these are mostly noise and don't help to convey the meaning of the text. Including these words will decrease the accuracy of the model.

Similarly, stop words which are not common in English language, but are common in scientific literature (e.g. et., al, fig etc.) are also largely noise and should also be removed to further increase the accuracy of the model.

Identification of the key terminology for classification of mutations will be helped greatly by the exploratory data analysis & visualisation above.

I will use several Python packages to perform the text processing. These include:

NLTK (Natural Language Toolkit)

- NLTK Tokenizer - for splitting the text into word 'tokens' for visualisation / manipulation
- WordNetLemmatizer - for condensing words with similar meaning i.e. mutation & mutant are derivatives and can be condensed to find the main term used for explanation in the text
- English stopwords - common words in the English language which have little meaning e.g. the, of, and

Doc2Vec

- For 'vectorising' the text documents. Doc2Vec is a powerful tool which creates vectors between the sentences of text corpus to encode meaning of the text body. These vectors can then be used in predictive models.

Re

- Python regular expressions - for string matching / replacement / cleansing of punctuation which is meaningless in terms of interpreting the text.

TruncatedSVD

- Takes the vectorised text documents as input and performs a variant of singular value decomposition (SVD) that only computes the *k* largest singular values, where *k* is a user-specified parameter. When truncated SVD is applied to term-document matrices, this transformation is known as latent semantic analysis (LSA), because it transforms such matrices to a “semantic” space of low dimensionality. In particular, LSA is known to combat the effects of synonymy and polysemy (both of which roughly mean there are multiple meanings per word), which cause term-

document matrices to be overly sparse and exhibit poor similarity under measures such as cosine similarity.

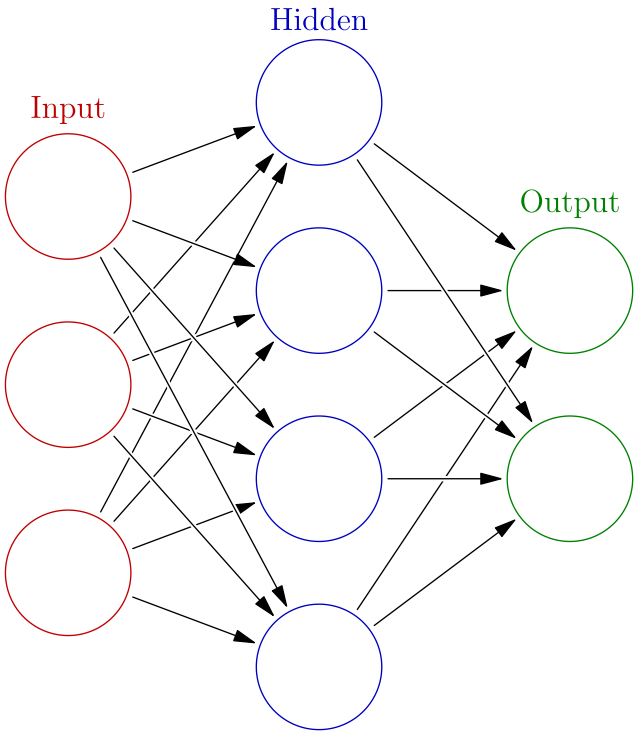
References:

NLTK (http://nltk.org)
Doc2Vec (https://radimrehurek.com/gensim/models/doc2vec.html)
Re (https://docs.python.org/3/library/re.html)
TruncatedSVD (http://scikit-learn.org/stable/modules/decomposition.html#lsa)

2.4.2 Building a classification model

As stated in my project proposal I'm interested in assessing the performance of the Keras deep learning library and in particular the Keras classification wrapper around the Scikit learn package to build a model for this classification task.

As mentioned in section 1.2 I utilised an artificial neural network for this task. As a reminder, this is a network of several layers of neurons which process some input and pass it via the various layers to produce an output signal.



In this case the inputs were the features of the training data, namely the Gene, Variant & Text.

Components of an artificial neural network

Neurons

A neuron with label *j* receiving an input $p_j(t)$ from predecessor neurons consists of the following components:

- an activation $\alpha_j(t)$, depending on a discrete time parameter,
- possibly a threshold θ_j, which stays fixed unless changed by a learning function,
- an activation function *f* that computes the new activation at a given time t+1 from $\alpha_j(t)$, θ_j and the net input $pj(t)$ giving rise to the relation
  $\alpha_j(t+1) = f\,(\alpha_j(t), p_j(t), \theta_j)$,
- and an output function f_{out} computing the output from the activation $o_j(t) = f_{out}\,(\alpha_j(t))$.

Often the output function is simply the Identity function.

An input neuron has no predecessor but serves as input interface for the whole network. Similarly an output neuron has no successor and thus serves as output interface of the whole network.

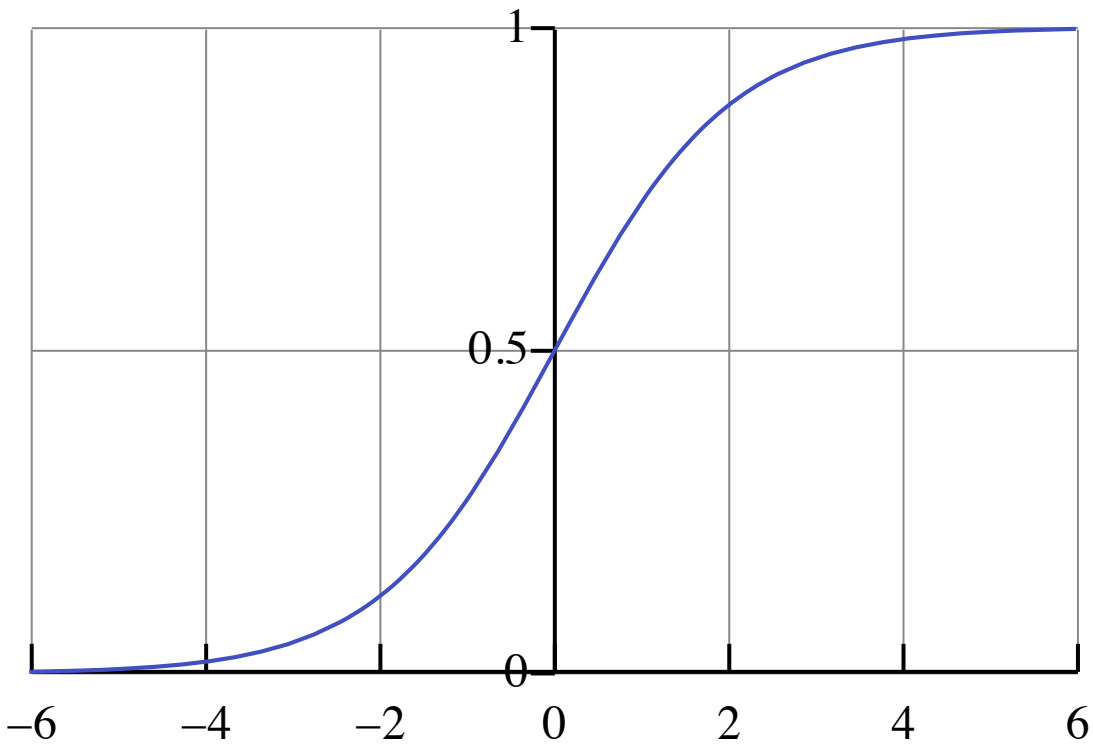
Connections and weights

The network consists of connections, each connection transferring the output of a neuron *i* to the input of a neuron *j*. In this sense *i* is the predecessor of *j* and *j* is the successor of *i*. Each connection is assigned a weight w_{ij}.

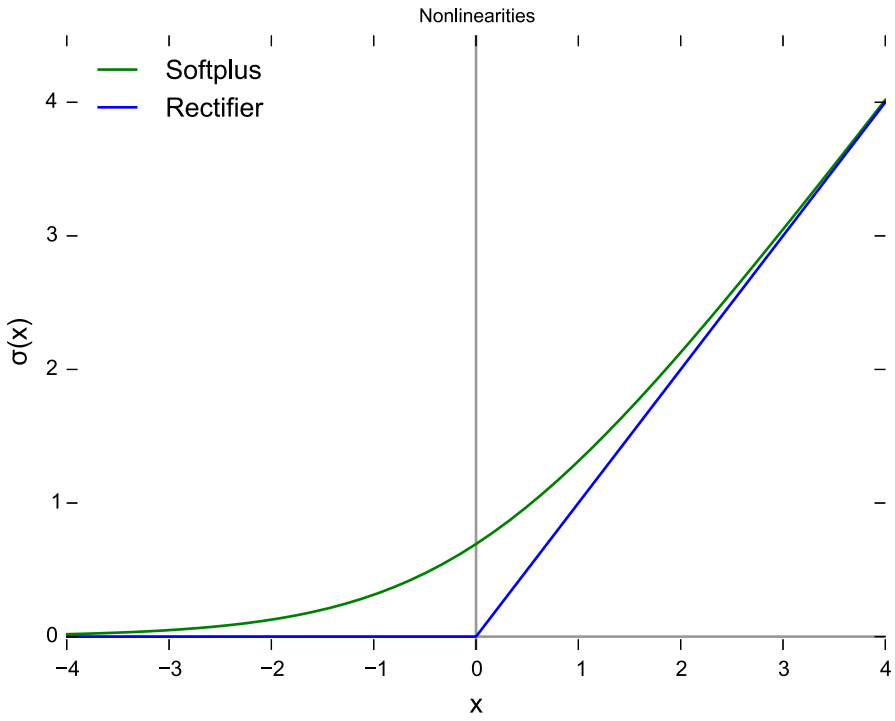
Activation Functions

The input layer passes its output to a neuron in layer it is connected to. Depending on the type of layer this could be connected to all of input neuron, or a subset. The connected neuron will be prescribed an activation function which acts as a logic gate to determine whether or not that neuron should in turn 'fire' and pass on a signal to the layer that it is connected to.

An example of an activation function is the Sigmoid function:



The specification of activation function is critical to the correct training and output of the network and many research papers have been written about this subject. Suffice to say that for this project I used the ReLU function for fully connected layers to prevent the 'disappearing gradient function' in backpropogation whilst training the model and Sigmoid function in conjucntion with Softmax for classification on the output layer, as this is suitable in for assigning the probability that a record in the data belongs to a certain output class where the number of classes is greater than 2.



Propagation function

The propagation function computes the input $p_j(t)$ to the neuron j from the outputs $o_i(t)$ of predecessor neurons and typically has the form:

$p_j(t) = \sum i \; o_i(t) \; w_{ij}$.

The learning rule is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output. This learning process typically amounts to modifying the weights and thresholds.

Model Summary

In layman's terms, the neural network attempts to fit a gradient to the data in order to make it seperable, thus resulting in a classification of the data in the output. It acheieves this by altering the weights / gradients of each of the neurons in the network on each training pass until it fits the data as best as it can on the final pass.

I chose to start with a simple model to assess a baseline of performance versus the benchmark before trying to apply more advanced models to improve performance.

Here is a summary of the initial model:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               154112
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 80)                41040
_________________________________________________________________
dense_3 (Dense)              (None, 9)                 729
=================================================================
Total params: 195,881
Trainable params: 195,881
Non-trainable params: 0
_________________________________________________________________
```

The first layer (input layer) is a fully connected layer with 512 neurons and ReLU activation which is connected to a dropout layer to ensure that all neurons are trained by randomly excluding neurons whilst the model is being trained. This in turn is connected to a further fully connected layer with 80 neurons and ReLU activation and finally to the output layer of 9 neurons (one for each of the classes to be predicted) with Sigmoid activation.

In total there are 195k tunable parameters in the model which can be altered to improve it's fit on the training data.

The final output of the model is a prediction (expressed as a percentage) of the class of mutation to which the particular Gene and Variant record belongs. The sum of the predictions across all the classes is 100%.

Keras Scikit Learn API (https://keras.io/scikit-learn-api/)
Scikit Learn (http://scikit-learn.org/stable/)
Artificial Neural Networks - Wikipedia (https://en.wikipedia.org/wiki/Artificial_neural_network)

2.4.3 Predicting the class the genetic mutation belongs to using the constructed model

Including:

- Using the model constructed to predict the correct class of a genetic mutation
- The model should outperform the naive benchmark
- Submit the result to the Kaggle compeition in the prescribed format

2.5 Benchmark

The benchmark for this project will be the naive predictor of the distribution of the classes in the training data.

Class	Percent of Total
1	17.10
2	13.61
3	2.68
4	20.66
5	7.29
6	8.28
7	28.70
8	0.57
9	1.11

Upon creating an populating a submission file with these predictions of the classes for every record in the test data the **multiclass log loss was returned as 1.74362**. This is the benchmark I intend to beat.

Although, by comparison with the public leaderboard on Kaggle this doesn't look like a difficult target to beat, I am very much aware that this is my first competition and the task at hand is not easy, especially given my limited experience with natural language & deep learning tools.

3. Methodology

Please refer to the 'Predict Cancer Project.ipnyb' file for the detailed code to support this section.

3.1 Data Preprocessing

Initially I imported the relevant Python packages and set a random seed to allow reproducible results in later steps.

3.1.1 Import Data into Pandas Dataframes

I then created several Pandas dataframes to hold the data:

- train_x - contains the training data dropping the class field (merges the train_variant and train_text files on ID so that the text field is now in the same dataframe as the gene and variant data)
- train_y - contains the labels for the training data i.e. the class field which was dropped from train_x
- text_x - contains the gene and variant data to be tested on (also merges the test_variant and test_text files on ID so that the text field is in the same dataframe as the gene and variant data)
- all_data is both training and test data combined

Here is the relevant code:

```
# Read Data
train_variant = pd.read_csv("training_variants")
test_variant = pd.read_csv("test_variants")
train_text = pd.read_csv("training_text", sep="\|\|", engine='python', hea
der=None, skiprows=1, names=["ID","Text"])
test_text = pd.read_csv("test_text", sep="\|\|", engine='python', header=N
one, skiprows=1, names=["ID","Text"])

# Join text and variant data on ID
train = pd.merge(train_variant, train_text, how='left', on='ID')

# Drop Class labels from training data
train_y = train['Class'].values
train_x = train.drop('Class', axis=1)

#  Check shape of training data
train_size = train_x.shape[0]
# number of train data : 3321

# Join text and variant data on ID
test_x = pd.merge(test_variant, test_text, how='left', on='ID')

# Check shape and size of test data
test_size = test_x.shape[0]
test_index = test_x['ID'].values
# number of test data : 5668

# Join all data for analysis
all_data = np.concatenate((train_x, test_x), axis=0)
all_data = pd.DataFrame(all_data)
all_data.columns = ["ID", "Gene", "Variation", "Text"]
```

3.1.2 Text Pre-processing

The next step was to pre-process and clean the text. Here I used NLTK, Re and Doc2Vec Python packages.

I defined several functions which act to clean the text of punctuation, convert it to lower case, remove stopwords, split it into individual word tokens and construct a list of numbered sentences for use in Doc2Vec in later steps. Please refer to the code in this section for explanatory notes:

```
# Define text pre-processing funtions

# Constructs a labelled list of sentences from a text corpus
def constructLabeledSentences(data):
    sentences=[]
    for index, row in data.iteritems():
        sentences.append(LabeledSentence(utils.to_unicode(row).split(),
 ['Text' + '_%s' % str(index)]))
    return sentences

# Converts text to lower case and removes punctuation
def cleanup(text):
    text = text.lower()
    text = text.translate(str.maketrans("","", string.punctuation))
    return text

# Removes punctuations, converts text to lower case, splits words into tok
ens, removes stopwords and digits from text
def textClean(text):
    text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)
    text = text.lower().split()
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops and not w.isdigit()]
    text = " ".join(text)
    return(text)
```

Initially I simply ran the cleanup function on the text corpus to check the predictive power of the text without removing stopwords. This gave me a baseline for later analysis.

Some issues I faced here included: how to effectively remove single digits and reference notation from the scientific papers, which I managed to do via the use of the string package in Python, removing punctuation and looping through each word token to check if it is a digit.

This is one of the more time consuming and challenging parts of the process due to the large corpus of text in this dataset. Deciding how to pre-process the text component has a significant impact on the output of the model and much time could be spent here refining and experimenting with various parameters, stop words and other natural language processing tools to exploit some of the features that I identified in the EDA. My skills in this area are limited, hence the minimal pre-processing applied for this project, however, the best performing kernels on Kaggle have likely applied more advanced techniques here.

3.1.3 Text Vectorisation with Doc2Vec

The next step was to create the document vectors which aim to determine the relationships between sentences in the text body. Given the length of the body I elected to initally use 300 as the variable for number of dimensions. If time has allowed, I would have adjusted this variable to see how it affected the accuracy of the model.

The other initial variables set here are: min_count = 1 which is used to give words which occur more frequently a higher rating; window = 5, the maximum distance between the current and predicted word within a sentence; iter=5, is the number of training iterations, I would have liked to have used more, but due to only being able to run this project on a CPU I was time limited; sample = 1e-4, threshold for configuring which higher-frequency words are randomly downsampled.

```
# Create text features using Doc2Vec

doc2vec_epocs=5 #TODO Change to 100
model=None
filename='vectors/docEmbeddings_nostop_all_5.d2v'
if os.path.isfile(filename):
    model = Doc2Vec.load(filename)
else:
    model = Doc2Vec(min_count=1, window=5, size=INPUT_DIM, sample=1e-4, ne
gative=5, workers=8, iter=doc2vec_epocs, seed=1)
    model.build_vocab(sentences)
    model.train(sentences, total_examples=model.corpus_count, epochs=mode
l.iter)
    model.save(filename)
```

Processing the text vectors via Doc2Vec was also an extremely time consuming process whilst running on a CPU and I would recommend using a GPU or high powered machine for this processing. This is again due to the large text corpus and the number of vectors the algorithm needed to create to map the relationships in the text.

Given time, there would be many parameters I would have liked to experiment with here including more training passes over the text, to better extract the predictive power it contains.

I highly recommend saving any vectors produced so that they can be re-loaded later if needed. This can be acheieved by the model.save and model.load function calls with Doc2Vec respectively.

These articles were very helpful in setting the variables and creating the code to proces the data with Doc2Vec:

- Doc2Vec in a Simple Way (https://medium.com/@mishra.thedeepak/doc2vec-in-a-simple-way-fa80bfe81104)
- Sentiment Analysis Using Doc2Vec (https://linanqiu.github.io/2015/10/07/word2vec-sentiment/)
- Doc2Vec Tutorials (https://github.com/RaRe-Technologies/gensim/blob/develop/tutorials.md#tutorials)

3.1.4 Create Arrays for Model Input

The next step was to create arrays of the vectors generated in the last step for both the training and test data and to use SKLearn's LabelEncoder function to convert the vectors for the training data to the 9 class categories we are attempting to predict.

```
# Create vector arrays for Keras model input
train_arrays = np.zeros((train_size, INPUT_DIM))
test_arrays = np.zeros((test_size, INPUT_DIM))

for i in range(train_size):
    train_arrays[i] = model.docvecs['Text_'+ str(i)]

label_encoder = LabelEncoder()
label_encoder.fit(train_y)
encoded_y = np_utils.to_categorical((label_encoder.transform(train_y)))

j=0
for i in range(train_size, train_size + test_size):
    test_arrays[j] = model.docvecs['Text_'+ str(i)]
    j=j+1
```

The data is now in the correct format and held in arrays which can be passed to the Keras model for fitting and prediction.

3.2 Implementation

Finally, I needed to create the initial Keras Deep Learning model to train on the data we have prepared. As discussed in 2.4.2, I chose a simple 3 layer model here which is summarised as follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               154112
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 80)                41040
_________________________________________________________________
dense_3 (Dense)              (None, 9)                 729
=================================================================
Total params: 195,881
Trainable params: 195,881
Non-trainable params: 0
_________________________________________________________________
```

The model is composed of a fully connected input layer with 512 nodes and relu activation, a dropout layer set to 20% dropout, a fully connected layer with 8 nodes and relu activation and finally a fully connected layer with 9 nodes and sigmoid activation, one for each of the classes we wish to predict.

The code required to define the model in Keras is quite minimal. The initial model is defined as follows:

```
# define initial model
def initial_model():
    model = Sequential()
    model.add(Dense(512, input_dim=INPUT_DIM, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(80, activation='relu'))
    model.add(Dense(9, activation="softmax"))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metri
cs=['accuracy'])
    return model
```

When fitting this model it was run for 10 epochs to avoid overfitting, with a batch size of 64 and a validation split of 5% of the training data. This due to the small size of the data set and wanting to retain as much of the data to train on as possible, larger training data sets have been shown to be very beneficial for text / semantic analysis s shown by Word2Vec & Doc2Vec experiments.

The following code is used to fit the model using the Scikit learn Keras Classifier wrapper:

```
estimator = KerasClassifier(build_fn=initial_model, epochs=10, batch_size=
64)
estimator.fit(train_arrays, encoded_y, validation_split=0.05)
```

Fitting the initial model produced the following output:

```
Train on 3154 samples, validate on 167 samples
Epoch 1/10
3154/3154 [==============================] - 43s - loss: 1.5865 - acc: 0.4245 - val_loss: 1.5344 - val_acc: 0.4012
Epoch 2/10
3154/3154 [==============================] - 0s - loss: 1.2073 - acc: 0.5694 - val_loss: 1.6177 - val_acc: 0.4671
Epoch 3/10
3154/3154 [==============================] - 0s - loss: 1.0250 - acc: 0.6376 - val_loss: 1.6358 - val_acc: 0.4132
Epoch 4/10
3154/3154 [==============================] - 0s - loss: 0.8878 - acc: 0.6915 - val_loss: 1.8148 - val_acc: 0.4192
Epoch 5/10
3154/3154 [==============================] - 0s - loss: 0.7774 - acc: 0.7334 - val_loss: 1.8705 - val_acc: 0.4431
Epoch 6/10
3154/3154 [==============================] - 0s - loss: 0.6930 - acc: 0.7530 - val_loss: 1.9155 - val_acc: 0.4132
Epoch 7/10
3154/3154 [==============================] - 0s - loss: 0.6010 - acc: 0.7812 - val_loss: 1.9981 - val_acc: 0.4371
Epoch 8/10
3154/3154 [==============================] - 0s - loss: 0.5540 - acc: 0.8069 - val_loss: 2.0817 - val_acc: 0.4611
Epoch 9/10
3154/3154 [==============================] - 0s - loss: 0.5073 - acc: 0.8186 - val_loss: 2.1813 - val_acc: 0.4431
Epoch 10/10
3154/3154 [==============================] - 0s - loss: 0.4614 - acc: 0.8370 - val_loss: 2.1631 - val_acc: 0.4790
```

The model can now be used for prediction by calling the predict_proba function:

```
y_pred = estimator.predict_proba(test_arrays)
```

And the accuracy of those predictions is assessed via the score function:

```
estimator.score(test_arrays, y_pred)
```

3.3 Refinement

The first refinement I attempted to try and improve the result was by altering the Keras model to a deep learning model as follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 512)               154112
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_14 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_15 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_16 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_17 (Dense)             (None, 80)                41040
_________________________________________________________________
dense_18 (Dense)             (None, 9)                 729
=================================================================
Total params: 983,849
Trainable params: 983,849
Non-trainable params: 0
_________________________________________________________________
```

This model has several more tunable layers and a large increase of total parameters over the initial model (983k vs. 193k).

Generally speaking, by increasing the depth of a deep learning model each of the additional layers is able to add abstraction to each the previous layers which should help to improve the predictive power of the model. Although, academics have shown, that relatively shallow (16-layer) models can perform nearly as well as models with nearly ten times as many layers if correctly tuned and parameterised.

Given my relatively inexperience with deep learning models I was unsure of whether or not this model would outperform the simple model, or how many layers would be needed to significantly increase its performance. The main restriction on adding further layers was the processing time required to run them. Given more time to work on this I would have liked to try much deeper models to assess performance and I would encourage readers with higher computing power to proceed along those lines.

As the text feature seemed to be the the feture with the most predictive power the next logical step for refinement was to further manipulate the text by removing stopwords and re-running Doc2Vec to see if this helped to remove noise from the vectors.

I will provide discussion of the outputs from those alterations against both of the Keras models in the results section below.

4. Results

4.1 Model Evaluation & Validation

Initial Model
(initial_sub.csv)

Implementing this model and creating a submission file ('initial_sub.csv') returned an MLL of **1.09704** which already beats the naive predictor benchmark of **1.74362**.

This is quite impressive with a relatively simple model, but there is much refinement that can be done with respect to model parameters at both the natural language and Keras model stages of the process.

Initial Model - No stop words
(initial_nostop_sub.csv)

This model produced my best MLL of **0.98359** and proves that there was significant noise contained in stopwords such as 'the', 'in', 'at'. I'm sure that this could be further refined, but my natural language processing skills are limited and it would take me siginificant further time for what would likely be little progression with this model.

Advanced Model
(advanced_sub.csv)

Surprisingly this model did not perform as well as the simple model, producing a MLL of **1.41949**, even though the validation accuracy based on categorical cross entropy we similar on both models. This was quite concerning as it was then difficult to use this metric as a guide to model performance.

Advanced Model - No stop words (advanced_nostop_sub.csv)

Again this model showed significant improvement over the baseline advanced model. The MLL improved to **1.07628** from 1.41949. This also shows that removing stopwords had a dramatic effect on the predictive power of the model all else equal.

4.1.1 Summary

I do not feel that I have produced a particularly robust model to address this problem because of these reasons:

- The accuracy produced on the test data set is only around 23% in the case of the best model
- The MLL on the test data is only 0.98359. Much superior results have been produced by fellow Kaggle competitors using more skilled and advanced techniques

Much more work is required to either enhance or extract meaning from the text in order to increase its predictive power.

I am confident that it would generalise well to other scientific papers in this domain give then large corpus of text in this dataset on which it was trained. This would certainly be another area for further exploration. There is a risk that the model is sensitive to the machine generated, or outlying data in this dataset as I did little pre-processing to address this before training the Doc2Vec model. Again this provides an opportunity to further improve the model.

4.2 Justification

The naive predictor for this classification task was simply the distribution of the classes in the training data. This should not be difficult to beat with one of a number of trained classifcation tools which are trained on the same data set.

I elected to use a deep learning model which outperformed the naive predictor by a significant margin with minimal pre-processing of the input data and little parameter tuning of the Keras classifier.

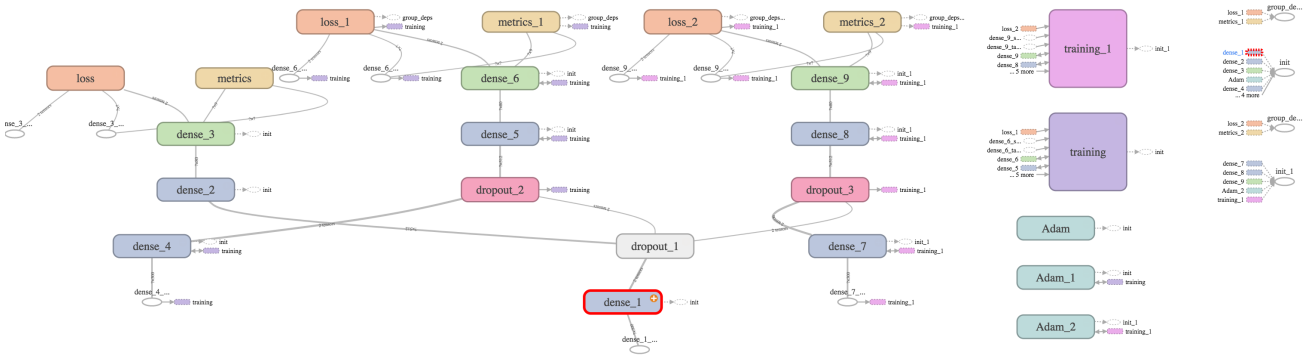
5. Conclusion

5.1 Free Form Visualisation

I have used TensorBoard from the TensorFlow package to explore some visulisations of the model.

5.1.1 Model Graph

The first figure shows a graph of the final model **Initial Model - No stop words**:



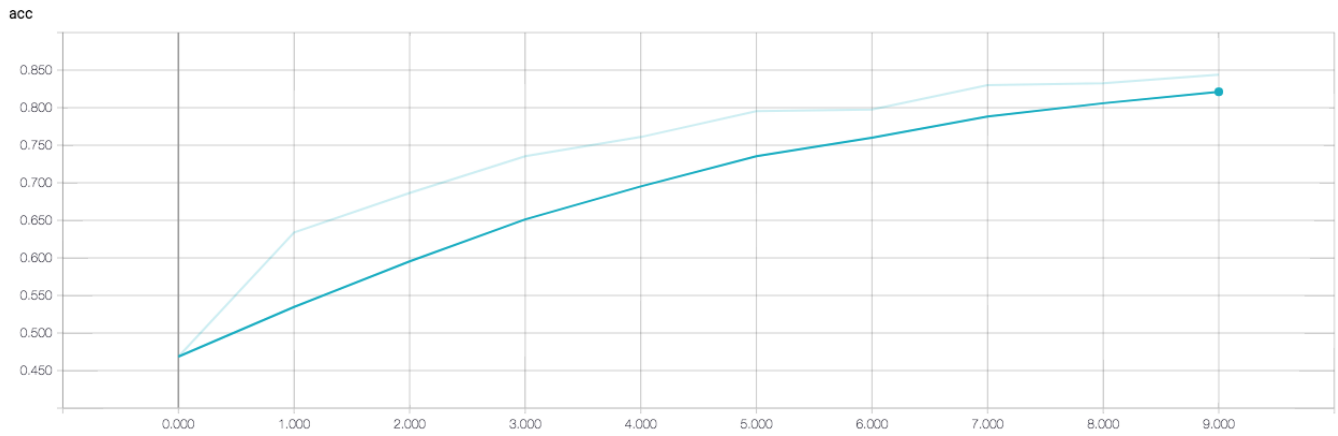
In summary, what this image shows are the pathways through the model during the 10 training epochs.

The model begins at the input layer (dense_1) which has an associated dropout layer (dropout_1) to ensure all neurons in the layer are trained. The input layer is then fed to the second fully connected layer (dense_2) and subsequently to the output layer (dense_3), the output from this layer is used to calculate the accuracy and loss metrics which the model attempts to incease the performance of during the fitting process.

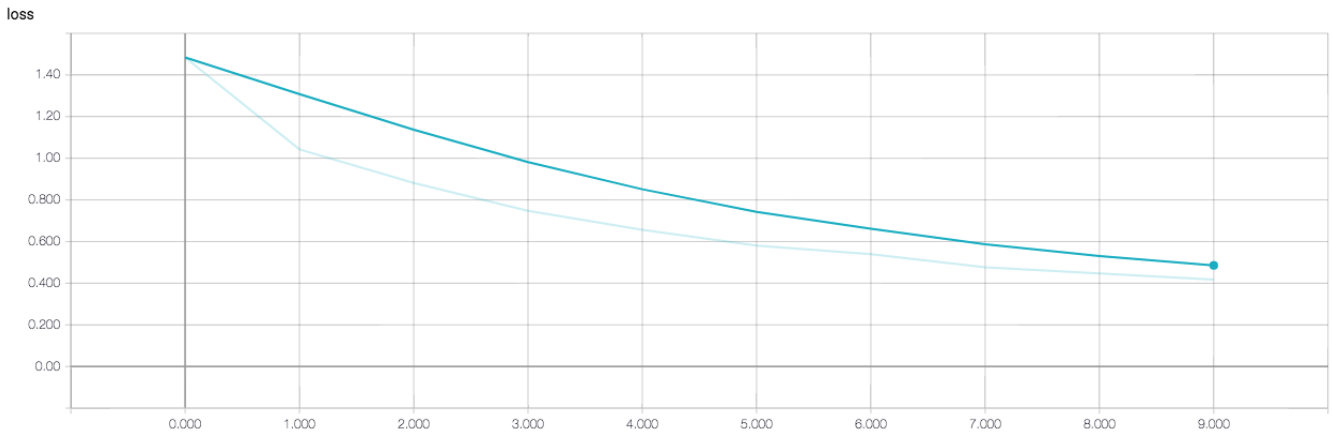
5.1.2 Accuracy and Loss Metrics

Training Data

Figure 2 shows how the accuracy of the model increased on the training data over the training steps. (Y axis = accuracy, X axis = training step). The bold line is a smoothed trend line vs. the light-weight line showing the actual results. This depiction will be common to each of the figures in this section.



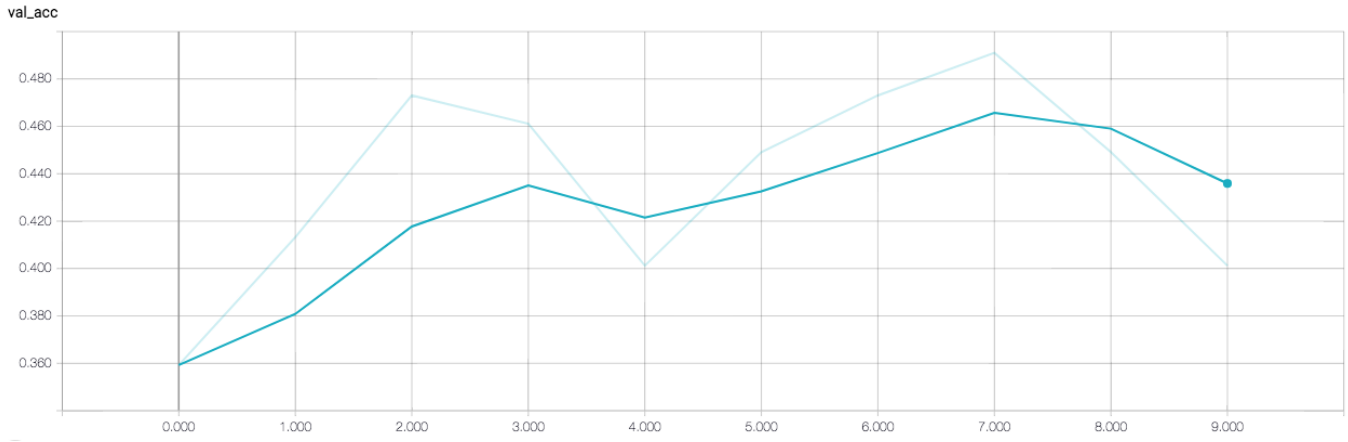
The cross validation loss, which the model was tasked to minimise, also decreased over the training steps.



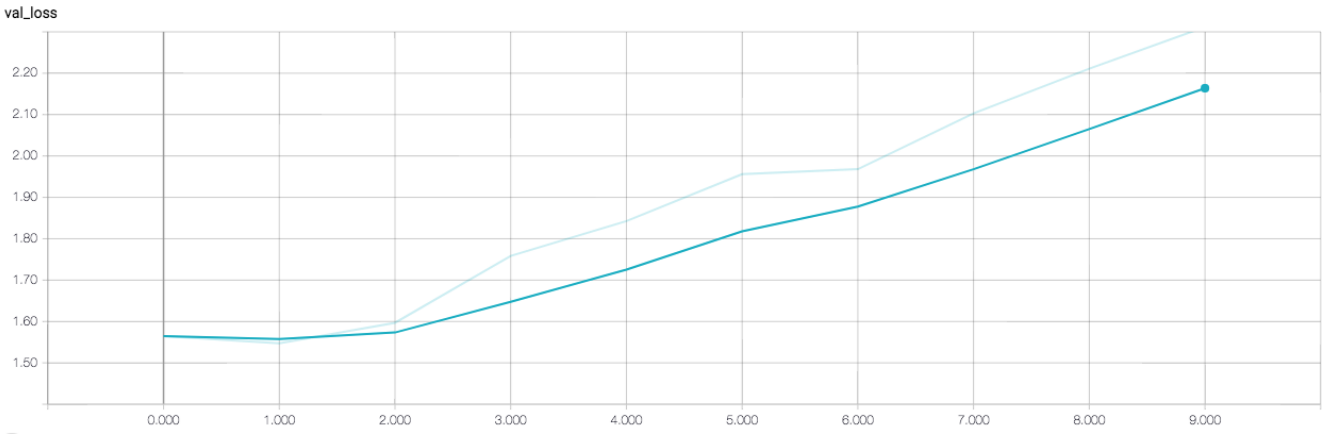
Validation Data

In contrast, the increase in accuracy on the validation data (held out training data from the training data set) was much less pronounced and could not increase beyond ~48%.

This helps to illustrate perhaps why the model would not generalise well to unseen data. The model was likely overfitting to the training data which is often a risk with deep learning models and hence why I used a minimal number of traning epochs.



The categorical cross entropy also increased over successive training runs. This is not desirable, as it indicates that the model was probably overfit to the training data.

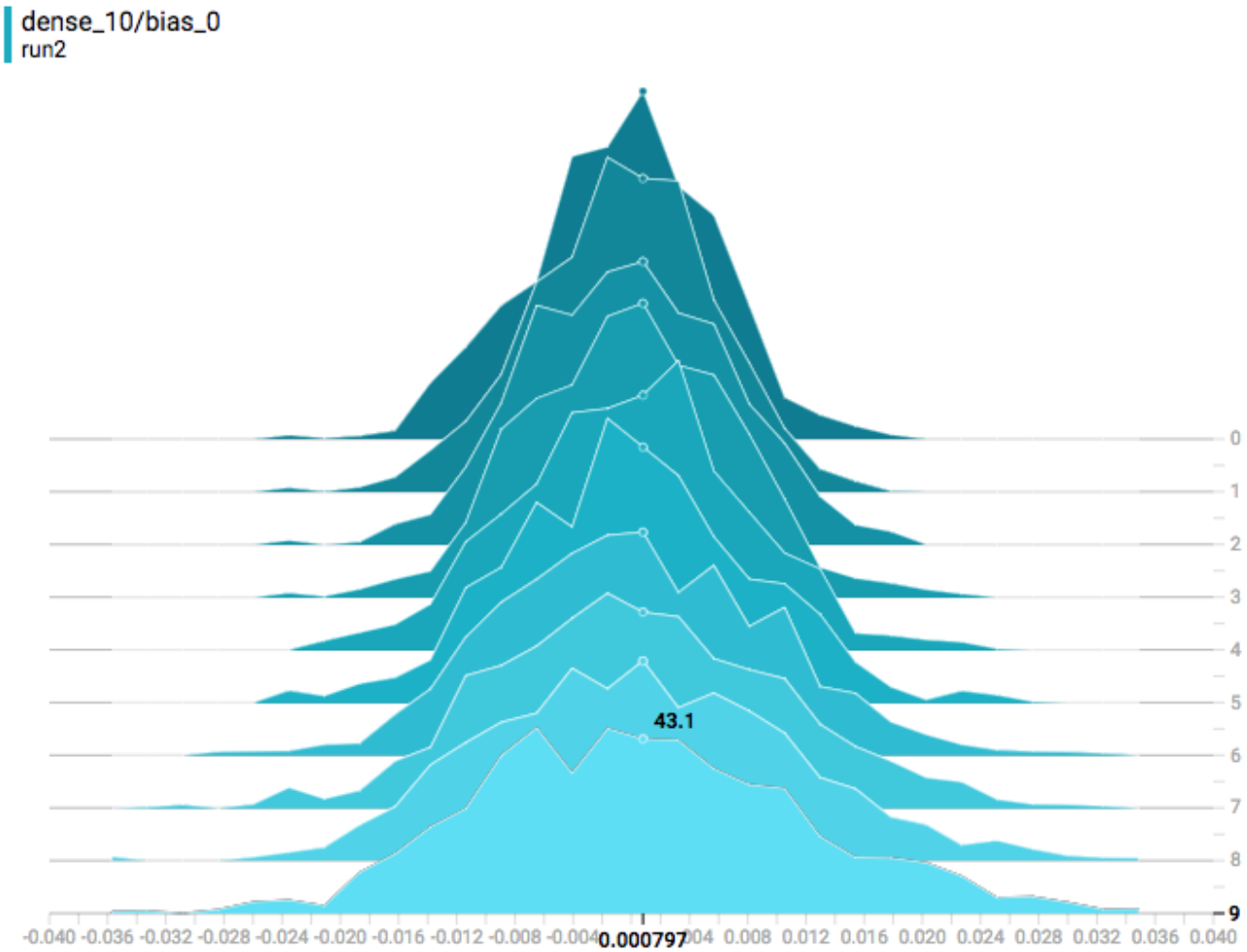


5.1.3 Model Weights & Biases

TensorBoard also provides a way to visualise the bias and weights of each layer via histograms showing the distribution of those weights across the nodes in the layer.

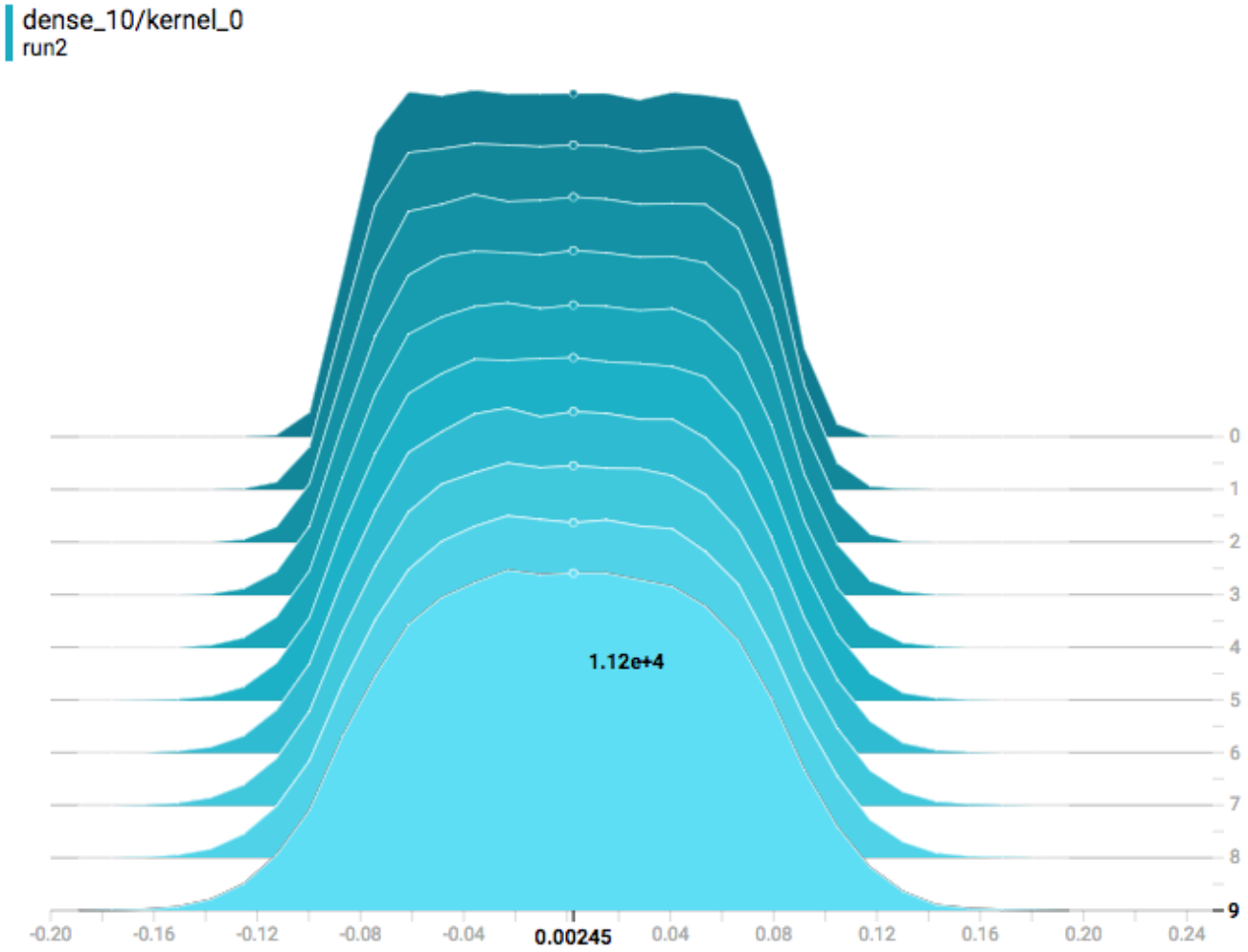
Input Layer - Dense_1

Here is the distribution of the bias for the input layer (dense_1):



The Y axis shows the the training epoch and the X axis shows the weight. The number highlighted on the chart shows the number of elements in the layer with that weight.

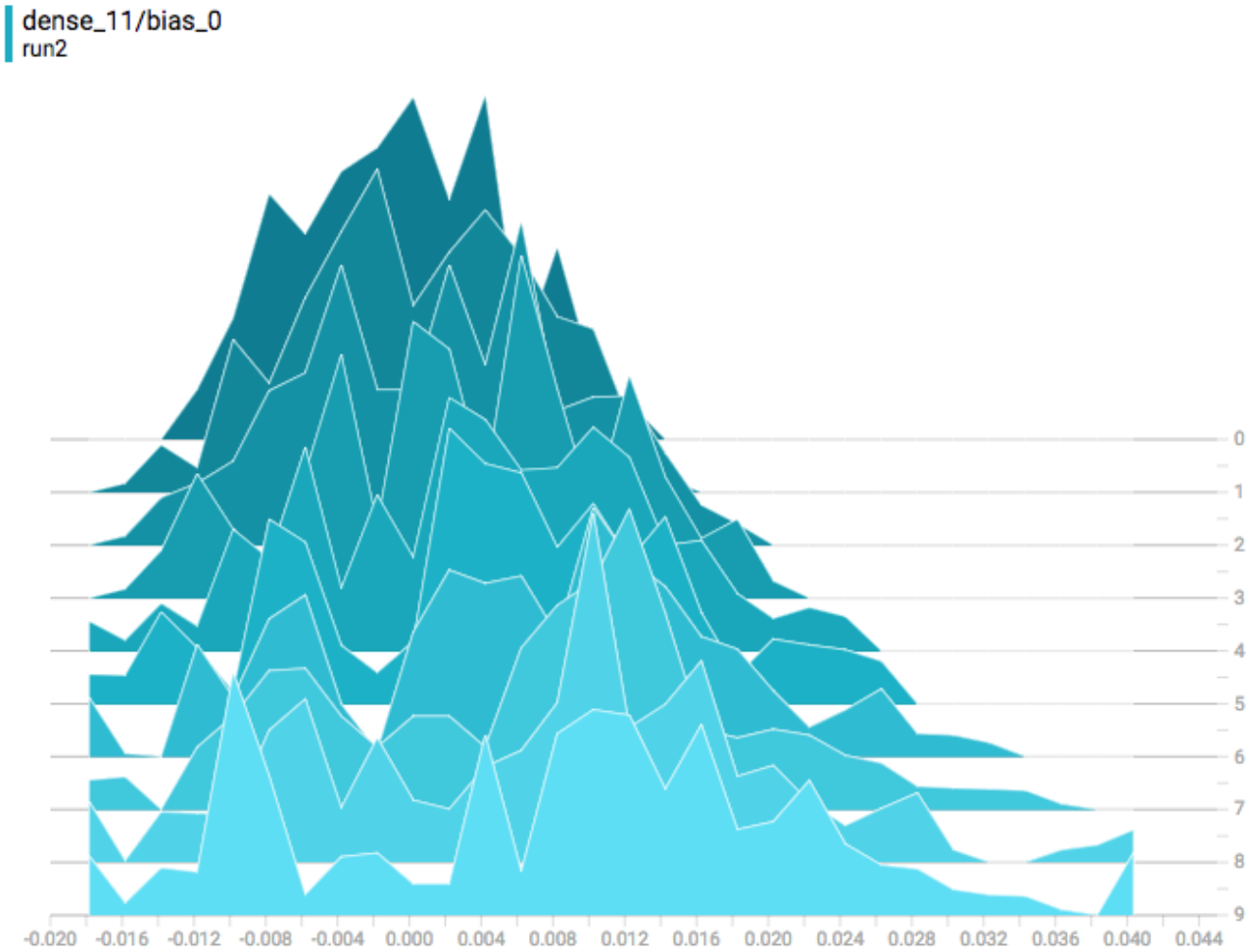
Here is the evolution of weights in the input layer over the training runs:



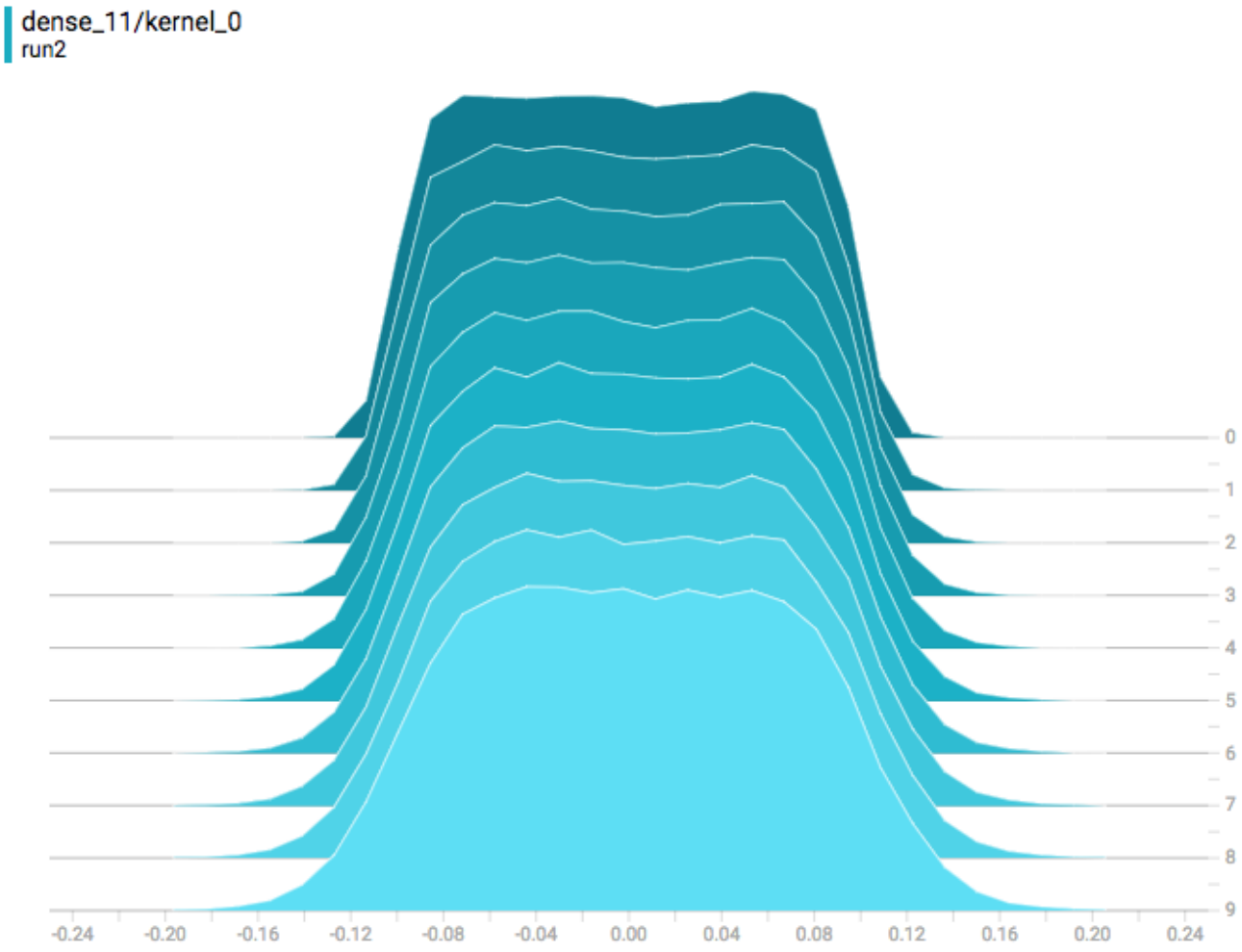
We can see that the weights are normally distributed around a zero mean, ranging from approx -0.16 to 0.16 in this layer.

Fully Connectecd Layer - Dense_2

Here is the evolution of the bias for the second fully connected layer:



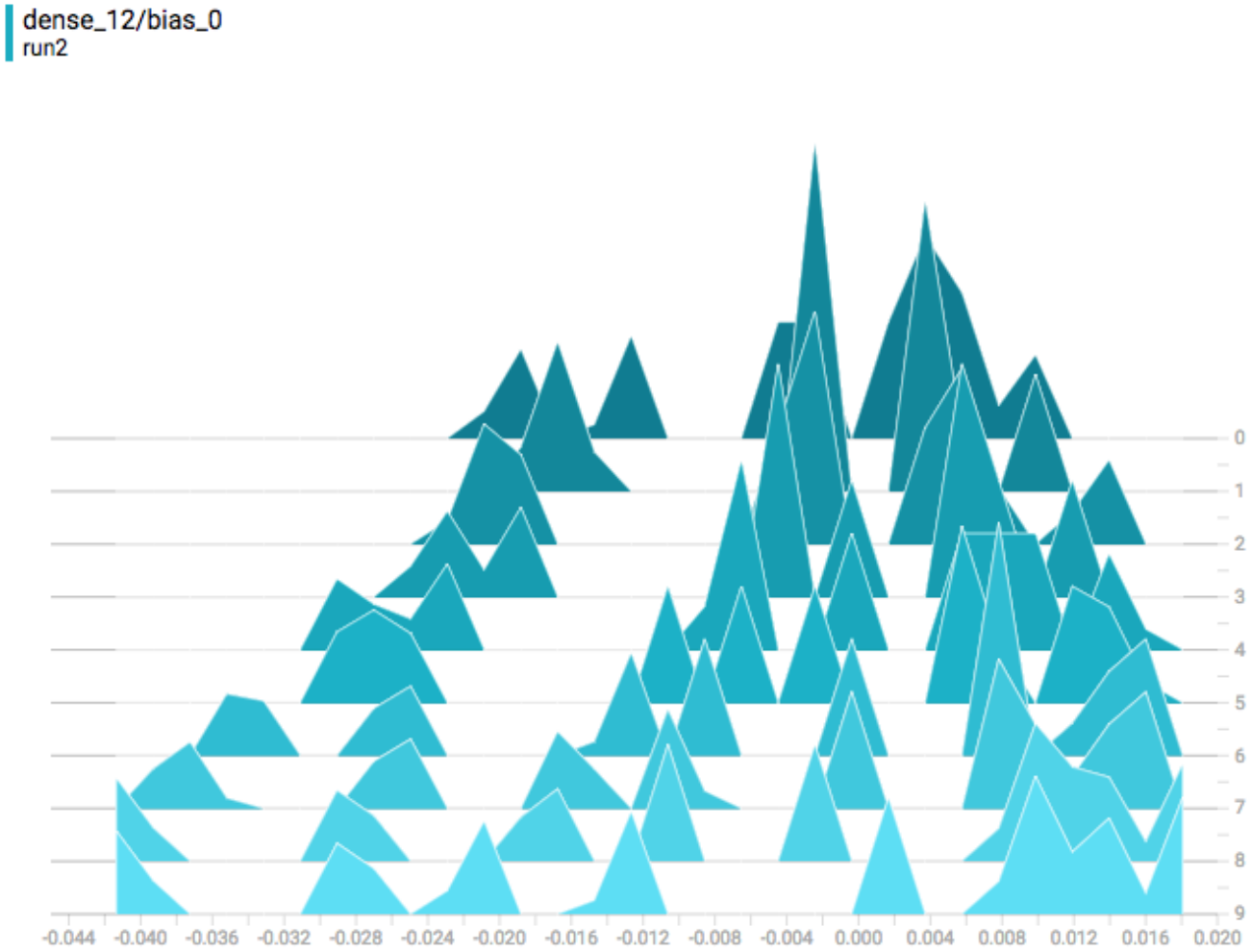
We can see that the variance of the bias increased over the training runs and became less normally distributed.



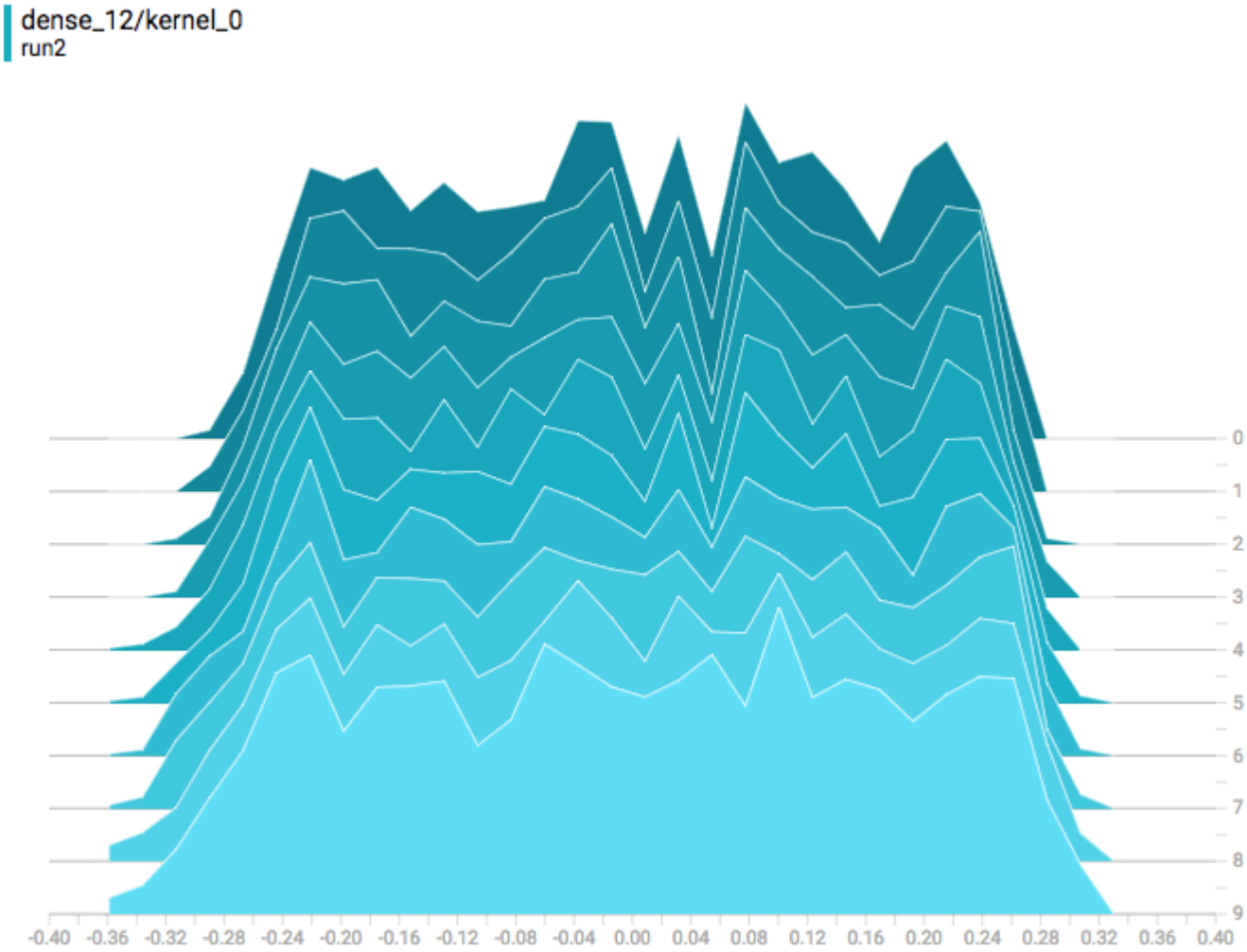
The weights for the second layer are similar to the input layer: normally distributed between approximately -0.16 and 0.16. The evolution of these weights seemed to be limited over the training runs.

Output Layer - Dense_3

Here is the evolution of the bias weights for the output layer:



Note the increasing variance of the bias over the training runs.



Finally, we observe the distribution of the weights in the output layer. The range of the distribution is from -0.36 to ~0.32. Note the moderation of the frequency of elements at 0.04 over the training runs, which was highly pronounced at the initial epoch.

5.2 Reflection

Given that this project was the first that I had attempted involving a significant amount of natural language processing I am satisfied with the results, however, there is much work that could be done to improve the accuracy of this task.

Looking at the submissions from other teams on Kaggle, experts have been able to reach an MLL of 0.43269 (without using the leaked label data for the test set).

Initially I assumed that this task would be primarily about tuning Keras model parameters, but it quickly became apparent that it was largely about extracting meaning from a corpus of text.

This task also showed me that I have a great deal to learn about the practicalities of applying machine learning to real world problems. This is certainly something I hope to improve on by participating in other competitions and hopefully through work assignments in future.

5.3 Improvement

If I had access to more computational power, rather than using my laptop CPU, I would have liked to have spent more time:

- Performing many more training iterations with Doc2Vec
- Experimenting with various sets of stop words
- Using vectors trained on scientific text corpus
- Using alternate NLP packages such as Spacy
- Tuning various parameters for Doc2Vec - including varying the number of dimensions from 300
- Featurizing both the Gene and Variation features using one hot encoding
- Exploring many other classification algorithms including basic ones such as linear regression all the way through to XGBoost which seemed to be delivering the highest performance on this task

5.3 Acknowledgments

I would particularly like to thank the other teams involved in this competition for ideas on which directions to improve, help with the EDA and various code snippets which I was able to incorporate into my analysis.

6. References

Kaggle Competition Site (https://www.kaggle.com/c/msk-redefining-cancer-treatment)
Distributed Representation of Sentences and Documents (https://arxiv.org/abs/1405.4053)
Efficient Estimation of Word Representation in Vector Space (https://arxiv.org/abs/1301.3781)
A gentle introduction to Doc2Vec (https://medium.com/towards-data-science/a-gentle-introduction-to-doc2vec-db3e8c0cce5e)
Word2Vec Tutorial (http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/)
OncoKB (http://oncokb.org)
Making Sense of Logarithmic Loss (http://www.exegetic.biz/blog/2015/12/making-sense-logarithmic-loss/)
Using Keras Deep Learning Models with Scikit Learn (https://machinelearningmastery.com/use-keras-deep-learning-models-scikit-learn-python/)
How to improve deep learning performance (https://machinelearningmastery.com/improve-deep-learning-performance/)
Doc2Vec in a simple way (https://medium.com/@mishra.thedeepak/doc2vec-in-a-simple-way-fa80bfe81104)
Convolutional Neural Networks for Sentence Classification (http://www.aclweb.org/anthology/D14-1181)
Deep Learning - Wikipedia (https://en.wikipedia.org/wiki/Deep_learning)
A word is worth a thousand vectors (http://multithreaded.stitchfix.com/blog/2015/03/11/word-is-worth-a-thousand-vectors/)
Decomposing signals in components - LSA (http://scikit-learn.org/stable/modules/decomposition.html#lsa)
How does Doc2Vec represent the feature vector of a document (https://www.quora.com/How-does-doc2vec-represent-feature-vector-of-a-document-Can-anyone-explain-mathematically-how-the-process-is-done) Doc2Vec tutorial using Gensim (https://medium.com/@klintcho/doc2vec-tutorial-using-gensim-ab3ac03d3a1)
Doc2Vec Tutorial - Rare Technologies (https://rare-technologies.com/doc2vec-tutorial/)
Doc2Vec Tutorial - Gensim Github (https://github.com/RaRe-Technologies/gensim/blob/develop/tutorials.md#tutorials)
Sentiment Analysis Using Doc2Vec (https://linanqiu.github.io/2015/10/07/word2vec-sentiment/)
Keras (http://keras.io)
Scikit Learn (http://scikit-learn.org/stable/index.html)
TensorBoard Histograms (https://www.tensorflow.org/get_started/tensorboard_histograms)