

Predicting Cancer Types based on Genetics Expression

Kevin Hu

1. Introduction

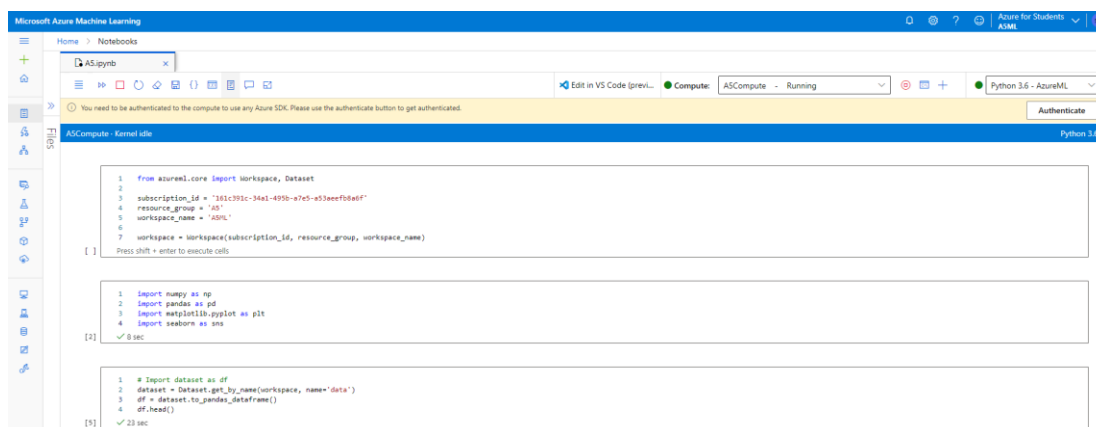
This report is part of the assignment for the Big Data course at the University of Toronto. For this project, I'll be working on the "Gene expression cancer RNA-Seq Data Set" provided by the UCI Machine Learning Repository¹. The goal of this dataset is to come up with a classification model to label the type of cancer cells based on the expression levels of their genomes. The problem is that the samples in this dataset has very high number of dimensions (over 20K). To combat this, I will be reducing dimension using projection method (or PCA). In terms of classifying the right type of tumor, I will be using traditional classifying methods available in the Azure Machine Learning toolkit. These can either be multi-class neural networks or logistic regression.

2. Data & Data Cleaning

The data I work with comes in two files, genome expression data and a label data². There are 20,531 attributes (i.e. genomes) with different level of expression in positive real number. For the label, there is only one column with five different types of cancerous tumors. These include PRAD (prostate), LUAD (lung), BRCA (breast), KIRC (kidney) and COAD (colon).

Although there are over 20K attributes from this dataset, there is only 801 instances. This is good for the assignment because we can run machine learning models that will most likely not exceed the education credit that Microsoft gives us. Of course, large number of attributes is a major issue, but I have a plan to deal with this which will be discussed below.

Before I present the data explanation outputs, I would like to show how I set up the workflow first. First, I created a workspace with Machine Learning studio inside. I then uploaded my datasets (i.e. data and label that were separately given) into Azure Machine learning. As shown from the screenshot below, I used the following codes to run them.



```
1 from azureml.core import Workspace, Dataset
2
3 subscription_id = "31c393c-34e1-495b-a7e5-a53eeef0ad0f"
4 resource_group = "RG"
5 workspace_name = "JOPS"
6
7 workspace = Workspace(subscription_id, resource_group, workspace_name)

[1] Press shift + enter to execute cells

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns

[2] ✓ 0 sec

1 # Import dataset as df
2 dataset = Dataset.get_by_name(workspace, name='data')
3 df = dataset.to_pandas_dataframe()
4 df.head()

[3] ✓ 23 sec
```

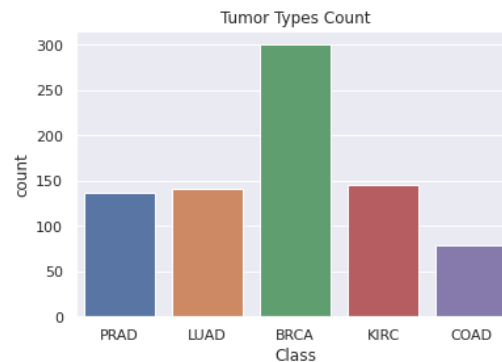
¹ <https://archive.ics.uci.edu/ml/datasets.php>

² <https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq>

3. Exploratory Data Analysis

3.1 Tumor types Analysis

According to my plot below, the most frequent types of gene is breast cancer. The least frequent type is Colon Adenocarcinoma.



3.2 Most Expressive Genes

It is difficult for my case to summarize explanatory variates in my dataset. This is because there are 20,531 columns of genes with different level of expressions. To get a meaningful summarization, I sum all expressions across all samples for each gene using the code below. I plot the top 10 most expressive genes as a result.

Gene	Expression Sum	Percent of Total
gene_230	13,160.78	0.01240%
gene_5380	13,121.95	0.01240%
gene_232	12,790.36	0.01210%
gene_18570	12,637.98	0.01190%
gene_6857	12,587.38	0.01190%
gene_5388	12,542.59	0.01180%
gene_1322	12,407.19	0.01170%
gene_6698	12,300.69	0.01160%
gene_15242	12,144.09	0.01150%
gene_3371	12,070.59	0.01140%

The most expressive gene is the 230th one. The percentage of total expressiveness is 0.000124. On average, I'd have expected the percentage of total to be $1/20,531 = 0.000048707$. Taking the ratio of these two numbers, I found that the most expressive gene by sum is 2.6 times more expressive than average. I conclude there are not many overly expressive genes from the data.

3.3 Most Expressive Samples

In a similar exercise as the prior one, I'm summing all expression levels among each sample instead of each gene. Unfortunately, we are not given more details on what each expression is (i.e. cell, tumor sample, etc.) so we will assume that they are biological sample collected from humans. The code is very similar as prior plot, where we sum by rows instead of by columns. Once again, I ranked the most expressive genes.

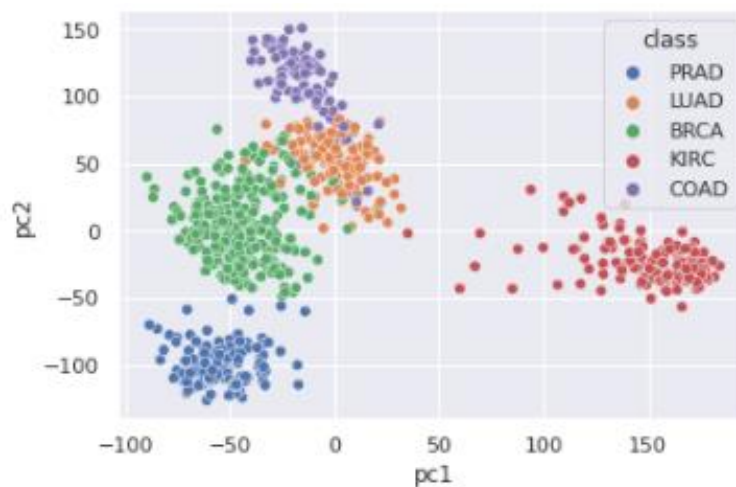
There are 801 instances in total, so the average expression level of each sample should be $1/801 = 0.00125$ of total. Based on my table below, the highest expression level is 0.001371, which is only 1.10 times over the average. Thus, based on my analysis below there is no specific sample that is overwhelmingly expressive.

Sample	Expression Sum	Percent of Total
798	145,246.95	0.13710%
190	141,222.45	0.13330%
218	140,644.24	0.13270%
419	140,222.26	0.13230%
318	139,276.15	0.13140%
675	139,079.83	0.13130%
37	139,017.24	0.13120%
635	138,933.92	0.13110%
523	138,773.57	0.13100%
513	138,691.35	0.13090%

3.4 Principal Components Analysis

Unfortunately, this is probably as far as we could go with original data that has over 20K attributes. To plot more meaningful graphs, I decided to reduce the dimensions by projecting the data frame via principal components analysis. Please see the screenshot, queries, and scatterplot below. Fortunately, the scatterplot shows that PCA works very well with this dataset.

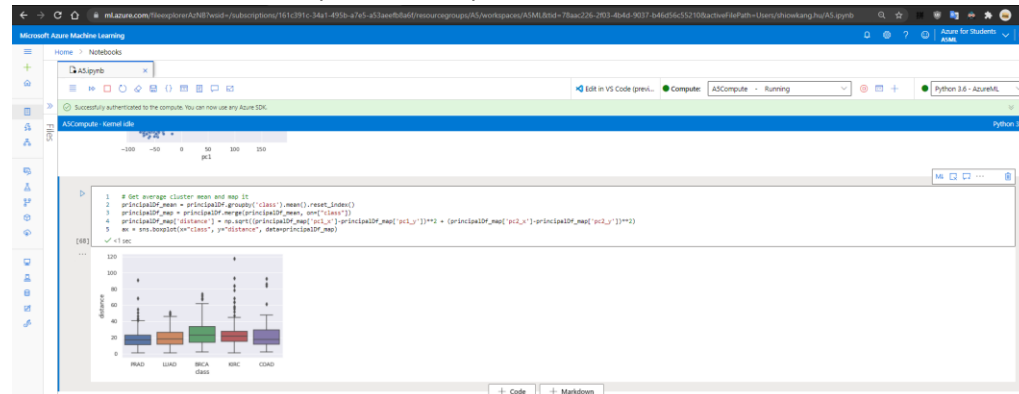
Sample	pc1	pc2	class
0	-62.7554	-94.072	PRAD
1	-2.4329	90.58584	LUAD
2	-71.2669	-8.06461	PRAD
3	-84.7708	-73.2446	PRAD
4	-69.5602	-9.61294	BRCA



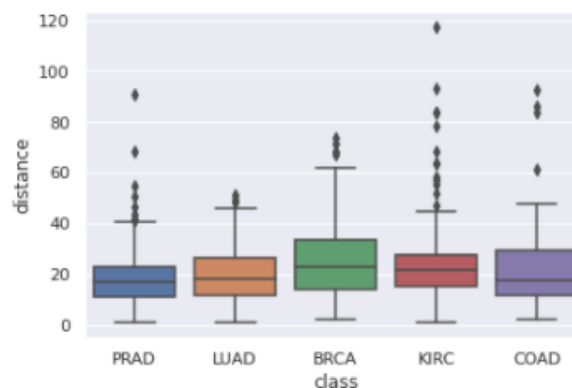
Based on my observation above, kidney cancer samples (KIRC in red) has the most distinct principal components comparing to the rest of the samples.

3.5 PCA's distance from cluster

We want to see how far apart the samples from PCA's cluster center are. Please see below:



I started by calculating cluster mean using pandas. Then I looked up each sample with their cluster center and calculate the Euclidean distance from center (i.e. $\sqrt{(x1 - \text{mean}(x1))^2 + (x2 - \text{mean}(x2))^2}$). Lastly, I created a boxplot with each cancer type as x-axis and distance as y-axis. Unsurprisingly, kidney samples (KIRC) have the most volatile distance as shown in boxplot below (but we can already see from the scatterplot above). Although I was surprised to see that kidney samples have the thinnest box plots covering 75% of its data. Similarly, the least volatile principal components genome expression is lung cancer (LUAD).



4. Python Models

To produce enough exploratory outputs (5 plots and tables), I needed to pre-process my data already using PCA. To complete this question, I will explain my rationale again here. The reason I use PCA is because I have over 20K variables, and it is very difficult to work with in Azure ML Pipeline (i.e. I can't even preview the data because there are too many variables). I decided to apply PCA to project my data into two variables because I can graph them into in 2D plot. You may refer to plot 4 and 5 of the prior question to see the PCA python query and its result.

In terms of data cleanliness, it was already ready for analysis. As mentioned by the instructor in the final lecture, I will compensate for the lack of data cleaning task by implementing an additional model (3

instead of 2) in Azure ML pipeline. The models that I will be implementing are neural networks and logistic regressions. I choose these models because they are well suited for classifying categorized data.

4.1 Neural Network

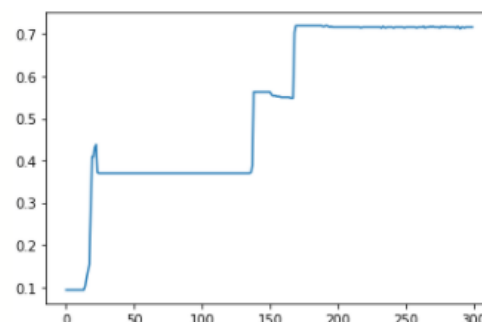
I would like to learn new Python package so I'm starting with TensorFlow's Keras for neural network. The output table can be seen below:

	pc1	pc2	class
0	-62.755415	-94.071974	PRAD
1	-2.432896	90.585842	LUAD
2	-71.266853	-8.064607	PRAD
3	-84.770785	-73.244566	PRAD
4	-69.560171	-9.612940	BRCA

Next, we will need to split our data into training and testing set. I also performed one-hot-encoding on the label axis. Our train/test split is 70%/30% of the data.

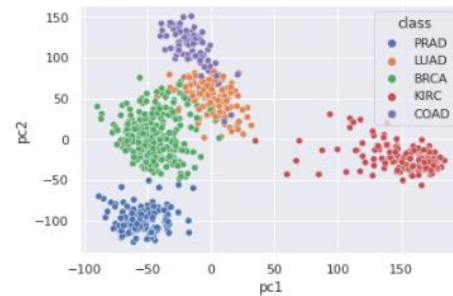
	BRCA	COAD	KIRC	LUAD	PRAD
207	0	0	0	0	1
329	1	0	0	0	0
199	0	0	0	1	0
583	0	0	1	0	0
618	0	1	0	0	0

Next, we import TensorFlow and Keras, and define our neural network as shown below. There are two hidden layers with 10 and 7 hidden nodes respectively. I specified the model to use Stochastic Gradient Descent with learning rate of 0.01. I used relu activation function for the hidden layers except the last one (it has to be sigmoid activation function). I specified 300 epochs for the training phrase. After running the model, I graphed the accuracy curve from training set by epochs below:



The accuracy was originally very low but increased before it converges at around the 200th epoch. I used the query below to calculate the testing set accuracy rate of 66%, which is like the training set accuracy of the final model. The second metric is confusion matrix:

	BRCA	COAD	KIRC	LUAD	PRAD
0	89	0	0	0	4
1	26	0	0	0	0
2	0	0	38	0	0
3	44	0	3	0	0
4	4	0	0	0	33



It is easy to see that the model has problem classifying colon and lung cancers (COAD and LUAD respectively). It is obvious when comparing it to the PCA scatterplot from prior question (on the right), as lung (LUAD), colon (COAD) and breast (BRCA) tumors are very close by. I tried running the model with different settings (i.e. higher epochs, more hidden layers and nodes and different learning rates), but all model outputs will always misclassify either colon (COAD) or lung (LUAD) tumors. Thus, I left the model at this point and continue with the second algorithm.

4.2 Multiclass Logistic Regression

Multiclass Logistic Regression

The model accuracy on the testing set can be calculated below. I received 97% accuracy, which I considered impressive. Especially since the whole algorithm is much faster comparing to neural network.

The confusion matrix can be calculated below.

	BRCA	COAD	KIRC	LUAD	PRAD
0	93	0	0	0	0
1	0	22	0	4	0
2	0	0	38	0	0
3	2	2	0	43	0
4	0	0	0	0	37

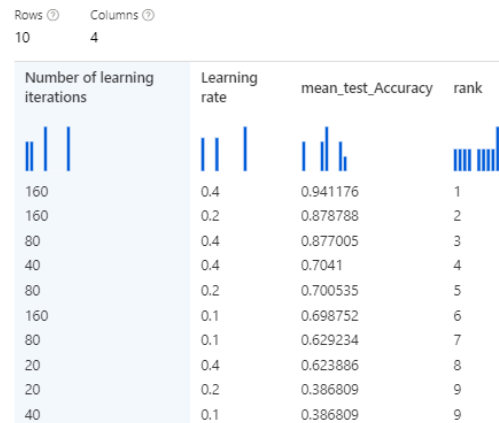
Observe that the logistic regression has no problem classifying more closely clustered tumors, which neural network had issue with. Given the results above, I picked logistic regression over neural network as the model of choice based on speed, ease of usage and accuracy.

5. MS Azure Machine Learning Pipelines

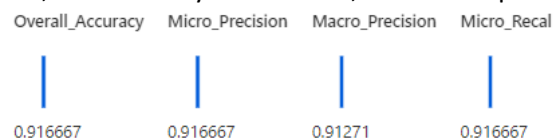
Now that we can use MS Azure ML pipeline, I would like to validate my algorithm in prior question using this tool too.

5.1 Neural Network

This was my first model of choice because neural network is a natural fit for detecting pattern in a classification setting. I turned my model with 10 hyperparameters with the results shown below. To minimize the cost of running model, I specified 5 hidden nodes across all 10 neural networks models:



As show above, the best combination best on hyperparameter tuning is with 160 number of iterations and the earning rate of 0.4. The mean test accuracy is 94% which is impressive (given that the data was already in principal components and there are only 5 hidden nodes). We proceeded with this model. Upon evaluating it with the testing set, the accuracy level is 92%, which is impressive as well.

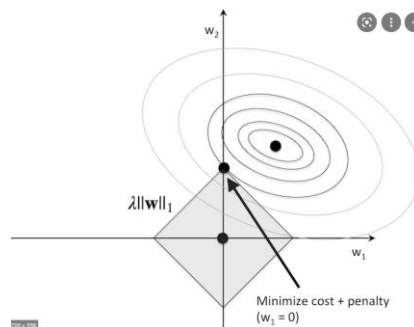


The confusion matrix of the optimized model is shown below. We see that prostate (PRAD) and kidney (KIRC) cancers are predicted almost always accurately (98% and 100%). This is not surprising to me because when we graphed PCA scatterplot, they are clearly separated from other types of tumors (i.e. red and blue dots in the right graph below, same graph as prior question). For other types of tumors, the scatterplots are more mixed. Of course, we sacrificed some data accuracy when we project them using PCA (from 20K to 2 attributes). But without PCA, we wouldn't even be able to visualize the data to begin with.



5.2 Multi-Class Logistic Regression

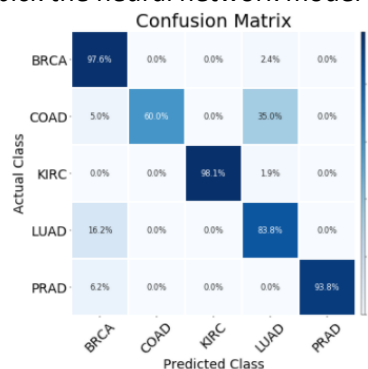
I was motivated to use this model because it is closely aligned to my actual work outside of class. Personally, I would not expect logistic regression to work as well given that it will produce a linear decision boundary. However, the concept is simple to understand so it is worth a try. I also want to note that I did not tune parameters for logistic regression since the only parameter worth tuning was a regularization factor. Since it was already set to L1 (as shown in the picture below), I was already satisfied with the default setting.



Surprisingly, I find the testing set accuracy to be as good as the neural network at 92%:



Judging from the confusion matrix below, logistic regression also does a very good job at predicting prostate, breast and kidney tumors. Recall that these are clearly separated from our PCA scatterplot. However, it has problem labeling colon and lung cancers (only 60% accuracy for colon cancer). Given that this is a major miscalculation, I will pick the neural network model over logistic regression for now.



5.3 Random Forest

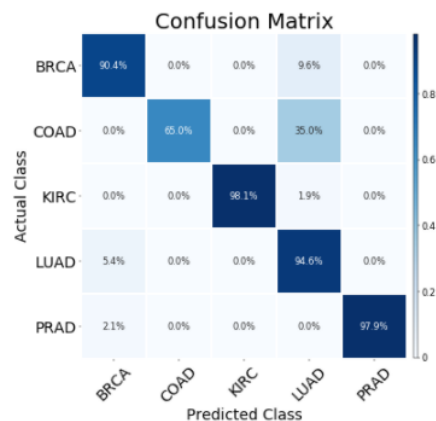
I would also like to try an extra model to compensate for not needing to clean my data (except performing PCA). I decided to tune the hyperparameters because random forest can quite vary (by number of decision trees, minimum sample per node and tree depths). Please see my tuning results ranked below:

Number of decision trees	Minimum number of samples per leaf node	Maximum depth of the decision trees	mean_test_Accuracy	rank
32	4	16	0.960784	1
32	4	64	0.960784	1
8	4	16	0.955437	3
32	1	16	0.948307	4
8	1	64	0.948307	4
32	1	64	0.948307	4
32	16	16	0.939394	7
1	16	64	0.88057	8
32	1	1	0.809269	9
32	4	1	0.809269	9

As shown above, the best model has 32 decision trees, 4 samples per leaf node, and maximum depth of 16. The training accuracy is impressive at 97%. I proceeded with the best model and tested it with the testing set. I received 92% accuracy as a result. Although random forest is the most accurate out of the 3 models I tried, it is not much better than other two.

Overall_Accuracy	Micro_Precision	Macro_Precision	Micro_Recall	Macro_Recall
0.920833	0.920833	0.929563	0.920833	0.891899

The confusion matrix is quite like the other two models too as shown below. Because prostate, breast and lung cancers are easy to classify (see PCA scatterplot), we shift our focus to the colon and lung cancer. The accuracy of colon cancer classification is 65%, which is better than logistic regression's 60% but worse than neural network's 70%.



6. Conclusion

Based on the three models above, I selected the (tuned) neural network as the model of choice. Although it does not have the highest testing set accuracy, it performs best when predicting colon cancer (COAD), which is the hardest samples to draw decision boundary on. If I were to have more time and resources, I would add more dimensions when performing PCA (i.e. having 3 to 5 components). That will probably make the attributes between colon (COAD) and lung (LUAD) tumors more separated. However, given that I already reduced the dimensions from 20K attributes to 2 and the testing set accuracy is already over 90%, I am quite happy with the results.

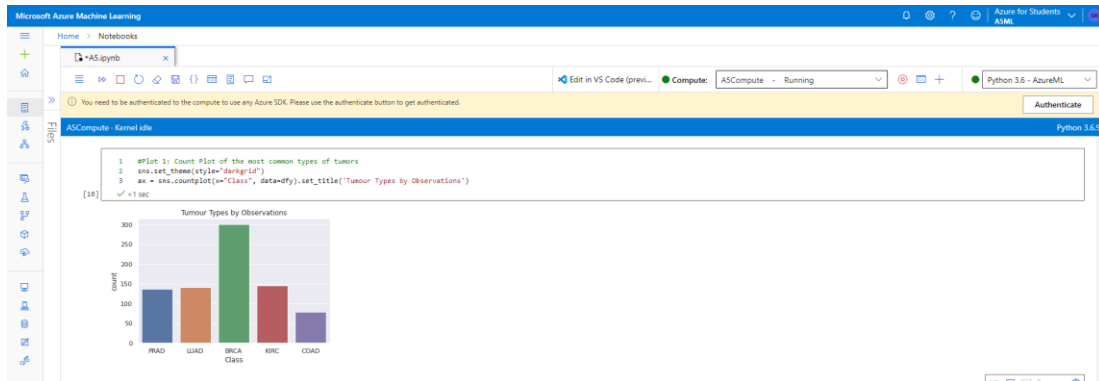
Appendix I – Data Loading Python Code

The query below sets up MS Azure workspace and load in genetics dataset.

```
from azureml.core import Workspace, Dataset
subscription_id = '161c391c-34a1-495b-a7e5-a53aeefb8a6f'
resource_group = 'A5'
workspace_name = 'A5ML'
workspace = Workspace(subscription_id, resource_group, workspace_name)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Import dataset as df
dataset = Dataset.get_by_name(workspace, name='data')
df = dataset.to_pandas_dataframe()
df.head()
# Import label as dfy
datasety = Dataset.get_by_name(workspace, name='label')
dfy = datasety.to_pandas_dataframe()
dfy.head()
# Drop column 2 from both data and label dataframes
df = df.drop(['Column2'], axis=1)
dfy = dfy.drop(['Column2'], axis=1)
# Check the dataset again if that works
df.head()
# Check the label as well
dfy.head()
```

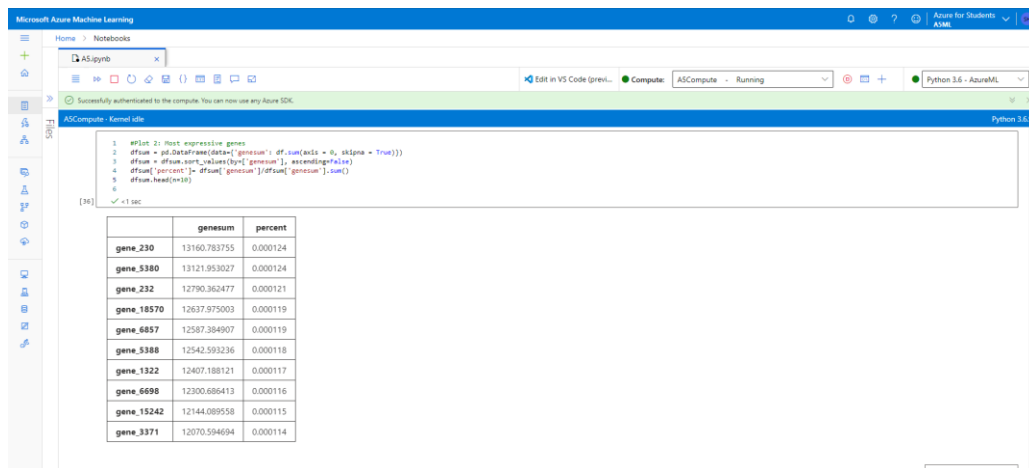
Appendix II – Exploratory Data Analysis Codes

Please see below for figure 1:



```
sns.set_theme(style="darkgrid")
ax = sns.countplot(x="Class", data=df).set_title('Tumor Types Count')
```

Please see below for figure 4:



```
dfsum = pd.DataFrame(data={'genesum': df.sum(axis = 0, skipna = True)})
dfsum = dfsum.sort_values(by=['genesum'], ascending=False)
dfsum['percent'] = dfsum['genesum']/dfsum['genesum'].sum()
dfsum.head(n=10)
```

Please see below for figure 5:

```
# Get average cluster mean and map it
principalDf_mean = principalDf.groupby('class').mean().reset_index()
principalDf_map = principalDf.merge(principalDf_mean, on=["class"])
principalDf_map['distance'] = np.sqrt((principalDf_map['pc1_x']-
principalDf_map['pc1_y'])*2 + (principalDf_map['pc2_x']-principalDf_map['pc2_y'])*2)
ax = sns.boxplot(x="class", y="distance", data=principalDf_map)
```

Appendix III – Python Model Codes

Please see neural network codes below. First, I imported the packages, set up the workspace and import the scrubbed PCA data as shown below.

```
# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from azureml.core import Workspace, Dataset
# Create workspace
subscription_id = '161c391c-34a1-495b-a7e5-a53aeefb8a6f'
resource_group = 'A5'
workspace_name = 'A5ML'
workspace = Workspace(subscription_id, resource_group, workspace_name)
# Import pca projected dataframe
dataset = Dataset.get_by_name(workspace, name='pcdf')
df = dataset.to_pandas_dataframe()
df.head()
# Train test split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow import keras
X = df.iloc[:,0:2]
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=5)
# One hot encoding
y_train_dummy = pd.get_dummies(y_train)
y_test_dummy = pd.get_dummies(y_test)
y_test_dummy.head()

import tensorflow as tf
from tensorflow import keras
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(2,)), activation='relu'),
    keras.layers.Dense(7, activation='relu'),
    keras.layers.Dense(5, activation='sigmoid')])
opt = tf.keras.optimizers.SGD(learning_rate=0.01)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(X_train, y_train_dummy, epochs=300)
```

```

1 # Get prediction column
2
3 # Get calculation probabilities dataframe
4 y_pred = model.predict(X_test)
5 y_pred_dummy = y_test_dummy.iloc[0:0]
6 y_pred_dummy = y_pred_dummy.append(pd.DataFrame(y_pred, columns=y_pred_dummy.columns, ignore_index=True))
7
8
9 # Get the label column
10 y_test_pred = y_pred_dummy.idxmax(axis=1)

```

[137] ✓ <1 sec

+ Code + Markdown

```

1 # Get prediction accuracy
2 sum(1 for x,y in zip(pd.Series.tolist(y_test), pd.Series.tolist(y_test_pred)) if x == y) / len(pd.Series.tolist(y_test))

```

[138] ✓ <1 sec

0.6639804149377593

```

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
# Get confusion matrix
from sklearn.metrics import confusion_matrix
y_pred_dummy = y_test_dummy.iloc[0:0]
nn_confusion_matrix = y_pred_dummy.append(pd.DataFrame(confusion_matrix(y_test, y_test_pred),
columns=y_pred_dummy.columns, ignore_index=True))
nn_confusion_matrix

```

Please see logistic regression codes below. Unlike neural network, I can construct multiclass logistic regression model directly using scikit-learn package using the code below:

```

# Train Logistic Regression
from sklearn.linear_model import LogisticRegression
model_mlr = LogisticRegression(random_state=0).fit(X_train, y_train)
# Predict & Accuracy calculation
y_pred_test_mlr = model_mlr.predict(X_test)
sum(1 for x,y in zip(pd.Series.tolist(y_test), y_pred_test_mlr) if x == y) / len(pd.Series.tolist(y_test))

```

```

1 # Predict & Accuracy calculation
2 y_pred_test_mlr = model_mlr.predict(X_test)
3 sum(1 for x,y in zip(pd.Series.tolist(y_test), y_pred_test_mlr) if x == y) / len(pd.Series.tolist(y_test))

```

153] ✓ <1 sec

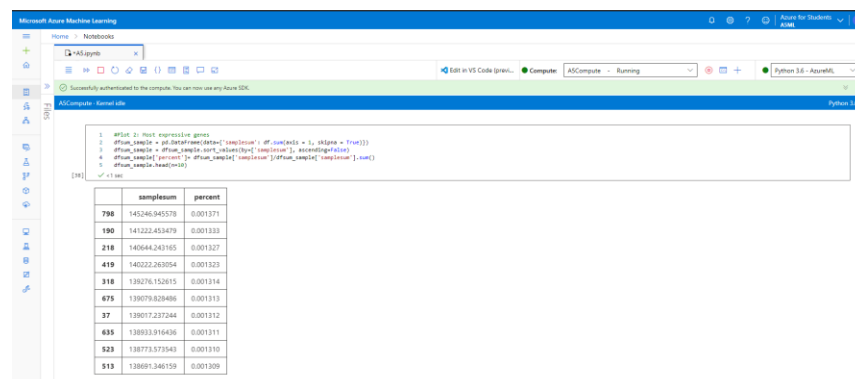
... 0.966804979253112

```

# Get confusion matrix for multiclass logistic regression
nn_confusion_matrix_mlr = y_pred_dummy.append(pd.DataFrame(confusion_matrix(y_test, y_pred_test_mlr),
columns=y_pred_dummy.columns, ignore_index=True))
nn_confusion_matrix_mlr

```

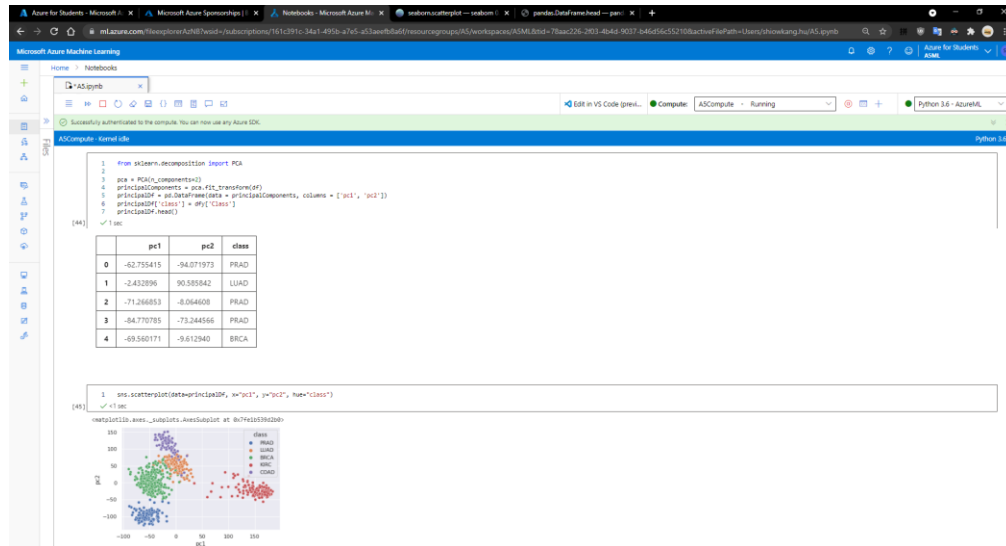
Please see below for figure 3:



#Plot 3: Most expressive samples

```
dfsum_sample = pd.DataFrame(data={'samplesum': df.sum(axis = 1, skipna = True)})
dfsum_sample = dfsum_sample.sort_values(by='samplesum', ascending=False)
dfsum_sample['percent'] = dfsum_sample['samplesum']/dfsum_sample['samplesum'].sum()
dfsum_sample.head(n=10)
```

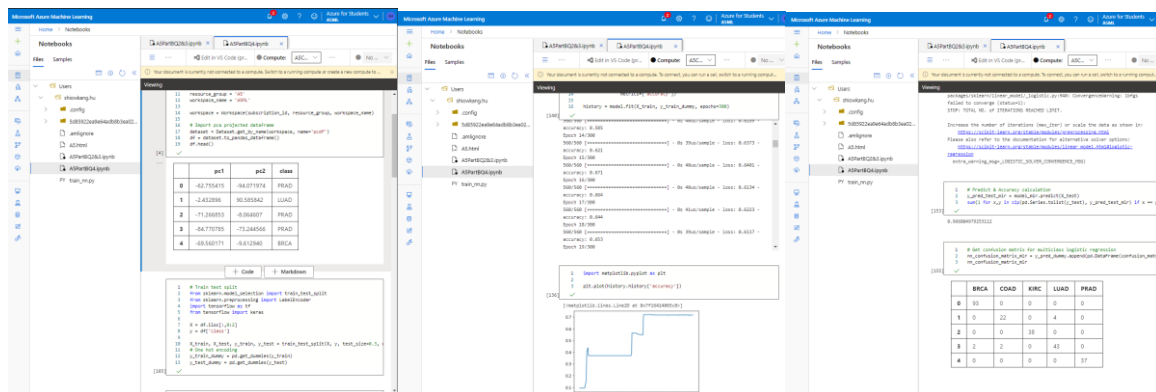
Please see below for figure 4:



```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(df)
principalDf = pd.DataFrame(data = principalComponents, columns = ['pc1', 'pc2'])
principalDf['class'] = df['Class']
principalDf.head()

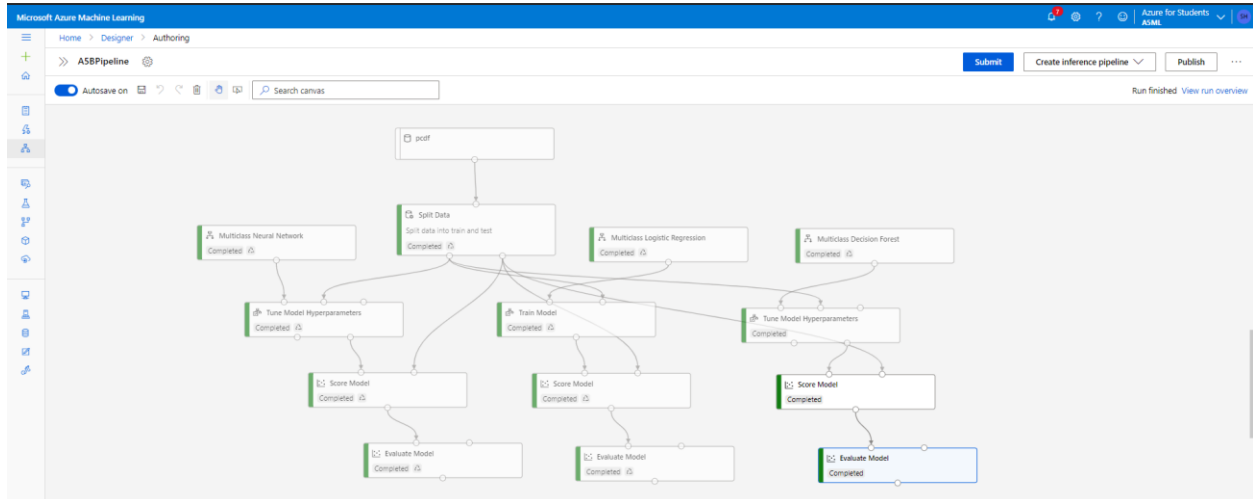
sns.scatterplot(data=principalDf, x="pc1", y="pc2", hue="class")
```

As a proof, please see a few screenshots of this step below:

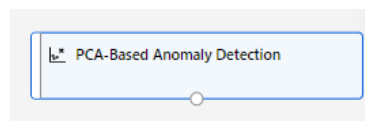


Appendix IV – MS Azure Pipeline Screenshots

Please see the screenshot of my overall pipelines below. There are three main branches corresponding to three ML models as mentioned in prior question (neural network, logistic regression, and random forest respectively).



I first started off with the principal component projected data frame (see part B-2 plot 4 on what the data frame looks like). I originally planned on applying PCA within the pipeline. However, PCA is only available as an Anomaly Detection tool (without any data input available as shown below).



Based on my finding, I could perform PCA by attaching it to the data directly in the old MS Azure ML studio. I'm unsure why this feature is taken out for my working version. Because of this, I used the PCA projected data that I wrote in Python Notebook (in part B-2 plot 4) as my starting point instead. I used the query below to save the data frame from Python notebook to Data Container.

```
datastore = workspace.get_default_datastore()
Dataset.Tabular.register_pandas_dataframe(principalDf,datastore,'pcdf')
```

Although I could have written a python query in pipeline too, the original data (without PCA) is quite large (i.e. 20K columns). It is impossible to get a preview in the pipeline interface, so I decided to start with the PCA projected one instead.

The following screenshots show the setting for each pipeline steps.

1. Split data into 70% training and 30% testing.

The screenshot displays the ASB Pipeline interface with a workflow diagram on the left and the 'Split Data' configuration panel on the right. The workflow includes steps for 'Split Data', 'Train Model', 'Score Model', and 'Evaluate Model' for three different models: Multiclass Logistic Regression, Multiclass Decision Forest, and Multiclass Neural Network. The 'Split Data' step is highlighted in blue.

Split Data

- Refresh
- Parameters: Outputs + logs, Details, Metrics, Child runs, ...
- Splitting mode: Split Rows
- Fraction of rows in the first output dataset: 0.7
- Randomized split: True
- Random seed: 0
- Stratified split: False
- Output settings
- Run settings
- Comment
- Module information

2. Tuning hyperparameters (same for neural network and random forest)

The screenshot displays the ASB Pipeline interface with a workflow diagram on the left and the 'Tune Model Hyperparameters' configuration panel on the right. The workflow includes steps for 'Split Data', 'Train Model', 'Score Model', and 'Evaluate Model' for three different models: Multiclass Logistic Regression, Multiclass Decision Forest, and Multiclass Neural Network. The 'Tune Model Hyperparameters' step is highlighted in blue.

Tune Model Hyperparameters

- Refresh
- Parameters: Outputs + logs, Details, Metrics, Child runs, ...
- Specify parameter sweeping mode: Random sweep
- Maximum number of runs on random sweep: 10
- Random seed: 0
- Label column: class
- Column names: class
- Metric for measuring performance for classification: Accuracy
- Metric for measuring performance for regression: Mean absolute error
- Output settings
- Run settings
- Comment
- Module information

3. Decision Forest parameters.

The screenshot displays the ASB Pipeline interface with a workflow diagram on the left and the 'Multiclass Decision Forest' configuration panel on the right. The workflow includes steps for 'Split Data', 'Train Model', 'Score Model', and 'Evaluate Model' for three different models: Multiclass Logistic Regression, Multiclass Decision Forest, and Multiclass Neural Network. The 'Multiclass Decision Forest' step is highlighted in blue.

Multiclass Decision Forest

- Refresh
- Parameters: Outputs + logs, Details, Metrics, Child runs, ...
- Create trainer mode: SingleParameter
- Number of decision trees: 8
- Maximum depth of the decision trees: 32
- Minimum number of samples per leaf node: 1
- Resampling method: Bagging resampling
- Output settings
- Run settings
- Comment
- Module information