Steps to Install Spark and Java to Google Cloud

```
!apt-get update
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q http://archive.apache.org/dist/spark/spark-3.1.2/spark-3.1.2-bin-hadoop2.7.t
!tar xf spark-3.1.2-bin-hadoop2.7.tgz
!pip install -q findspark
```

```
Hit:1 http://security.ubuntu.com/ubuntu bionic-security InRelease
Hit:2 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease
Ign:3 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64
Hit:4 http://archive.ubuntu.com/ubuntu bionic InRelease
Ign:5 https://developer.download.nvidia.com/compute/machine-learning/repos/ubunt
Hit:6 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64
Hit:7 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease
Hit:8 https://developer.download.nvidia.com/compute/machine-learning/repos/ubunt
Hit:9 http://archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:10 http://archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:11 http://ppa.launchpad.net/cran/libgit2/ubuntu bionic InRelease
Hit:12 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease
Hit:13 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease
Reading package lists... Done
```

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.2-bin-hadoop2.7"
```

```
import findspark
findspark.init()
from pyspark import SparkContext

sc = SparkContext.getOrCreate()
sc
```

SparkContext

Spark UI

Version
    v3.1.2
Master
    local[*]
AppName
    pyspark-shell

```
import pyspark
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.getOrCreate()
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
v3.1.2
Master
local[*]
AppName
pyspark-shell

Importing Numpy, Pandas and Matplotlib

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Printing Sample Data for all the three datasets(dog,human and chimpanzee)

```
human_gene_data = pd.read_table('human_gene_data.txt')
human_gene_data.head()
```

	sequence	class
0	ATGCCCCAACTAAATACTACCGTATGGCCCACCATAATTACCCCCA...	4
1	ATGAACGAAAATCTGTTCGCTTCATTCATTGCCCCCACAATCCTAG...	4
2	ATGTGTGGCATTTGGGCGCTGTTTGGCAGTGATGATTGCCTTTCTG...	3
3	ATGTGTGGCATTTGGGCGCTGTTTGGCAGTGATGATTGCCTTTCTG...	3
4	ATGCAACAGCATTTTGAATTTGAATACCAGACCAAAGTGGATGGTG...	3

```
chimp_gene_data = pd.read_table('chimp_gene_data.txt')
dog_gene_data = pd.read_table('dog_gene_data.txt')
chimp_gene_data.head()
```

	sequence	class
0	ATGCCCCAACTAAATACCGCCGTATGACCCACCATAATTACCCCCA...	4

```
dog_gene_data.head()
```

	sequence	class
0	ATGCCACAGCTAGATACATCCACCTGATTTATTATAATCTTTTCAA...	4
1	ATGAACGAAAATCTATTCGCTTCTTTCGCTGCCCCCTCAATAATAG...	4
2	ATGGAAACACCCTTCTACGGCGATGAGGCGCTGAGCGGCCTGGGCG...	6
3	ATGTGCACTAAAATGGAACAGCCCTTCTACCACGACGACTCATACG...	6
4	ATGAGCCGGCAGCTAAACAGAAGCCAGAACTGCTCCTTCAGTGACG...	0

Below Image shows what all gene families are present and to which class they belong, this code helps us classify the class of a gene based a set of test data

```
from IPython.display import Image
Image("genefamily.png")
```

Gene family	Number	Class label
G protein coupled receptors	531	0
Tyrosine kinase	534	1
Tyrosine phosphatase	349	2
Synthetase	672	3
Synthase	711	4
Ion channel	240	5
Transcription factor	1343	6

```
# Below function converts the sequence of strings into k-mer words of size 6(default
# these are also called hexamers
```

```
def getHexamers(seq, size=6):
    return [seq[i:i+size].lower() for i in range(len(seq) - size + 1)]


# for each and every sequence we get we are applying lambda function to call the getH
# and generate words of 6 letters
#sequence column is dropped and a new one words column is created
human_gene_data['words'] = human_gene_data.apply(lambda i: getHexamers(i['sequence'])
human_gene_data = human_gene_data.drop('sequence', axis=1)
human_gene_data.head()
```

	class	words
0	4	[atgccc, tgcccc, gcccca, ccccaa, cccaac, ccaac...
1	4	[atgaac, tgaacg, gaacga, aacgaa, acgaaa, cgaaa...
2	3	[atgtgt, tgtgtg, gtgtgg, tgtggc, gtggca, tggca...
3	3	[atgtgt, tgtgtg, gtgtgg, tgtggc, gtggca, tggca...
4	3	[atgcaa, tgcaac, gcaaca, caacag, aacagc, acagc...

```
# for each and every sequence we get we are applying lambda function to call the getH
# and generate words of 6 letters
#sequence column is dropped and a new one words column is created
chimp_gene_data['words'] = chimp_gene_data.apply(lambda i: getHexamers(i['sequence'])
chimp_gene_data = chimp_gene_data.drop('sequence', axis=1)
chimp_gene_data.head()
```

	class	words
0	4	[atgccc, tgcccc, gcccca, ccccaa, cccaac, ccaac...
1	4	[atgaac, tgaacg, gaacga, aacgaa, acgaaa, cgaaa...
2	4	[atggcc, tggcct, ggcctc, gcctcg, cctcgc, ctcgc...
3	4	[atggcc, tggcct, ggcctc, gcctcg, cctcgc, ctcgc...
4	6	[atgggc, tgggca, gggcag, ggcagc, gcagcg, cagcg...

```
# for each and every sequence we get we are applying lambda function to call the getH
# and generate words of 6 letters
#sequence column is dropped and a new one words column is created
dog_gene_data['words'] = dog_gene_data.apply(lambda i: getHexamers(i['sequence']), ax
dog_gene_data = dog_gene_data.drop('sequence', axis=1)
dog_gene_data.head()
```

	class	words
0	4	[atgcca, tgccac, gccaca, ccacag, cacagc, acagc...
1	4	[atgaac, tgaacg, gaacga, aacgaa, acgaaa, cgaaa...
2	6	[atggaa, tggaaa, ggaaac, gaaaca, aaacac, aacac...
3	6	[atgtgc, tgtgca, gtgcac, tgcact, gcacta, cacta...

we need to convert the lists of words for each gene into string of sentences.so that the count vectorizer can use. We will create a y variable to store the class labels
https://medium.com/@joshsungasong/natural-language-processing-count-vectorization-and-term-frequency-inverse-document-frequency-49d2156552c1

```
human_gene_texts = list(human_gene_data['words'])
for item in range(len(human_gene_texts)):
    human_gene_texts[item] = ' '.join(human_gene_texts[item])
y_data = human_gene_data.iloc[:, 0].values
```

```
print(human_gene_texts[2])
```

```
atgtgt tgtgtg gtgtgg tgtggc gtggca tggcat ggcatt gcattt catttg atttgg tttggg ttg
```

```
y_data
```

```
array([4, 4, 3, ..., 6, 6, 6])
```

```
chimp_gene_texts = list(chimp_gene_data['words'])
for item in range(len(chimp_gene_texts)):
    chimp_gene_texts[item] = ' '.join(chimp_gene_texts[item])
y_chimp = chimp_gene_data.iloc[:, 0].values
```

```
dog_gene_texts = list(dog_gene_data['words'])
for item in range(len(dog_gene_texts)):
    dog_gene_texts[item] = ' '.join(dog_gene_texts[item])
y_dog = dog_gene_data.iloc[:, 0].values
```

```
# CountVectorizer() is used to create a Bag of Words model
# https://medium.com/@joshsungasong/natural-language-processing-count-vectorization-a
# This is the same as k-mer counting.
# Used hit and trail approach to determine the n gram size of 4
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(ngram_range=(4,4))
X = cv.fit_transform(human_gene_texts)
```

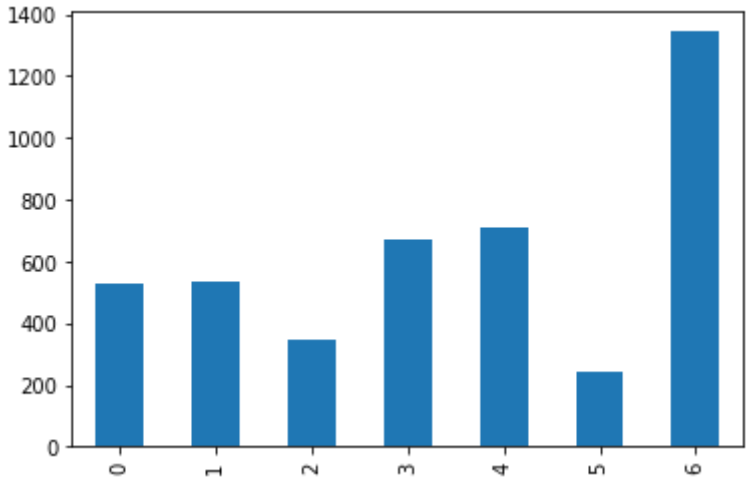
```
X_chimp = cv.transform(chimp_gene_texts)
X_dog = cv.transform(dog_gene_texts)

print(X.shape)
print(X_chimp.shape)
print(X_dog.shape)
```

```
(4380, 232414)
(1682, 232414)
(820, 232414)
```

```
#When we look at the class balance, we can see that
# the dataset is fairly balanced.
human_gene_data['class'].value_counts().sort_index().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3abeebc6d0>
```




```
# Preparing a training and test set for human dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y_data,
                                                    test_size = 0.20,
                                                    random_state=42)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(3504, 232414)
(876, 232414)
```

```
### Multinomial Naive Bayes Classifier ###
# The alpha parameter was determined by grid search previously
#https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digit
from sklearn.naive_bayes import MultinomialNB
```

```
classifier = MultinomialNB(alpha=0.1)
classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test)
```

We're receiving excellent results on our unseen data, so it doesn't appear that our model was overfit to the training data.

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
print("Confusion matrix\n")
print(pd.crosstab(pd.Series(y_test, name='Actual'), pd.Series(y_pred, name='Predicted
def get_metrics(y_test, y_predicted):
    accuracy = accuracy_score(y_test, y_predicted)
    precision = precision_score(y_test, y_predicted, average='weighted')
    recall = recall_score(y_test, y_predicted, average='weighted')
    f1 = f1_score(y_test, y_predicted, average='weighted')
    return accuracy, precision, recall, f1
accuracy, precision, recall, f1 = get_metrics(y_test, y_pred)
print("accuracy = %.3f \nprecision = %.3f \nrecall = %.3f \nf1 = %.3f" % (accuracy, p
```

```
Confusion matrix

Predicted   0    1   2    3    4   5    6
Actual
0          99    0   0    0    1   0    2
1           0  104   0    0    0   0    2
2           0    0  78    0    0   0    0
3           0    0   0  124    0   0    1
4           1    0   0    0  143   0    5
5           0    0   0    0    0  51    0
6           1    0   0    1    0   0  263
accuracy = 0.984
precision = 0.984
recall = 0.984
f1 = 0.984
```

✓ 0s completed at 8:34 PM