In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import seaborn as sns
from random import sample
from pyspark.mllib.stat import Statistics
from pyspark.ml import Pipeline
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import Imputer
from pyspark.ml.feature import Normalizer
from pyspark.ml.feature import StandardScaler
from pyspark.ml.feature import OneHotEncoderModel
from pyspark.ml.feature import StringIndexer, OneHotEncoderEstimator, Ve
ctorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import Normalizer
from pyspark.sql.functions import isnan, when, count, col
from pyspark.sql.types import IntegerType
from pyspark.sql.functions import log
from pyspark.ml.feature import PCA, PCAModel
from pyspark.ml.linalg import Vectors
```

In [2]:
```
# help functions

# function of convert RDD to Spark dataframe or create sample dataset

def create_dataframe(inputRDD):

    def parse(line):
        context = line.split('\t')
        yield(context)

    def empty_as_null(x):
        return when(col(x) != "", col(x)).otherwise(None)

    sampleData = inputRDD.flatMap(parse)\
                         .cache()

    indexDF = sampleData.toDF()

    index = [str(i) for i in range(len(indexDF.columns))]
    sampleData = sampleData.toDF(index)

  # function to fill empty space with null value
    for j in range(len(index)):
        sampleData = sampleData.withColumn(str(j), empty_as_null(str(j
)))

    return sampleData
```

In [3]:
```
# incorporate into function for data transformation
def data_transform(dataframe):

  # drop columns with many null value
    drop_col = ['1','10','12','32','33','35','38','39','16','17','20','23','24','25','26','28','29','31','34','37']
    toy_df_choose = dataframe.drop(*drop_col)

  # covert string to float for numeric values
    for i in toy_df_choose.columns[1:11]:
        toy_df_choose = toy_df_choose.withColumn(i, toy_df_choose[i].cast('float'))

  # run log transformations on numeric variables
    num_cols = ['2','3','4','5','6','7','8','9','11','13']

    log_num_toy = toy_df_choose
    for col in num_cols:
        log_num_toy = log_num_toy.withColumn("log_{}".format(col),log(toy_df_choose[col]+1))

    imputer = Imputer(inputCols=log_num_toy.columns[-10:],
                      outputCols=["{}_imputed".format(c) for c in log_num_toy.columns[-10:]])

    toy_df_impute = imputer.fit(log_num_toy).transform(log_num_toy)

  # fill null with '0' for categorical variables
    toy_df_impute = toy_df_impute.na.fill('0')

  # perform one-hot encoding
    toy_hot = toy_df_impute

    indexers = [StringIndexer(inputCol=col, outputCol="{0}_indexed".format(col)) for col in list(toy_hot.columns[12:20])]

    encoder = OneHotEncoderEstimator(inputCols=[indexer.getOutputCol() for indexer in indexers],
                                     outputCols=["{0}_encoded".format(indexer.getOutputCol()) for indexer in indexers])
    cat_assembler = VectorAssembler(inputCols=encoder.getOutputCols(),outputCol="CatFeatures")

    pipeline = Pipeline(stages=indexers + [encoder, cat_assembler])
    toy_onehot = pipeline.fit(toy_hot).transform(toy_hot)

  # Perfom standard normalization for categorical variables
    from pyspark.ml.feature import StandardScaler

    scaler = StandardScaler(inputCol="CatFeatures", outputCol="scaled_categorical",
                            withStd=True, withMean=True)

  # Normalize each feature to have unit standard deviation.
    scaler_toy_onehot = scaler.fit(toy_onehot).transform(toy_onehot)
```

```
  # assemble imputed log numerical variables to a new feature vector for
normalization in next step
    num_assembler = VectorAssembler(inputCols=['log_2_imputed','log_3_im
puted','log_4_imputed','log_5_imputed',
                                               'log_6_imputed','log_7_imput
ed','log_8_imputed','log_9_imputed',
                                               'log_11_imputed','log_13_imp
uted'],
                                    outputCol="NumFeatures")

    v_num_toy = num_assembler.transform(scaler_toy_onehot)

  # normalize numeric variables
    normalizer = Normalizer(inputCol="NumFeatures", outputCol="normed_lo
g_numeric")
    normed_num_toy = normalizer.transform(v_num_toy)

  # duplicate column 0 as "target", and convert its data type to "float"
    normed_num_toy = normed_num_toy.withColumn("target", normed_num_toy[
"0"].cast("float"))

  # assemble normalized log numerical variables and scaled categorical v
ariables to a new feature vector
  # select 'features' and 'target' to save as transformed toy df
    trans_assembler = VectorAssembler(inputCols=['normed_log_numeric','s
caled_categorical'],
                                      outputCol="features")
    trans_toy_df = trans_assembler.transform(normed_num_toy)
    trans_toy_df = trans_toy_df.select(trans_toy_df.features, trans_toy_
df.target)

  # perform PCA to reduce the data dimention
    pca = PCA(k=100, inputCol="features", outputCol="pca_features")
    pca_toy_df = pca.fit(trans_toy_df).transform(trans_toy_df)

  # finalize the dataframe for regression
    final_toy_df = pca_toy_df.select(pca_toy_df.pca_features, trans_toy_
df.target)

    return final_toy_df
```

In [4]:
```
# function for logestic regression

from sklearn.metrics import log_loss

def log_regression(trainingDF, developeDF):

    # Train a LR model with training data
    lr = LogisticRegression(featuresCol = 'pca_features', labelCol = 'ta
rget', maxIter=10)
    lrModel = lr.fit(trainingDF)

    # Run model on testing data to get predictions
    dev_results = lrModel.transform(developeDF)

    # get the total number of predited instances
    correct_counts= dev_results.filter(dev_results.target == dev_results
.prediction).count()
    total_counts = dev_results.count()

    # calculate accuracy
    accuracy = correct_counts / total_counts

    # calculate log loss
    dev_results.select(['target', 'prediction'])
    y_true = dev_results.select('target').rdd.flatMap(lambda x: list(x))
.collect()
    y_pred = dev_results.select('probability').rdd.flatMap(lambda x: lis
t(x)).collect()
    logloss = log_loss(y_true, y_pred)

    return accuracy, logloss, lrModel
```

In [5]:
```
traindataRDD = sc.textFile("gs://w261-t10/train.txt")
```

In [6]:
```
trainData = create_dataframe(traindataRDD)
trainData.show(5)
```

```
+---+----+---+----+----+----+----+---+---+---+----+---+----+----+------
--+--------+--------+--------+--------+--------+--------+--------+-----
---+--------+--------+--------+--------+--------+--------+--------+----
----+--------+--------+--------+--------+--------+--------+--------+---
-----+--------+
|  0|   1|  2|   3|   4|   5|   6|  7|  8|  9|  10| 11|  12|  13|
14|      15|      16|      17|      18|      19|      20|      21|
22|      23|      24|      25|      26|      27|      28|      29|
30|      31|      32|      33|      34|      35|      36|      37|
38|      39|
+---+----+---+----+----+----+----+---+---+---+----+---+----+----+------
--+--------+--------+--------+--------+--------+--------+--------+-----
---+--------+--------+--------+--------+--------+--------+--------+----
----+--------+--------+--------+--------+--------+--------+--------+---
-----+--------+
|  0|   1|  1|   5|   0|1382|   4| 15|  2|181|   1|  2|null|   2|68fd1e
64|80e26c9b|fb936136|7b4723c4|25c83c98|7e0ccccf|de7995b8|1f89b562|a73ee
510|a8cd5504|b2cb9c98|37c9c164|2824a5f6|1adce6ef|8ba8b39a|891b62e7|e5ba
7672|f54016b9|21ddcdc9|b1252a9d|07b5194c|    null|3a171ecb|c5c50484|e8b
83407|9727dd16|
|  0|   2|  0|  44|   1| 102|   8|  2|  2|  4|   1|  1|null|   4|68fd1e
64|f0cf0024|6f67f7e5|41274cd7|25c83c98|fe6b92e5|922afcc0|0b153874|a73ee
510|2b53e5fb|4f1b46f3|623049e6|d7020589|b28479f6|e6c5b5cd|c92f3b61|07c5
40c4|b04e4670|21ddcdc9|5840adea|60f6221e|    null|3a171ecb|43f13e8b|e8b
83407|731c3655|
|  0|   2|  0|   1|  14| 767|  89|  4|  2|245|   1|  3|   3|  45|287e68
4f|0a519c5c|02cf9876|c18be181|25c83c98|7e0ccccf|c78204a1|0b153874|a73ee
510|3b08e48b|5f5e6091|8fe001f4|aa655a2f|07d13a8f|6dc710ed|36103458|8efe
de7f|3412118d|    null|    null|e587c466|ad3062eb|3a171ecb|3b183c5c|
null|    null|
|  0|null|893|null|null|4392|null|  0|  0|  0|null|  0|null|null|68fd1e
64|2c16a946|a9a87e68|2e17d6f6|25c83c98|fe6b92e5|2e8a689b|0b153874|a73ee
510|efea433b|e51ddf94|a30567ca|3516f6e6|07d13a8f|18231224|52b8680f|1e88
c74f|74ef3502|    null|    null|6b3a5ca6|    null|3a171ecb|9117a34a|
null|    null|
|  0|   3| -1|null|   0|   2|   0|  3|  0|  0|   1|  1|null|   0|8cf072
65|ae46a29d|c81688bb|f922efad|25c83c98|13718bbd|ad9fa255|0b153874|a73ee
510|5282c137|e5d8af57|66a76a26|f06c53ac|1adce6ef|8ff4b403|01adbab4|1e88
c74f|26b3c7a7|    null|    null|21c9516a|    null|32c7478e|b34f3128|
null|    null|
+---+----+---+----+----+----+----+---+---+---+----+---+----+----+------
--+--------+--------+--------+--------+--------+--------+--------+-----
---+--------+--------+--------+--------+--------+--------+--------+----
----+--------+--------+--------+--------+--------+--------+--------+---
-----+--------+
only showing top 5 rows
```

In [ ]:
```
transform_train = data_transform(trainData)
transform_train.show(5)
```

```
+--------------------+------+
|        pca_features|target|
+--------------------+------+
|[0.01766165750162...|   0.0|
|[0.01273638874372...|   0.0|
|[0.02164586699568...|   0.0|
|[0.01993249462198...|   0.0|
|[0.02801350200406...|   0.0|
+--------------------+------+
only showing top 5 rows
```

In [8]:
```
trainDF, devDF = transform_train.randomSplit([0.8,0.2], seed = 5)
```

In [ ]:
```
accuracy, logloss, lrmodel = log_regression(trainDF , devDF)
print('Prediction accuracy: %0.3f' %accuracy)
print('Log loss: %0.3f' %logloss)
```

```
Prediction accuracy: 0.746
Log loss: 0.535
```