```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter(action='ignore',category=FutureWarning)

y = pd.read_csv('/content/actual.csv')
X_train = pd.read_csv('/content/data_set_ALL_AML_train.csv')
X_test = pd.read_csv('/content/data_set_ALL_AML_independent.csv')

# Remove "call" columns from training and testing data
train_to_keep = [col for col in X_train.columns if "call" not in col]
test_to_keep = [col for col in X_test.columns if "call" not in col]

X_train_tr = X_train[train_to_keep]
X_test_tr = X_test[test_to_keep]

train_columns_titles = ['Gene Description', 'Gene Accession Number',
'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']

X_train_tr = X_train_tr.reindex(columns=train_columns_titles)

test_columns_titles = ['Gene Description', 'Gene Accession
Number','39', '40', '41', '42', '43', '44', '45', '46',
       '47', '48', '49', '50', '51', '52', '53',  '54', '55', '56',
'57', '58', '59',
       '60', '61', '62', '63', '64', '65', '66', '67', '68', '69',
'70', '71', '72']

X_test_tr = X_test_tr.reindex(columns=test_columns_titles)

X_train = X_train_tr.T
X_test = X_test_tr.T

X_train.head()

{"type":"dataframe","variable_name":"X_train"}

X_test.head()

{"type":"dataframe","variable_name":"X_test"}

# Clean up the column names for training and testing data
X_train.columns = X_train.iloc[1]
X_train = X_train.drop(["Gene Description", "Gene Accession
Number"]).apply(pd.to_numeric)
```

```
# Clean up the column names for Testing data
X_test.columns = X_test.iloc[1]
X_test = X_test.drop(["Gene Description", "Gene Accession
Number"]).apply(pd.to_numeric)

X_train.head()
```


```
{"type":"dataframe","variable_name":"X_train"}
```


```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

y["cancer"] = le.fit_transform(y["cancer"])

y
```


```
{"summary":"{\n  \"name\": \"y\",\n  \"rows\": 72,\n  \"fields\": [\n
{\n      \"column\": \"patient\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 20,\n        \"min\": 1,\n
\"max\": 72,\n        \"num_unique_values\": 72,\n        \"samples\":
[\n          5,\n          63,\n          19\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cancer\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"y"}
```


```
X_train["patient"] = X_train.index.values

X_test["patient"] = X_test.index.values

X_train = X_train.astype("int32")
X_test = X_test.astype("int32")

X_train
```


```
{"type":"dataframe","variable_name":"X_train"}
```


```
y
```


```
{"summary":"{\n  \"name\": \"y\",\n  \"rows\": 72,\n  \"fields\": [\n
{\n      \"column\": \"patient\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 20,\n        \"min\": 1,\n
\"max\": 72,\n        \"num_unique_values\": 72,\n        \"samples\":
[\n          5,\n          63,\n          19\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cancer\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
```

```
\"min\": 0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"y"}
```

```
X_train = pd.merge(X_train, y, on="patient")

X_test = pd.merge(X_test, y, on="patient")

X_train["cancer"].value_counts()
```

```
cancer
0    27
1    11
Name: count, dtype: int64
```

```
X_test["cancer"].value_counts()
```

```
cancer
0    20
1    14
Name: count, dtype: int64
```

```
import random

upsampled_data = random.sample(X_train.query("cancer == 1")
["patient"].index.to_list(), k=8, )

train_data_upsampled = pd.concat([X_train,
X_train.iloc[upsampled_data, :]])

X_train = train_data_upsampled.drop(columns=["patient", "cancer"])
y_train = train_data_upsampled["cancer"]
y_test = X_test["cancer"]
X_test = X_test.drop(columns=["patient", "cancer"])

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)

from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

parameters = {"bootstrap": [False, True],
              "n_estimators": [60, 70, 80, 90, 100],
              "max_features": [0.6, 0.65, 0.7, 0.75, 0.8],
              "min_samples_leaf": [8, 10, 12, 14],
              "min_samples_split": [3, 5, 7]
```

```
        }

model = RandomForestClassifier()

cv = cross_val_score(estimator=model,
 X=X_train_scaled,
 y=y_train,
 cv=3,
 scoring="accuracy")


rf_GS = GridSearchCV(model, parameters,cv=3, scoring="accuracy")
rf_GS.fit(X_train_scaled, y_train)
best_rf = rf_GS.best_estimator_
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-35-c4f7a56edccc> in <cell line: 23>()
     21 rf_GS = GridSearchCV(model, parameters,cv=3,
scoring="accuracy")
     22 rf_GS.fit(X_train_scaled, y_train)
---> 23 best_rf = rf_search.best_estimator_

NameError: name 'rf_search' is not defined
```

```
best_rf
```

```
RandomForestClassifier(bootstrap=False, max_features=0.6,
min_samples_leaf=14,
                       min_samples_split=7, n_estimators=70)
```

```
import sklearn.metrics as metrics
rf_prediction = best_rf.predict(X_test_scaled)
accuracy = metrics.accuracy_score(y_test, rf_prediction)
accuracy
```

```
0.9117647058823529
```

```
from sklearn.neighbors import KNeighborsClassifier

parameters = {
    "n_neighbors": [i for i in range(1, 30, 5)],
    "weights": ["uniform", "distance"],
    "algorithm": ["kd_tree"],
    "leaf_size": [1, 10, 20, 30],
    "p": [1, 2]
}

model = KNeighborsClassifier()
```

```
cv = cross_val_score(estimator=model,
 X=X_train_scaled,
 y=y_train,
 cv=3,
 scoring="accuracy")

knn_GS = GridSearchCV(KNeighborsClassifier(), parameters, n_jobs=-1,
verbose=1, scoring="accuracy")
knn_GS.fit(X_train_scaled, y_train)
best_knn = knn_GS.best_estimator_
```

```
Fitting 5 folds for each of 96 candidates, totalling 480 fits
```

```
best_knn
```

```
KNeighborsClassifier(algorithm='kd_tree', leaf_size=1, n_neighbors=6,
p=1,
                     weights='distance')
```

```
knn_prediction = best_knn.predict(X_test_scaled)
accuracy = metrics.accuracy_score(y_test, knn_prediction)
accuracy
```

```
0.7647058823529411
```

```
from sklearn.linear_model import LogisticRegression

parameters = {"C": [1e-03, 1e-2, 1e-1, 1, 10],
      "penalty": ["l1", "l2"]}

model = LogisticRegression()

cv = cross_val_score(estimator=model,
 X=X_train_scaled,
 y=y_train,
 cv=3,
 scoring="accuracy")

log_GS = GridSearchCV(model, parameters, cv=3, scoring="accuracy")
log_GS.fit(X_train_scaled, y_train)
best_lr = log_GS.best_estimator_
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:378: FitFailedWarning:
15 fits failed out of a total of 30.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.
```

```
Below are more details about the failures:
--------------------------------------------------------------------------------
----------
15 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan 0.95833333        nan 0.95833333        nan 0.95833333
        nan 0.95833333        nan 0.9375    ]
  warnings.warn(
```

```
best_lr = log_GS.best_estimator_
```

```
lr_prediction = best_lr.predict(X_test_scaled)
accuracy = metrics.accuracy_score(y_test, lr_prediction)
accuracy
```

```
0.8235294117647058
```

```
from sklearn.tree import DecisionTreeClassifier

parameters = {'max_leaf_nodes': list(range(2, 100)),
'min_samples_split': [2, 3, 4, 5, 6], 'max_depth':[3,4,5,6,7,8]}

model = DecisionTreeClassifier()

cv = cross_val_score(estimator=model,
 X=X_train_scaled,
 y=y_train,
 cv=3,
 scoring="accuracy")

dtc_GS = GridSearchCV(model, parameters, verbose=1, cv=3,
scoring="accuracy")
```

```
dtc_GS.fit(X_train_scaled, y_train)
best_dtc = dtc_GS.best_estimator_
```

```
Fitting 3 folds for each of 2940 candidates, totalling 8820 fits
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-61-80555a0ae8bb> in <cell line: 15>()
     13 dtc_GS = GridSearchCV(model, parameters, verbose=1, cv=3,
scoring="accuracy")
     14 dtc_GS.fit(X_train_scaled, y_train)
---> 15 best_dtc = dst_search.best_estimator_

NameError: name 'dst_search' is not defined
```

```
best_dtc
```

```
DecisionTreeClassifier(max_depth=3, max_leaf_nodes=3,
min_samples_split=4)
```

```
dtc_prediction = best_dtc.predict(X_test_scaled)
accuracy = metrics.accuracy_score(y_test, dtc_prediction)
accuracy
```

```
0.9117647058823529
```

```
rf_acc = metrics.accuracy_score(y_test,
best_rf.predict(X_test_scaled))
knn_acc = metrics.accuracy_score(y_test,
best_knn.predict(X_test_scaled))
lr_acc = metrics.accuracy_score(y_test,
best_lr.predict(X_test_scaled))
dtc_acc = metrics.accuracy_score(y_test,
best_dtc.predict(X_test_scaled))
```

```
model_acc = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression', 'Decision Tree',
              'Random Forest'],

    'accuracy score': [knn_acc, lr_acc, dtc_acc,
              rf_acc]})
```

```
model_acc.sort_values(by='accuracy score', ascending=False)
```


```
{"summary":"{\n  \"name\": \"model_acc\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          \"Random
Forest\",\n          \"KNN\",\n          \"Decision Tree\"\
n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"accuracy score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.07204381596421114,\n        \"min\":
0.7647058823529411,\n        \"max\": 0.9117647058823529,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.9117647058823529,\n          0.8235294117647058,\n
0.7647058823529411\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
mean_acc = np.mean([rf_acc, knn_acc, lr_acc, dtc_acc])
weight_rf = rf_acc / mean_acc
weight_knn = knn_acc / mean_acc
weight_lr = lr_acc / mean_acc
weight_dt = dtc_acc / mean_acc

model_weights = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression', 'Decision Tree',
              'Random Forest'],

    'weight': [weight_knn, weight_lr, weight_dt,
               weight_rf]})
model_weights.sort_values(by='weight', ascending=False)
```

```
{"summary":"{\n  \"name\": \"model_weights\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          \"Random
Forest\",\n          \"KNN\",\n          \"Decision Tree\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"weight\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.08446516354424746,\n        \"min\": 0.896551724137931,\n
\"max\": 1.0689655172413792,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.0689655172413792,\n
0.9655172413793104,\n          0.896551724137931\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

```
from sklearn.ensemble import VotingClassifier

ensemble = VotingClassifier(estimators=[("rf", best_rf), ("knn",
best_knn), ("lr", best_lr), ("dt", best_dtc)],
                                    voting="soft",
weights=[weight_rf, weight_knn, weight_lr, weight_dt])

ensemble.fit(X_train_scaled, y_train)
```

```
VotingClassifier(estimators=[('rf',
                              RandomForestClassifier(bootstrap=False,
                                                     max_features=0.6,
```

```
min_samples_leaf=14,

min_samples_split=7,

n_estimators=70)),
                               ('knn',

KNeighborsClassifier(algorithm='kd_tree',
                                                    leaf_size=1,
n_neighbors=6,
                                                    p=1,
weights='distance')),
                               ('lr', LogisticRegression(C=0.001)),
                               ('dt',
                                DecisionTreeClassifier(max_depth=3,
                                                       max_leaf_nodes=3,

min_samples_split=4))],
                  voting='soft',
                  weights=[1.0689655172413792, 0.896551724137931,
                           0.9655172413793104, 1.0689655172413792])

ens_prediction = ensemble.predict(X_test_scaled)
acc_score = metrics.accuracy_score(y_test, ens_prediction)

acc_score

0.9411764705882353
```