alpha-thalassemia-classifier-1.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

## Import libraries and dataset

```
import matplotlib.pyplot
import pandas as pd
import numpy as np
%matplotlib inline
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
alphanorm = pd.read_csv('alphanorm.csv', index_col = False)
```

```
alphanorm.head()
```

| | sex | hb | pcv | rbc | mcv | mch | mchc | rdw | wbc | neut | lymph | plt | hba | hba2 | hbf | phenotype |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | female | 10.8 | 35.2 | 5.12 | 68.7 | 21.2 | 30.8 | 13.4 | 9.6 | 53.0 | 33.0 | 309.0 | 88.5 | 2.6 | 0.11 | alpha carrier |
| 1 | male | 10.8 | 26.6 | 4.28 | 62.1 | 25.3 | 40.8 | 19.8 | 10.3 | 49.4 | 43.1 | 687.0 | 87.8 | 2.4 | 0.90 | alpha carrier |
| 2 | female | 10.8 | 35.2 | 5.12 | 68.7 | 21.2 | 30.8 | 13.4 | 9.6 | 53.0 | 33.0 | 309.0 | 88.5 | 2.6 | 0.10 | alpha carrier |
| 3 | male | 14.5 | 43.5 | 5.17 | 84.0 | 28.0 | 33.4 | 12.1 | 11.9 | 31.0 | 50.0 | 334.0 | 86.8 | 2.8 | 0.30 | alpha carrier |
| 4 | male | 11.5 | 34.4 | 5.02 | 68.7 | 22.9 | 33.4 | 15.7 | 20.4 | 67.0 | 30.0 | 596.0 | 86.3 | 2.4 | 1.30 | alpha carrier |

```
alphanorm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 203 entries, 0 to 202
Data columns (total 16 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   sex        203 non-null    object
 1   hb         203 non-null    float64
 2   pcv        203 non-null    float64
 3   rbc        202 non-null    float64
 4   mcv        203 non-null    float64
 5   mch        201 non-null    float64
 6   mchc       203 non-null    float64
 7   rdw        203 non-null    float64
 8   wbc        203 non-null    float64
 9   neut       203 non-null    float64
 10  lymph      203 non-null    float64
 11  plt        203 non-null    float64
 12  hba        203 non-null    float64
 13  hba2       203 non-null    float64
 14  hbf        203 non-null    float64
 15  phenotype  203 non-null    object
dtypes: float64(14), object(2)
memory usage: 25.5+ KB
```

## Handle categorical values

```
alphanorm['rbc'] = alphanorm['rbc'].fillna(alphanorm.groupby('phenotype')['rbc'].transform('mean'))
alphanorm['mch'] = alphanorm['mch'].fillna(alphanorm.groupby('phenotype')['mch'].transform('mean'))
```

```
for col in alphanorm.columns:
    if alphanorm[col].dtype != object:
        Q1 = alphanorm[col].quantile(0.25)
        Q3 = alphanorm[col].quantile(0.75)
        IQR = Q3 - Q1
        S = 1.5*IQR
        LB = Q1 - S
        UB = Q3 + S
        print(UB)
        alphanorm.loc[alphanorm[col] > UB,col] = UB
        alphanorm.loc[alphanorm[col] < LB,col] = LB
```

```
17.025000000000002
47.92500000000002
6.537500000000001
104.32500000000003
35.125
36.29074075
19.924999999999997
15.125000000000002
65.875
59.5
564.75
90.61428571249999
3.0000000000000004
1.4730769225
```

```
alphanorm = alphanorm.astype({'sex' : 'category', 'phenotype' : 'category'})
```

```
alphanorm['phenotype'] = alphanorm['phenotype'] == 'alpha carrier'
alphanorm['phenotype'] = alphanorm['phenotype'].replace({True:1, False:0})
```

```
<ipython-input-169-e8284ae63eeb>:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `
  alphanorm['phenotype'] = alphanorm['phenotype'].replace({True:1, False:0})
```

## Distinguish features & label

```
X = alphanorm.drop('phenotype', axis=1)
y = alphanorm['phenotype']
```

```
categorical_vars = list((X.select_dtypes(include=['category'])).columns)
categorical_vars
```

```
['sex']
```

```
X.head()
```

| | sex | hb | pcv | rbc | mcv | mch | mchc | rdw | wbc | neut | lymph | plt | hba | hba2 | hbf |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | female | 10.8 | 35.2 | 5.12 | 68.7 | 21.2 | 30.800000 | 13.4 | 9.600 | 53.000 | 33.0 | 309.00 | 88.5 | 2.6 | 0.11 |
| 1 | male | 10.8 | 26.6 | 4.28 | 62.1 | 25.3 | 36.290741 | 19.8 | 10.300 | 49.400 | 43.1 | 564.75 | 87.8 | 2.4 | 0.90 |
| 2 | female | 10.8 | 35.2 | 5.12 | 68.7 | 21.2 | 30.800000 | 13.4 | 9.600 | 53.000 | 33.0 | 309.00 | 88.5 | 2.6 | 0.10 |
| 3 | male | 14.5 | 43.5 | 5.17 | 84.0 | 28.0 | 33.400000 | 12.1 | 11.900 | 31.000 | 50.0 | 334.00 | 86.8 | 2.8 | 0.30 |
| 4 | male | 11.5 | 34.4 | 5.02 | 68.7 | 22.9 | 33.400000 | 15.7 | 15.125 | 65.875 | 30.0 | 564.75 | 86.3 | 2.4 | 1.30 |

## Perform encoding & Split the dataset

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
one_hot = OneHotEncoder()
transformer = ColumnTransformer([('one_hot', one_hot, categorical_vars)],
                                remainder= 'passthrough')
X = pd.DataFrame(transformer.fit_transform(X))
```

```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.20, random_state=42)
for train_index, test_index in split.split(alphanorm, alphanorm["phenotype"]):
    strat_train = alphanorm.loc[train_index]
    strat_test = alphanorm.loc[test_index]
```

```
train_X = strat_train.drop('phenotype', axis=1)
train_y = strat_train['phenotype']
test_x = strat_test.drop('phenotype', axis=1)
test_y = strat_test['phenotype']
```

```
one_hot1 = OneHotEncoder()
transformer = ColumnTransformer([('one_hot', one_hot1, categorical_vars)],
                                remainder= 'passthrough')
train_X = transformer.fit_transform(train_X)
train_X = pd.DataFrame(train_X)
```

```
one_hot2 = OneHotEncoder()
transformer = ColumnTransformer([('one_hot', one_hot2, categorical_vars)],
                                remainder= 'passthrough')
test_x = transformer.fit_transform(test_x)
test_x = pd.DataFrame(test_x)
```

```
train_X.head()
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 0.0 | 11.3 | 35.900000 | 4.71 | 76.0 | 24.0 | 31.5 | 14.0 | 8.90 | 65.875 | 29.0 | 268.0 | 86.523291 | 2.588608 | 0.769231 |
| 1 | 1.0 | 0.0 | 11.6 | 36.000000 | 4.69 | 77.0 | 25.0 | 32.2 | 13.3 | 7.70 | 51.000 | 59.5 | 249.0 | 86.600000 | 2.500000 | 0.100000 |
| 2 | 1.0 | 0.0 | 13.8 | 35.876404 | 5.81 | 74.0 | 23.7 | 31.6 | 11.8 | 9.90 | 53.000 | 37.0 | 312.0 | 87.100000 | 2.600000 | 0.100000 |
| 3 | 0.0 | 1.0 | 11.8 | 36.000000 | 5.27 | 68.3 | 22.4 | 32.5 | 14.8 | 12.48 | 42.000 | 51.0 | 311.0 | 86.600000 | 2.700000 | 0.500000 |
| 4 | 0.0 | 1.0 | 13.4 | 42.600000 | 5.13 | 83.0 | 26.2 | 31.6 | 12.0 | 7.10 | 43.000 | 50.0 | 235.0 | 84.864586 | 2.679310 | 0.537931 |

## Perform normalization & over sampling

```
from sklearn import preprocessing
train_X = preprocessing.normalize(train_X)
test_x = preprocessing.normalize(test_x)
```

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
train_X, train_y = sm.fit_resample(train_X, train_y)
```

```
train_X
```

```
array([[3.26934925e-03, 0.00000000e+00, 3.69436466e-02, ...,
        2.82874857e-01, 8.46306231e-03, 2.51488404e-03],
       [3.43141227e-03, 0.00000000e+00, 3.98043823e-02, ...,
        2.97160302e-01, 8.57853067e-03, 3.43141227e-04],
       [2.91233460e-03, 0.00000000e+00, 4.01902174e-02, ...,
        2.53664343e-01, 7.57206995e-03, 2.91233460e-04],
       ...,
       [1.15035313e-04, 3.08944499e-03, 3.59361935e-02, ...,
        2.77420552e-01, 8.29645257e-03, 2.38799967e-03],
       [0.00000000e+00, 3.22289376e-03, 4.35363515e-02, ...,
        2.75678880e-01, 7.98737908e-03, 1.18190454e-03],
       [2.86426670e-03, 3.27939116e-04, 4.44285301e-02, ...,
```

```
        2.71211984e-01, 8.46131327e-03, 1.63915968e-03]])
```

```
[ ] train_y.value_counts()
```

| phenotype | count |
|---|---|
| 0 | 118 |
| 1 | 118 |

dtype: int64

## Random Forest Classifier

```
[ ] from sklearn.ensemble import RandomForestClassifier
```

```
[ ] # rf = RandomForestClassifier(n_estimators=50, criterion='gini', max_features='sqrt')
    rf = RandomForestClassifier(n_estimators=100, criterion='entropy', max_features='log2', random_state=0)
```

```
[ ] rf.fit(train_X, train_y)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_features='log2', random_state=0)
```

```
[ ] y_pred = rf.predict(test_x)
```

```
[ ] y_pred
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
[ ] print("Accuracy:", accuracy_score(test_y, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(test_y, y_pred))
    print("Classification Report:\n", classification_report(test_y, y_pred))
```

```
Accuracy: 0.7317073170731707
Confusion Matrix:
 [[ 0 11]
 [ 0 30]]
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00        11
           1       0.73      1.00      0.85        30

    accuracy                           0.73        41
   macro avg       0.37      0.50      0.42        41
weighted avg       0.54      0.73      0.62        41

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sample
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sample
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sample
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

## Bagging Classifier

```
[ ] # We train a Bagging Classifier with Random Forest as the base model.
```

```
[ ] from sklearn.ensemble import BaggingClassifier
```

```
[ ] bagging = BaggingClassifier(estimator=RandomForestClassifier(criterion='gini', max_features='sqrt'), n_estimators=50, random_state=0)
```

```
[ ] bagging.fit(train_X, train_y)
```

```
BaggingClassifier
estimator:
RandomForestClassifier
RandomForestClassifier
```

```
[ ] y_pred = bagging.predict(test_x)
```

```
[ ] accuracy_score(test_y, y_pred)
```

```
0.6585365853658537
```

## AdaBoost Classifier

```
[ ] from sklearn.ensemble import AdaBoostClassifier
```

```
[ ] ada = AdaBoostClassifier(n_estimators=100, random_state=0, learning_rate=1.0)
```

```
[ ] ada.fit(train_X, train_y)
```

```
AdaBoostClassifier
AdaBoostClassifier(n_estimators=100, random_state=0)
```

```
[ ] y_pred = ada.predict(test_x)
```

```
[ ] accuracy_score(test_y, y_pred)
```

```
0.6585365853658537
```

## KNN Classifier

```
[ ] from sklearn.neighbors import KNeighborsClassifier
```

```
[ ] knn = KNeighborsClassifier(n_neighbors = 10, metric='minkowski', p=5)
```

```
[ ] knn.fit(train_X, train_y)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10, p=5)
```

```
print("accuracy score: ", knn.score(test_x, test_y))
print("confusion matrix: \n", confusion_matrix(test_y, knn.predict(test_x)))
print("classification report: ", classification_report(test_y, knn.predict(test_x)))
```

```
accuracy score:  0.4146341463414634
confusion matrix:
 [[ 7  4]
 [20 10]]
classification report:                precision    recall  f1-score   support

           0       0.26      0.64      0.37        11
           1       0.71      0.33      0.45        30

    accuracy                           0.41        41
   macro avg       0.49      0.48      0.41        41
weighted avg       0.59      0.41      0.43        41
```

## Use Cross Validation to find best hyperparameter for RF

```
[ ] from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import KFold
```

```
[ ] k = KFold(n_splits=10)
```

```
[ ] # Implement cross validation for diabetes data set
    for i, j in k.split(X):
        # Use .iloc to select rows based on indices
        X_train_cv, X_test_cv = X.iloc[i], X.iloc[j]
        y_train_cv, y_test_cv = y.iloc[i], y.iloc[j]
        rf = RandomForestClassifier(n_estimators=100, criterion='entropy', max_features='log2', random_state=0)
        rf.fit(X_train_cv, y_train_cv)
```

```
[ ] print("cross_val_score: ", cross_val_score(rf, train_X, train_y, cv = 10))
    print("\n")
    print("mean cross_val_score: ",np.mean(cross_val_score(rf, train_X, train_y, cv = 10)))
```

```
cross_val_score:  [0.75       0.83333333 0.58333333 0.66666667 0.875      0.83333333
 0.91304348 0.86956522 0.86956522 0.86956522]


mean cross_val_score:  0.806340579710145
```

## Define accuracies of models

```
[ ] rf_accuracy = cross_val_score(estimator=rf, X=train_X, y=train_y, cv=10)
    knn_accuracy = cross_val_score(estimator=knn, X=train_X, y=train_y, cv=10)
    ada_accuracy = cross_val_score(estimator=ada, X=train_X, y=train_y, cv=10)
    bagging_accuracy = cross_val_score(estimator=bagging, X=train_X, y=train_y, cv=10)
```

```
[ ] print("RF Accuracy: {:.2f}%".format(rf_accuracy.mean()*100))
    print("Ada Boost Accuracy: {:.2f}%".format(ada_accuracy.mean()*100))
    print("KNN Accuracy: {:.2f}%".format(knn_accuracy.mean()*100))
    print("Bagging Accuracy: {:.2f}%".format(bagging_accuracy.mean()*100))
```

```
RF Accuracy: 80.63%
Ada Boost Accuracy: 73.08%
KNN Accuracy: 57.16%
```

## Use GridSearchCV to find the best hyperparameters

```
[ ] from sklearn.model_selection import GridSearchCV
```

## For RF Model

```python
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_features': ['auto', 'sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}
```

```python
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=10, scoring='accuracy', n_jobs=-1)
grid_search.fit(train_X, train_y)
```

Show hidden output

```python
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

```
Best Hyperparameters: {'criterion': 'entropy', 'max_features': 'log2', 'n_estimators': 100}
Best Accuracy: 0.8320652173913043
```

## For KNN Model

```python
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['minkowski', 'euclidean']
}
```

```python
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=10, scoring='accuracy', n_jobs=-1)
grid_search.fit(train_X, train_y)
```

```
GridSearchCV
best_estimator_:
KNeighborsClassifier
KNeighborsClassifier
```

```python
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

```
Best Hyperparameters: {'metric': 'minkowski', 'n_neighbors': 3, 'weights': 'distance'}
Best Accuracy: 0.6735507246376812
```

## For Bagging Model

```python
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_samples': [0.5, 0.75, 1.0]
}
```

```python
grid_search = GridSearchCV(estimator=bagging, param_grid=param_grid, cv=10, scoring='accuracy', n_jobs=-1)
grid_search.fit(train_X, train_y)
```

```python
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

## For AdaBoost Model

```python
param_grid = {
    'n_estimators': [10, 50, 100],
    'learning_rate': [0.1, 0.5, 1.0]
}
```

```
Best Hyperparameters: {'learning_rate': 1.0, 'n_estimators': 100}
Best Accuracy: 0.7307971014492753
```

```python
grid_search = GridSearchCV(estimator=ada, param_grid=param_grid, cv=10, scoring='accuracy', n_jobs=-1)
grid_search.fit(train_X, train_y)
```

```python
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

## Create pickle

```python
import joblib
 # Save the model as a pickle in a file
joblib.dump(rf, 'thalassemia_model.pkl')
```

```
['thalassemia_model.pkl']
```

Colab paid products - Cancel contracts here

Connected to Python 3 Google Compute Engine backend