Model tuning

This notebook details the process of tuning the model.

From this point forward, I will be modeling with a few to predicting a binary outcome - ‘intervene’ or ‘not\_intervene’ and only use the ‘transformed’ data - that is, I will not be looking at the PCA dataset.

### Import libraries and datasets

import pandas as pd  
  
X\_train\_transformed = pd.read\_csv('../../data/X\_train\_transformed.csv')  
#X\_test\_transformed = pd.read\_csv('../../data/X\_test\_transformed.csv')  
X\_val\_transformed = pd.read\_csv('../../data/X\_val\_transformed.csv')  
#X\_train\_pca = pd.read\_csv('../../data/X\_train\_pca.csv')  
#X\_test\_pca = pd.read\_csv('../../data/X\_test\_pca.csv')  
#X\_val\_pca = pd.read\_csv('../../data/X\_val\_pca.csv')  
  
  
y\_train = pd.read\_csv('../../data/y\_train.csv')  
#y\_test = pd.read\_csv('../../data/y\_test.csv')  
y\_val = pd.read\_csv('../../data/y\_val.csv')

#### Refactoring the ‘final\_result’ target variable

# copy the data  
y\_train\_binary = y\_train.copy()  
  
# map values to 'intervene' and 'no\_intervene'  
y\_train\_binary.replace({'Pass': 'no\_intervene', 'Distinction': 'no\_intervene',  
 'Withdrawn': 'intervene', 'Fail': 'intervene'}, inplace=True)  
  
y\_val\_binary = y\_val.copy()  
y\_val\_binary.replace({'Pass': 'no\_intervene', 'Distinction': 'no\_intervene',  
 'Withdrawn': 'intervene', 'Fail': 'intervene'}, inplace=True)

### Hyperparameter tuning

Because I have a validation set, I have decided to use the validation set to tune the hyperparameters of the model.

In future analyses, I will use cross-validation to tune the hyperparameters to make the analysis more robust but for this initial modelling, I will use the validation set.

from sklearn.ensemble import VotingClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import cross\_val\_score, RepeatedStratifiedKFold, GridSearchCV, StratifiedKFold  
from sklearn.metrics import accuracy\_score  
from matplotlib import pyplot as plt  
  
  
# hyperparameter grid for each model  
lr\_param\_grid = {'C': [0.1, 1.0, 10.0, 100], 'penalty': ['l1', 'l2']}  
dt\_param\_grid = {'max\_depth': [None, 2, 5, 10], 'min\_samples\_split': [2, 5, 10], 'min\_samples\_leaf': [1, 2, 4]}  
rf\_param\_grid = {'n\_estimators': [10, 50, 100, 200, 300], 'max\_depth': [None, 2, 5, 10], 'min\_samples\_split': [2, 5, 10], 'min\_samples\_leaf': [1, 2, 4]}  
svc\_param\_grid = {'C': [0.1, 1.0, 10.0], 'kernel': ['linear', 'poly', 'rbf'], 'gamma': ['scale', 'auto']}  
knn\_param\_grid = {'n\_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance'], 'metric': ['euclidean', 'manhattan']}  
gnb\_param\_grid = {} # GaussianNB has no hyperparameters  
  
# models and their parameter grids  
models = [  
 ('LR', LogisticRegression(max\_iter=10000), lr\_param\_grid),  
 ('DT', DecisionTreeClassifier(), dt\_param\_grid),  
 ('RF', RandomForestClassifier(), rf\_param\_grid),  
 ('SVC', SVC(), svc\_param\_grid),  
 ('KNN', KNeighborsClassifier(), knn\_param\_grid),  
 ('GNB', GaussianNB(), gnb\_param\_grid)  
]  
  
# number of folds for k-fold cross-validation  
n\_folds = 5  
  
# hyperparameter tuning and cross-validation for each model  
accuracies = []  
model\_names = []  
best\_models = {}   
  
# hyperparameter tuning and cross-validation for each model  
for model\_name, model, param\_grid in models:  
 print(f"Model: {model\_name}")  
 grid\_search = GridSearchCV(model, param\_grid, cv=StratifiedKFold(n\_splits=n\_folds, shuffle=True), scoring='accuracy')  
 grid\_search.fit(X\_train\_transformed, y\_train\_binary.values.ravel())   
 print("Best Parameters:", grid\_search.best\_params\_)  
 print("Best Score:", grid\_search.best\_score\_)  
   
 # evaluate the model on the separate validation set  
 best\_model = grid\_search.best\_estimator\_  
 y\_pred = best\_model.predict(X\_val\_transformed)  
 accuracy = accuracy\_score(y\_val\_binary, y\_pred)  
 accuracies.append(accuracy)  
 model\_names.append(model\_name)  
 print("Validation Set Accuracy:", accuracy)  
  
 # save the best model from each classifier  
 best\_models[model\_name] = best\_model  
  
 print()

Model: LR

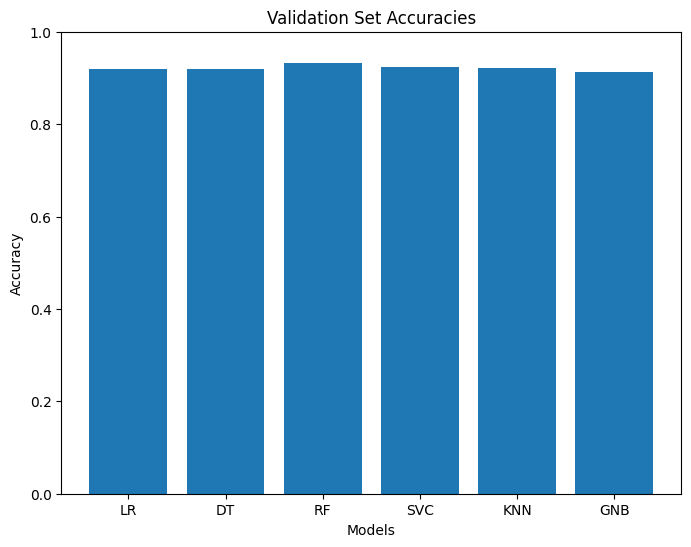
c:\Users\zoona\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\model\_selection\\_validation.py:378: FitFailedWarning:   
20 fits failed out of a total of 40.  
The score on these train-test partitions for these parameters will be set to nan.  
If these failures are not expected, you can try to debug them by setting error\_score='raise'.  
  
Below are more details about the failures:  
--------------------------------------------------------------------------------  
20 fits failed with the following error:  
Traceback (most recent call last):  
 File "c:\Users\zoona\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\model\_selection\\_validation.py", line 686, in \_fit\_and\_score  
 estimator.fit(X\_train, y\_train, \*\*fit\_params)  
 File "c:\Users\zoona\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\linear\_model\\_logistic.py", line 1162, in fit  
 solver = \_check\_solver(self.solver, self.penalty, self.dual)  
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
 File "c:\Users\zoona\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\linear\_model\\_logistic.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)  
c:\Users\zoona\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\model\_selection\\_search.py:952: UserWarning: One or more of the test scores are non-finite: [ nan 0.91925152 nan 0.91888038 nan 0.9189864  
 nan 0.91893339]  
 warnings.warn(

Best Parameters: {'C': 0.1, 'penalty': 'l2'}  
Best Score: 0.919251524548341  
Validation Set Accuracy: 0.9184160305343512  
  
Model: DT  
Best Parameters: {'max\_depth': 10, 'min\_samples\_leaf': 4, 'min\_samples\_split': 2}  
Best Score: 0.923545878667397  
Validation Set Accuracy: 0.9184160305343512  
  
Model: RF  
Best Parameters: {'max\_depth': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 10, 'n\_estimators': 50}  
Best Score: 0.933725648472332  
Validation Set Accuracy: 0.9320928753180662  
  
Model: SVC  
Best Parameters: {'C': 10.0, 'gamma': 'scale', 'kernel': 'rbf'}  
Best Score: 0.9245002514095942  
Validation Set Accuracy: 0.9243002544529262  
  
Model: KNN  
Best Parameters: {'metric': 'manhattan', 'n\_neighbors': 7, 'weights': 'distance'}  
Best Score: 0.923068959304811  
Validation Set Accuracy: 0.9212786259541985  
  
Model: GNB  
Best Parameters: {}  
Best Score: 0.9120407348186689  
Validation Set Accuracy: 0.9134860050890585

print(model\_names)  
print(accuracies)

['LR', 'DT', 'RF', 'SVC', 'KNN', 'GNB']  
[0.9184160305343512, 0.9184160305343512, 0.9320928753180662, 0.9243002544529262, 0.9212786259541985, 0.9134860050890585]

# plotting the accuracies  
plt.figure(figsize=(8, 6))  
plt.bar(model\_names, accuracies)  
plt.xlabel('Models')  
plt.ylabel('Accuracy')  
plt.title('Validation Set Accuracies')  
plt.ylim([0, 1])  
plt.show()



### Ensemble model from the best performing models

#### Voting classifier - Soft voting

# modify to probability True  
  
best\_models['SVC'].set\_params(probability=True)  
  
  
# ensemble model using the best models  
ensemble\_models = [('DT', best\_models['DT']),  
 ('LR', best\_models['LR']),  
 ('RF', best\_models['RF']),  
 ('SVC', best\_models['SVC']),  
 ('KNN', best\_models['KNN']),  
 ('GNB', best\_models['GNB'])]  
   
  
ensemble = VotingClassifier(estimators=ensemble\_models, voting='soft', weights=[1, 1, 3, 2, 2, 1])  
  
# fit the ensemble model   
ensemble.fit(X\_train\_transformed, y\_train\_binary.values.ravel())  
  
# evaluate the ensemble model on the validation set  
y\_pred\_ensemble = ensemble.predict(X\_val\_transformed)  
accuracy\_ensemble = accuracy\_score(y\_val\_binary, y\_pred\_ensemble)  
print("Ensemble Model Accuracy:", accuracy\_ensemble)

Ensemble Model Accuracy: 0.9257315521628499

#### Voting classifier - Hard voting

from sklearn.ensemble import VotingClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import cross\_val\_score, GridSearchCV  
from sklearn.metrics import accuracy\_score  
  
# ensemble model using the best models  
ensemble\_models = [('DT', best\_models['DT']),  
 ('LR', best\_models['LR']),  
 ('RF', best\_models['RF']),  
 ('SVC', best\_models['SVC']),  
 ('KNN', best\_models['KNN']),  
 ('GNB', best\_models['GNB'])]  
  
ensemble = VotingClassifier(estimators=ensemble\_models, voting='hard')  
  
# fit the ensemble model  
ensemble.fit(X\_train\_transformed, y\_train\_binary.values.ravel())  
  
# evaluate the ensemble model on the validation set  
y\_pred\_ensemble = ensemble.predict(X\_val\_transformed)  
accuracy\_ensemble = accuracy\_score(y\_val\_binary, y\_pred\_ensemble)  
print("Ensemble Model Accuracy:", accuracy\_ensemble)

Ensemble Model Accuracy: 0.9274809160305344

### Stacking Classifier

from sklearn.ensemble import StackingClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.naive\_bayes import GaussianNB  
  
# base models  
base\_models = [  
 ('DT', DecisionTreeClassifier()),  
 ('LR', LogisticRegression(max\_iter=10000)),  
 ('RF', RandomForestClassifier()),  
 ('SVC', SVC()),  
 ('KNN', KNeighborsClassifier()),  
 ('GNB', GaussianNB())  
]  
  
# meta-classifier  
meta\_model = LogisticRegression(max\_iter=10000)  
  
stacking\_classifier = StackingClassifier(estimators=base\_models, final\_estimator=meta\_model, cv=5)  
  
# fit the stacking classifier  
stacking\_classifier.fit(X\_train\_transformed, y\_train\_binary.values.ravel())  
  
# evaluate the stacking classifier on the validation set  
y\_pred\_stacking = stacking\_classifier.predict(X\_val\_transformed)  
accuracy\_stacking = accuracy\_score(y\_val\_binary, y\_pred\_stacking)  
print("Stacking Classifier Accuracy:", accuracy\_stacking)

Stacking Classifier Accuracy: 0.9317748091603053

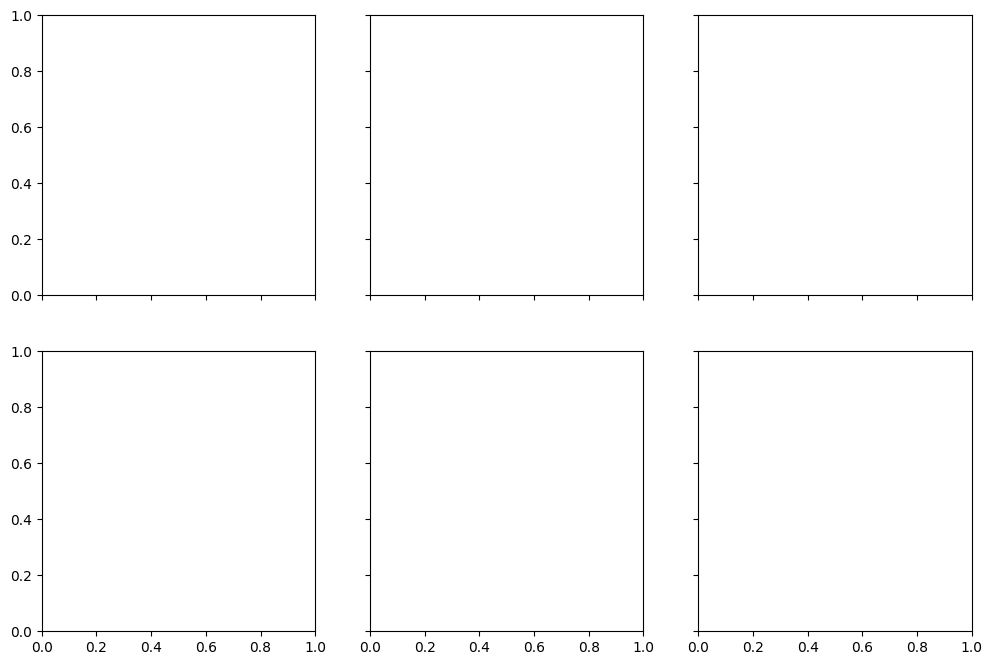
#### Decision boundary plotting

been fighting this for hours - another day, another time.

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.linear\_model import LogisticRegression  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.naive\_bayes import GaussianNB  
from itertools import product  
  
# best models  
best\_models = {  
 'LR': LogisticRegression(C=0.1, max\_iter=10000),  
 'DT': DecisionTreeClassifier(max\_depth=10, min\_samples\_leaf=4),  
 'RF': RandomForestClassifier(min\_samples\_split=10, n\_estimators=50),  
 'SVC': SVC(C=10.0, probability=True),  
 'KNN': KNeighborsClassifier(metric='manhattan', n\_neighbors=7, weights='distance'),  
 'GNB': GaussianNB()  
}  
  
# fit models with transformed data  
transformed\_data = []  
for features in product([0, 1], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]):  
 X\_train\_transformed\_subset = X\_train\_transformed.iloc[:, list(features)]  
 transformed\_data.append(X\_train\_transformed\_subset)  
  
for i, (model\_name, clf) in enumerate(best\_models.items()):  
 clf.fit(transformed\_data[i], y\_train\_binary.values.ravel())  
  
# decision regions  
x\_min, x\_max = transformed\_data[0].iloc[:, 0].min() - 1, transformed\_data[0].iloc[:, 0].max() + 1  
y\_min, y\_max = transformed\_data[0].iloc[:, 1].min() - 1, transformed\_data[0].iloc[:, 1].max() + 1  
xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.1), np.arange(y\_min, y\_max, 0.1))  
  
f, axarr = plt.subplots(2, 3, sharex='col', sharey='row', figsize=(12, 8))  
  
for idx, (model\_name, clf), tt in zip(range(6), best\_models.items(),  
 ['Decision Tree (depth=4)', 'KNN (k=7)',  
 'Kernel SVM', 'Logistic Regression', 'Random Forest', 'Gaussian NB']):  
 row\_idx, col\_idx = divmod(idx, 3) # convert the index to row and column  
  
 Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  
 Z = Z.reshape(xx.shape)  
  
 axarr[row\_idx, col\_idx].contourf(xx, yy, Z, alpha=0.4)  
 axarr[row\_idx, col\_idx].scatter(transformed\_data[idx].iloc[:, 0], transformed\_data[idx].iloc[:, 1],  
 c=y\_train\_binary, s=20, edgecolor='k')  
 axarr[row\_idx, col\_idx].set\_title(tt)  
  
plt.tight\_layout()  
plt.show()

c:\Users\zoona\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names  
 warnings.warn(

ValueError: could not convert string to float: 'intervene'



print(X\_train\_transformed.info())  
print(y\_train\_binary.shape)

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 18861 entries, 0 to 18860  
Data columns (total 11 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 subject\_SocSci 18861 non-null int64   
 1 subject\_Stem 18861 non-null int64   
 2 num\_of\_prev\_attempts 18861 non-null float64  
 3 studied\_credits 18861 non-null float64  
 4 prop\_submissions 18861 non-null float64  
 5 avg\_score 18861 non-null float64  
 6 submission\_distance 18861 non-null float64  
 7 stu\_activity\_count 18861 non-null float64  
 8 stu\_activity\_type\_count 18861 non-null float64  
 9 stu\_total\_clicks 18861 non-null float64  
 10 stu\_days\_active 18861 non-null float64  
dtypes: float64(9), int64(2)  
memory usage: 1.6 MB  
None  
(18861, 1)