kNN_stock_vs_tuned

March 25, 2021

Goal: Compare the base model's performance for the KNeighborsClassifier (kNN) to the tuned model's performance using a cleaned UCI-ML Repo dataset (Speech Recognition).

[337]:
```
# necessary imports for EDA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# sklearn imports
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score, train_test_split,␣
↪GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix
```

[338]:
```
# load data
srd = pd.read_csv('../Machine-Learning-Data/accent-mfcc-data-1.csv')

# view data sample
print('\nThe dataset is quite small. There are ', srd.shape[0], ' rows and ',␣
↪srd.shape[1], 'columns.\n')
display(srd.head())

srd.info()
```

```
The dataset is quite small. There are  329  rows and  13 columns.

  language         X1        X2        X3         X4        X5         X6  \
0       ES   7.071476 -6.512900  7.650800  11.150783 -7.657312  12.484021
1       ES  10.982967 -5.157445  3.952060  11.529381 -7.638047  12.136098
2       ES   7.827108 -5.477472  7.816257   9.187592 -7.172511  11.715299
3       ES   6.744083 -5.688920  6.546789   9.000183 -6.924963  11.710766
4       ES   5.836843 -5.326557  7.472265   8.847440 -6.773244  12.677218

          X7        X8        X9       X10       X11       X12
```

```
0 -11.709772  3.426596  1.462715 -2.812753  0.866538 -5.244274
1 -12.036247  3.491943  0.595441 -4.508811  2.332147 -6.221857
2 -13.847214  4.574075 -1.687559 -7.204041 -0.011847 -6.463144
3 -12.374388  6.169879 -0.544747 -6.019237  1.358559 -6.356441
4 -12.315061  4.416344  0.193500 -3.644812  2.151239 -6.816310
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 329 entries, 0 to 328
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   language  329 non-null    object
 1   X1        329 non-null    float64
 2   X2        329 non-null    float64
 3   X3        329 non-null    float64
 4   X4        329 non-null    float64
 5   X5        329 non-null    float64
 6   X6        329 non-null    float64
 7   X7        329 non-null    float64
 8   X8        329 non-null    float64
 9   X9        329 non-null    float64
 10  X10       329 non-null    float64
 11  X11       329 non-null    float64
 12  X12       329 non-null    float64
dtypes: float64(12), object(1)
memory usage: 33.5+ KB
```

Cleaning

[339]:
```
# convert the target variable to categorical
srd['language'] = srd['language'].astype('category')
```

EDA

[340]:
```
sns.pairplot(srd, hue = 'language')
plt.show()
```

Model Set-up

[341]:
```
# create train and test sets
X = srd.drop(['language'], axis = 'columns')
y = srd['language']

# random sampling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .25,␣
↪random_state = 28)

# stratified sampling
```

```
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X, y, test_size = .
↪25, random_state = 28, stratify = y)
```

[343]:
```
# check proportions of each category for both samples
prop_counts = pd.DataFrame()
prop_counts['y_train'] = pd.Series(y_train.value_counts(normalize = True))
prop_counts['y_test'] = pd.Series(y_test.value_counts(normalize = True))
prop_counts['y_train_s'] = pd.Series(y_train_s.value_counts(normalize = True))
prop_counts['y_test_s'] = pd.Series(y_test_s.value_counts(normalize = True))
display(prop_counts)
```

	y_train	y_test	y_train_s	y_test_s
US	0.487805	0.542169	0.500000	0.506024
UK	0.150407	0.096386	0.138211	0.132530
FR	0.101626	0.060241	0.093496	0.084337
IT	0.097561	0.072289	0.089431	0.096386
ES	0.093496	0.072289	0.089431	0.084337
GE	0.069106	0.156627	0.089431	0.096386

The data frame shows the stratified train/test split has similar proportions via category. This might actually hurt the model as there are fewer samples to further split during cross-validation. My assumption is that I will have to use 3-5 fold cross-validation to achieve higher accuracy for the stratified split, while I can use 5-10 fold cross-validation for the random split.

Stock Model

[344]:
```
# create the classifier and leave defaults values: p = 2 (Euclidean Distance),␣
↪weights = 'uniform', n_neighbors = 5
knn_base = KNeighborsClassifier()
knn_base_s = KNeighborsClassifier()

# fit the model to both samples
knn_base.fit(X_train, y_train)
knn_base_s.fit(X_train_s, y_train_s)
```

[344]:
```
KNeighborsClassifier()
```

Check the overall classification accuracy using the score method. It appears that the random sample produces slightly better accuracy than the stratified sample for the stock model.

[345]:
```
# random
print('The overall accuracy is: ', round(knn_base.score(X_test, y_test),2) *␣
↪100, '%')
# stratified
print('The overall accuracy is: ', round(knn_base_s.score(X_test_s,␣
↪y_test_s),2) * 100, '%')
```

```
The overall accuracy is:  47.0 %
```

```
The overall accuracy is:  47.0 %
```

[346]:
```
# make the predictions and check the score method manually
y_pred_base = knn_base.predict(X_test)
y_pred_base_s = knn_base_s.predict(X_test_s)

# create confusion matrices
cm_base = confusion_matrix(y_test, y_pred_base)
cm_base_s = confusion_matrix(y_test_s, y_pred_base_s)

# show results for manual calculation
print('Random Overall Accuracy: ', round((5+4+10+2+7+38)/ len(y_test), 2) *␣
 ↪100, '%')
print(cm_base,'\n')
print('Stratified Overall Accuracy: ', round((6+5+6+4+10+33)/ len(y_test_s), 2)␣
 ↪* 100, '%')
print(cm_base_s)
```

```
Random Overall Accuracy:  80.0 %
[[ 4  0  0  0  1  1]
 [ 0  1  0  0  2  2]
 [ 0  0  2  3  2  6]
 [ 0  0  0  3  1  2]
 [ 1  0  1  2  2  2]
 [ 0  9  3  3  3 27]]

Stratified Overall Accuracy:  77.0 %
[[ 6  0  1  0  0  0]
 [ 0  2  0  0  0  5]
 [ 1  0  2  0  0  5]
 [ 0  1  2  1  0  4]
 [ 2  0  0  0  6  3]
 [ 1  7  5  5  2 22]]
```

The stock model on random sampling generalized better to the unseen data. Check the average cross_val_score for folds: 3,5,10 for each split.

[347]:
```
# empty storage lists
random_cv_means = []
stratified_cv_means = []

# loop to get average values
for fold_num in [3,5,10]:
    print(fold_num, ' - cv:\n')
    res = cross_val_score(knn_base, X_train, y_train, cv = fold_num)
    res_s = cross_val_score(knn_base_s, X_train_s, y_train_s, cv = fold_num)
    print('random_scores: ', res,'\n')
    print('stratified_scores: ', res_s, '\n')
```

```
    random_cv_means.append(np.mean(res))
    stratified_cv_means.append(np.mean(res_s))

# results
avg_cvs = pd.DataFrame()
avg_cvs['Folds'] = pd.Series([3,5,10])
avg_cvs['Random Mean Score'] = pd.Series(random_cv_means)
avg_cvs['Stratified Mean Score'] = pd.Series(stratified_cv_means)
display(avg_cvs.set_index('Folds'))
```

```
3  - cv:

random_scores:  [0.57317073 0.51219512 0.52439024]

stratified_scores:  [0.51219512 0.57317073 0.6097561 ]

5  - cv:

random_scores:  [0.58       0.51020408 0.46938776 0.46938776 0.53061224]

stratified_scores:  [0.54       0.55102041 0.48979592 0.57142857 0.59183673]

10  - cv:

random_scores:  [0.52       0.52       0.48       0.56       0.44       0.6
 0.5        0.5        0.5        0.45833333]

stratified_scores:  [0.52       0.48       0.56       0.52       0.6        0.36
 0.5        0.625      0.45833333 0.70833333]
```

```
       Random Mean Score  Stratified Mean Score
Folds
3               0.536585               0.565041
5               0.511918               0.548816
10              0.507833               0.533167
```

Check Precision and Recall via generating a classification report.

[348]:
```
# view the classifcation reports
print("Random Sampling:\n")
classification_report(y_test, y_pred_base).split('\n')
```

```
Random Sampling:
```

[348]:
```
['              precision    recall  f1-score   support',
 '',
 '          ES       0.80      0.67      0.73         6',
 '          FR       0.10      0.20      0.13         5',
 '          GE       0.33      0.15      0.21        13',
 '          IT       0.27      0.50      0.35         6',
 '          UK       0.18      0.25      0.21         8',
 '          US       0.68      0.60      0.64        45',
 '',
 '    accuracy                           0.47        83',
 '   macro avg       0.39      0.40      0.38        83',
 'weighted avg       0.52      0.47      0.48        83',
 '']
```

[349]:
```
print("Stratified Sampling:\n")
classification_report(y_test_s, y_pred_base_s).split('\n')
```

```
Stratified Sampling:
```

[349]:
```
['              precision    recall  f1-score   support',
 '',
 '          ES       0.60      0.86      0.71         7',
 '          FR       0.20      0.29      0.24         7',
 '          GE       0.20      0.25      0.22         8',
 '          IT       0.17      0.12      0.14         8',
 '          UK       0.75      0.55      0.63        11',
 '          US       0.56      0.52      0.54        42',
 '',
 '    accuracy                           0.47        83',
 '   macro avg       0.41      0.43      0.41        83',
 'weighted avg       0.49      0.47      0.47        83',
 '']
```

0.0.1 KNeighborsClassifier() with tuned hyperparameters

[350]:
```
# set up the better format with the pipeline and steps
# scaling is not necessary as they are all MFCCs
steps = [('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)

# set up parameters and values to search over
parameters = {'knn__n_neighbors': range(1,20), # set range through p (features)
              'knn__p': range(1,3), # set range over manahttan and euclidean␣
 ↪distances
              'knn__weights': ['uniform', 'distance']} # use equal voting and␣
 ↪weighting voting via proximity
```

```
# instantiate the gridsearch cv object over the pipeline and parameters
# this searches for the best parameters for the model and uses those for scoring
knn_cv_object = GridSearchCV(pipeline, parameters, cv = 5) # use 5-fold cv as␣
↪this works best in tuned model
knn_cv_object_s = GridSearchCV(pipeline, parameters, cv = 5)

# fit the object - random sampling
knn_cv_object.fit(X_train, y_train)
```

[350]:
```
GridSearchCV(cv=5, estimator=Pipeline(steps=[('knn', KNeighborsClassifier())]),
             param_grid={'knn__n_neighbors': range(1, 20),
                         'knn__p': range(1, 3),
                         'knn__weights': ['uniform', 'distance']})
```

[351]:
```
# fit the object - stratified sampling
knn_cv_object_s.fit(X_train_s, y_train_s)
```

[351]:
```
GridSearchCV(cv=5, estimator=Pipeline(steps=[('knn', KNeighborsClassifier())]),
             param_grid={'knn__n_neighbors': range(1, 20),
                         'knn__p': range(1, 3),
                         'knn__weights': ['uniform', 'distance']})
```

Check the overall classification accuracy using the score method. It appears that the random sample produces much better accuracy than the stratified sample for the tuned model.

[352]:
```
# random
print('The overall accuracy is: ', round(knn_cv_object.score(X_test, y_test),2)␣
↪* 100, '%')
# stratified
print('The overall accuracy is: ', round(knn_cv_object_s.score(X_test_s,␣
↪y_test_s),2) * 100, '%')
```

```
The overall accuracy is:  61.0 %
The overall accuracy is:  53.0 %
```

The best parameters are:

[353]:
```
# print best params
print('Best Parameters - Random: ', knn_cv_object.best_params_, '\n')
print('Best Parameters - Stratified: ', knn_cv_object_s.best_params_, '\n')
```

```
Best Parameters - Random:  {'knn__n_neighbors': 19, 'knn__p': 1, 'knn__weights':
'uniform'}

Best Parameters - Stratified:  {'knn__n_neighbors': 14, 'knn__p': 1,
'knn__weights': 'uniform'}
```

Note: Setting the cv value for the grid search object above to 3,5,10 yields the following values for score accuracy:

Random: [81%, 83%, 83%]

Stratified: [81%, 81%, 73%]

This agrees with the original assumption I made that there wouldn't be enough observations in the training set for 10-fold CV to do well. This brings up an interesting point though. If the random state chosen pulled enough observations from the minority groups then the cross-validation would achieve good results, however it failed to pull enough, the generalizationto unseen data might score really low. Because of this, the stratified sample should be used and the number of folds should be chosen to maximize the accuracy for the stratified sampling.

[312]:
```
# make the predictions and check the score method manually
y_pred_tuned = knn_cv_object.predict(X_test)
y_pred_tuned_s = knn_cv_object_s.predict(X_test_s)

# create confusion matrices
cm_tuned = confusion_matrix(y_test, y_pred_tuned)
cm_tuned_s = confusion_matrix(y_test_s, y_pred_tuned_s)

# show results for manual calculation
print('Random Overall Accuracy: ', round((5+4+11+2+7+40)/ len(y_test), 2) *␣
↪100, '%')
print(cm_tuned,'\n')
print('Stratified Overall Accuracy: ', round((6+5+6+6+9+35)/ len(y_test_s), 2)␣
↪* 100, '%')
print(cm_tuned_s)
```

```
Random Overall Accuracy:  83.0 %
[[ 5  0  0  0  0  1]
 [ 0  4  0  0  0  1]
 [ 0  0 11  0  1  1]
 [ 0  1  1  2  0  2]
 [ 0  0  0  1  7  0]
 [ 0  1  2  0  2 40]]

Stratified Overall Accuracy:  81.0 %
[[ 6  0  0  0  0  1]
 [ 0  5  0  0  0  2]
 [ 0  0  6  1  0  1]
 [ 0  0  1  6  0  1]
 [ 0  0  0  0  9  2]
 [ 2  1  0  0  4 35]]
```

By tuning the hyperparameters of the model we achieved a 4% increase in accuracy for the proper stratified sampling.

[313]:
```python
print("Stratified Sampling:\n")
classification_report(y_test_s, y_pred_tuned_s).split('\n')
```

```
Stratified Sampling:
```

[313]:
```
['              precision    recall  f1-score   support',
 '',
 '          ES       0.75      0.86      0.80         7',
 '          FR       0.83      0.71      0.77         7',
 '          GE       0.86      0.75      0.80         8',
 '          IT       0.86      0.75      0.80         8',
 '          UK       0.69      0.82      0.75        11',
 '          US       0.83      0.83      0.83        42',
 '',
 '    accuracy                           0.81        83',
 '   macro avg       0.80      0.79      0.79        83',
 'weighted avg       0.81      0.81      0.81        83',
 '']
```

Saving the following tuned model to the working directory…

[354]:
```python
import joblib

joblib.dump(knn_cv_object_s, 'KNeighborsClassifier_tuned_model.sav')
```

[354]:
```
['KNeighborsClassifier_tuned_model.sav']
```

Load and view saved model…

[356]:
```python
best_model = joblib.load('KNeighborsClassifier_tuned_model.sav')
best_model
```

[356]:
```
GridSearchCV(cv=5, estimator=Pipeline(steps=[('knn', KNeighborsClassifier())]),
             param_grid={'knn__n_neighbors': range(1, 20),
                         'knn__p': range(1, 3),
                         'knn__weights': ['uniform', 'distance']})
```