COMP 4432 Machine Learning

Assignment 3

Assignment 3

- Opportunity to analyze an imperfect data set
  - Part 1F: Reverse engineer derived features
  - Part 1L: Stratified Imputation
  - Part 1O: Encoding categorical features
- Explore multiple classifiers
- New functionality
  - 2b) *cross_val_predict*
- Employ multiple evaluation metrics from predicted probabilities
- Identify the optimal hyperparameter settings for a Support Vector Machine

Assignment 3

- Part 1D: This dataset includes four redundant features, that is separate features that represent equivalent information (*alive* & *survived*, *class* & *pclass*, *embarked* & *embark_town*, and *adult_male* & *who*). Conduct and present **quantitative analysis** to verify these four pairings

```
df.groupby(['pclass'])['class'].value_counts()
```

```
pclass  class
1       First     216
        Second      0
        Third       0
2       Second    184
        First       0
        Third       0
3       Third     491
        First       0
        Second      0
Name: class, dtype: int64
```

```
df.groupby(['class'])['pclass'].unique()
```

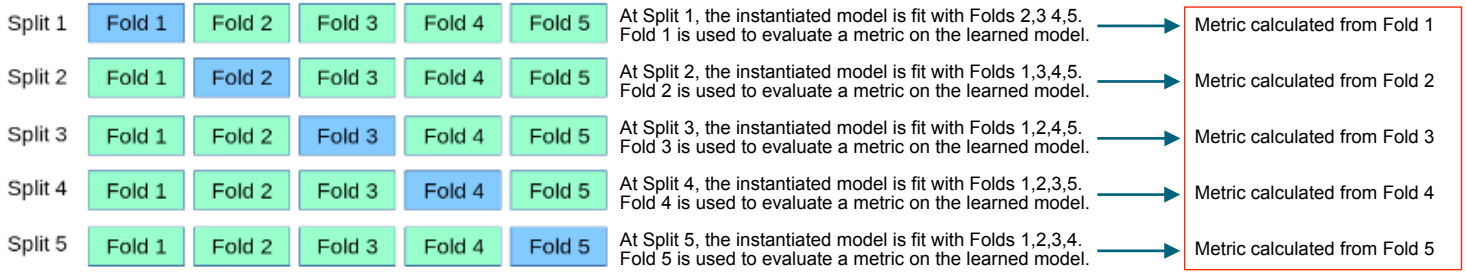
```
class
First      [1]
Second     [2]
Third      [3]
Name: pclass, dtype: object
```

```
df[['pclass', 'class']].drop_duplicates()
```

	pclass	class
0	3	Third
1	1	First
9	2	Second

cross_val_score

With a given instantiated model with fixed hyperparameters, the function *scikit.model_selection.cross_val_score* will return the evaluation scores calculated at each split (red outline).



Example usage of *cross_val_score.*
The instantiated model is a decision tree regressor with the default hyperparameters, *k*=5 folds, and the evaluation metric calculated is the negated mean squared error.

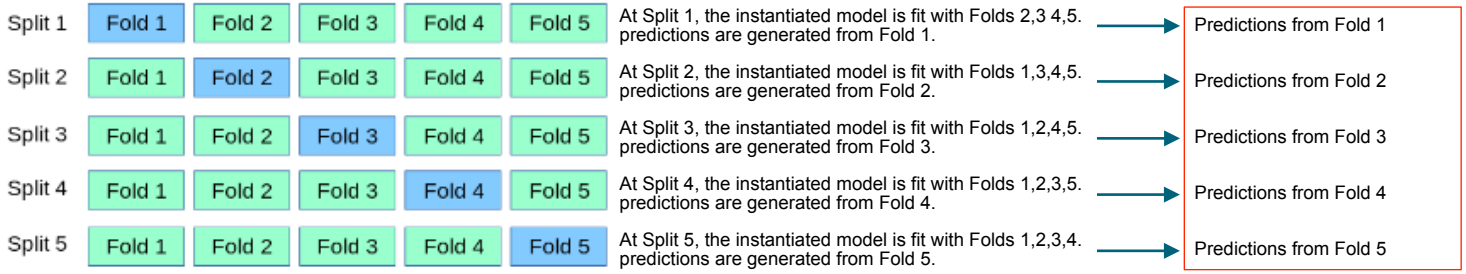
```
dec_tree_reg = DecisionTreeRegressor()

cross_val_score(estimator= dec_tree_reg, X= X_train, y= y_train, cv= 5, scoring= 'neg_mean_squared_error')
```

```
array([-0.57528967, -0.53376643, -0.55292476, -0.52752564, -0.49486407])
```

cross_val_predict

With a given instantiated model with fixed hyperparameters, the function *scikit.model_selection.cross_val_predict* will return the predictions calculated at each split



Example usage of *cross_val_predict.*
The model is a logistic regression classifier, *k*=5 folds, and we request the predicted probabilities to be returned.
Each instance belongs to only one validation (hold out) set. Predictions are made when the instance in is that set. The number of predictions in the *cross_val_predict* output will match the number of instances in X_train.

```
log_reg = LogisticRegression(max_iter= 1000)
cross_val_predict(estimator= log_reg, X= X_train, y= y_train, cv= 5, method= 'predict_proba')
```

```
array([[0.9556808 , 0.0443192 ],
       [0.77016454, 0.22983546],
       [0.69381657, 0.30618343],
       ...,
       [0.88743692, 0.11256308],
       [0.87440302, 0.12559698],
       [0.3900256 , 0.6099744 ]])
```

cross_val_predict

```
log_reg = LogisticRegression(max_iter= 1000)
cross_val_predict(estimator= log_reg, X= X_train, y= y_train, cv= 5, method= 'predict_proba')
```

Two points to discuss

max_iter = 1000

method = ‘predict_proba’

Logistic Regression Warning

```
log_reg = LogisticRegression(max_iter= 1000)
cross_val_predict(estimator= log_reg, X= X_train, y= y_train, cv= 5, method= 'predict_proba')
```

- *LogisticRegression()*
  - Default number of iterations for the solver is 100

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Probabilities versus Labels

```
log_reg = LogisticRegression(max_iter= 1000)
cross_val_predict(estimator= log_reg, X= X_train, y= y_train, cv= 5, method= 'predict_proba')
```

```
array([[0.9556808 , 0.0443192 ],
       [0.77016454, 0.22983546],
       [0.69381657, 0.30618343],
       ...,
       [0.88743692, 0.11256308],
       [0.87440302, 0.12559698],
       [0.3900256 , 0.6099744 ]])
```

Probability of instance being in class 1.

For confusion matrix based metrics (recall, accuracy, precision, etc.), need class label predictions, not probability predictions.
Python is helpful:

```
ValueError: Classification metrics can't handle a mix of binary and continuous targets
```

The ROC-AUC score function expects probabilities.
Python isn't helpful.
In the binary case, class labels are [0, 1], the bounds of probability.