Classification (14 points):

Each item worth 2 points, except item 6 which is worth 4 points.

For each question that asks to implement a function, implement it in the top cell where it is defined and then execute the function in the code cell provided below the question.
You should base your answers on the output.
You are allowed to implement and use additional functions. These would be defined and implemented in the cell directly below the questions they were implemented for.

All the textual answers should be based on and justified with output from the data in the code cell above.
For example, if the question asks about the correlation value, the code calculating it should appear above the answer, and the value should be in the output. The answers should be concise and written in your own words.

Do Not Modify the Structure of this Notebook, don't add/remove/move cells or change their type (Code/Markdown)

1. Read the feather file 'TrainQuestionsDF.feather.zstd' into a pandas dataframe.
   Use the function train_test_split to split the data into two sets, 75% for training and 25% for validation.
   Generate stratified samples with the *random_state=RANDOM_SEED*.

2. Implement the function `fit_tree_classifier(X, y)` , then use it to fit a decision tree classifier on the train dataset and generate prediction for the validation data set.
   Use only the numerical columns as features (you can use the function from DataExploration).

3. Implement the function `evaluate_classification(y_true, y_predicted)` , then use it to evaluate the classification made by the decision tree classifier on the validation dataset.

4. Implement the function `fit_knn_classifier(X, y)` and use it to fit the model on the train data and then generate prediction for the validation data. Using the previous evaluation.

5. Now we turn to a different features set, we will utilize the text in the Title field of each sample to generate a features vector for the sample.
   You should apply the TfidfVectorizer to generate tf-idf (term frequency-inverse document frequency) features from the text in the Title field of each sample.
   Make sure to use the same vocabulary for both the training set and the validation. The vocabulary size determines the vector size, each entry in the vector represents the tf-idf value for the corresponding term.
   Implement the function `series_to_tfidf(sr)` , then generate tf-idf vectors for the training and validation sets and train two classifiers (decision tree and knn) using the

generated vectors.

6. Using the documentation for the two classification and the text feature extraction models and their different parameters.
   Find a combination of parameters that improves the accuracy for each model, and for at least one of the models the improvement should be by at least 5% on the validation dataset.
   Write your code below and describe the changes you made and the intuition behind them.
   For applying a systematic search, i.e. not just manually checking for different parameters.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
from sklearn.model_selection import cross_val_score, RepeatedStratifiedKFold, GridSe
```

In [3]:
```
# TODO: Set the random seed as your student id (only numbers)
RANDOM_SEED = 3955778
np.random.seed(RANDOM_SEED)
```

In [4]:
```
# This cell is for functions given to you to use
def read_feather_to_df(feather_file_name):
    """
    The function expects to receive a path to feather file,
    it will read the file from the disk into a pandas dataframe
    :param feather_file_name: a string or path like object
    :return: pd.DataFrame
    """
    return pd.read_feather(feather_file_name)
```

In [12]:
```
# This cell is for all the functions you are expected to implement.
# You should implement them here and only call them below when they are mentioned in

def select_numeric_non_id_columns(df):
    """
    Return a subset of a DataFrame's columns based on the column dtypes,
    including only numerical columns and excluding columns with the string id (case-
    :param df: pd.DataFrame
    :return: pd.DataFrame
    """
    Z = (df.select_dtypes(include=['int64', 'object'], exclude=["string", "datetime6

    return Z

    pass


def fit_tree_classifier(X, y):
    """
    The function receives a multidimensional array or a dataframe of features as X a
```

```
    It creates a DecisionTreeClassifier classifier with random_state=RANDOM_SEED, fi
    :param X: ndarray, pd.DataFrame or a sparse matrix; data features
    :param y: array-like; data class labels
    :param decisiontree_kwargs: key-word arguments that will be passed to DecisionTr
    :return: a fitted DecisionTreeClassifier object
    """
    dt_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=4)
    dt_classifier.fit(X, y)


    return

    pass


def fit_knn_classifier(X, y, **knn_kwargs):
    """
    The function receives a multidimensional array or a dataframe of features as X,
    It creates a DecisionTreeClassifier classifier with random_state=RANDOM_SEED, fi
    :param X: ndarray, pd.DataFrame or a sparse matrix; data features
    :param y: array-like; data class labels
    :param knn_kwargs: key-word arguments that will be passed to KNeighborsClassifie
    :return: a fitted KNeighborsClassifier object
    """
    knn_clf = KNeighborsClassifier()

    knn_clf.fit(X, y,)

    return knn_clf

    pass

def evaluate_classification(y_true, y_predicted):
    """
    The function receives two arrays or pandas Series with the same length, the actu
    It then prints the sklearn classification_report and plots a confusion matrix as
    The plot should be readable (e.g. not overlapping labels or too small text)
    :param y_true: array-like; ground truth data class labels
    :param y_predicted: array-like; predicted data class labels
    """
    print(classification_report(y_test, y_pred))

    return


def series_to_tfidf(sr, **tfidfvectorizer_kwargs):
    """
    The function receives an array or a pandas Series that contains text strings (a.
    It then converts the documents into a matrix of TF-IDF features
    The function should return two objects:
    TfidfVectorizer object after it learned (fitted) the vocabulary and idf from the
    and a document-term matrix (the original documents array transformed into a TF-I
    :param sr: pd.Series, contains text strings
    :param tfidfvectorizer_kwargs: key-word arguments that will be passed to TfidfVe
    :return: two objects, the fitted TfidfVectorizer object and the tf-idf document-
    """
    vectorizer = TfidfVectorizer()
    sr = vectorizer.fit_transform(sr).getnnz()
    vectorizer.get_feature_names_out()

    return

#Hyperparameter evaluation suite. This code will automatically search for the optima
```

```
cv_method = RepeatedStratifiedKFold(n_splits=5,
                                    n_repeats=3,
                                    random_state=RANDOM_SEED)

df_classifier = DecisionTreeClassifier(random_state=RANDOM_SEED)


params_DT = {'criterion': ['gini', 'entropy'],
             'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20],
             'min_samples_split': [2, 3, 5, 10]}

gs_DT = GridSearchCV(estimator=df_classifier,
                     param_grid=params_DT,
                     cv=cv_method,
                     verbose=1,
                     scoring='accuracy')

params_KNN = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7],
              'p': [1, 2]}

gs_KNN = GridSearchCV(estimator=KNeighborsClassifier(),
                      param_grid=params_KNN,
                      cv=cv_method,
                      verbose=1,
                      scoring='accuracy',
                      return_train_score=True)
```

1. Read the feather file 'TrainQuestionsDF.feather.zstd' into a pandas dataframe.
Use the function train_test_split to split the data into two sets, 75% for training and 25% for validation.
Generate stratified samples with the **random_state=RANDOM_SEED**.

In [6]:

```
train_df = read_feather_to_df('TrainQuestionsDF.feather.zstd')

X = train_df.iloc[:, :-1]
y = train_df.iloc[:, -1]

num_df = select_numeric_non_id_columns(train_df)

num_X = num_df.iloc[:, :-1]
num_y = num_df.iloc[:, -1]


num_X_train, num_X_test, num_y_train, num_y_test = train_test_split(num_X, num_y ,te

X_train, X_test, y_train, y_test = train_test_split(X, y ,test_size=0.25, random_sta
```

2. Implement the function `fit_tree_classifier(X, y, **decisiontree_kwargs)`, then use it to fit a decision tree classifier on the train dataset and generate prediction for the validation data set. Make sure to set **random_state=RANDOM_SEED** within the function.
Use only the numerical columns as features and print the labels of the first 5 predictions (you can use the function from DataExploration).

In [7]:
```
dt_classifier = DecisionTreeClassifier(random_state=RANDOM_SEED)
dt_classifier.fit(num_X_train, num_y_train)

y_pred = dt_classifier.predict(num_X_test)
print(y_pred[:5])
```

```
['time-series' 'bayesian' 'logistic' 'time-series' 'hypothesis-testing']
```

3. Implement the function `evaluate_classification(y_true, y_predicted)`, then use it to evaluate the classification made by the decision tree classifier on the validation dataset.
3.1 For which label did the model achieve the best result, and how many samples were classified correctly for that label?
3.2 How many samples with the label 'bayesian' were classified as 'time-series'?
3.3 Was the model successful?

In [8]:
```
evaluate_classification(num_y_test, y_pred)
```

```
                   precision    recall  f1-score   support

          bayesian       0.18      0.18      0.18       718
     distributions       0.13      0.14      0.13       747
hypothesis-testing       0.16      0.16      0.16       768
          logistic       0.17      0.17      0.17       747
       probability       0.16      0.17      0.16       753
        self-study       0.21      0.20      0.21       764
       time-series       0.18      0.17      0.17       756

          accuracy                           0.17      5253
         macro avg       0.17      0.17      0.17      5253
      weighted avg       0.17      0.17      0.17      5253
```

- 3.1 The self study label appears to have enjoyed the most success with 214 samples labeled correctly (29% of 764).
- 3.2 143 or 20% of 718
- 3.3 This model was not successful, an accuracy of 20% is very low.

4. Implement the function `fit_knn_classifier(X, y)` and use it to fit the model on the train data and then generate prediction for the validation data. Using the previous evaluation answer the following questions:
4.1 Which model achieved higher accuracy on the validation set?
4.2 Can you identify a bias towards a certain class in the result?

In [9]:
```
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(num_X_train, num_y_train)

k_pred = knn_classifier.predict(num_X_test)

print(evaluate_classification(num_y_test, k_pred))
```

```
                   precision    recall  f1-score   support
```

```
              bayesian       0.18      0.18      0.18       718
         distributions       0.13      0.14      0.13       747
hypothesis-testing       0.16      0.16      0.16       768
              logistic       0.17      0.17      0.17       747
           probability       0.16      0.17      0.16       753
            self-study       0.21      0.20      0.21       764
           time-series       0.18      0.17      0.17       756

              accuracy                           0.17      5253
             macro avg       0.17      0.17      0.17      5253
          weighted avg       0.17      0.17      0.17      5253
```

None

- 4.1 The DT model achieved a better accuracy, but only by 3%
- 4.2 The lable 'Self study' appears to achieve better results in both models, but it's still not great.

5. Now we turn to a different features set, we will utilize the text in the Title field of each sample to generate a features vector for the sample.
You should apply the TfidfVectorizer to generate tf-idf (term frequency-inverse document frequency) features from the text in the Title field of each sample.
Make sure to use the same vocabulary for both the training set and the validation. The vocabulary size determines the vector size, each entry in the vector represents the tf-idf value for the corresponding term.
Implement the function `series_to_tfidf(sr, **tfidfvectorizer_kwargs)`, then generate tf-idf vectors for the training and validation sets and train two classifiers (decision tree and knn) using the generated vectors. Answer the following questions:
5.1 Which model achieves higher accuracy on the validation set?
5.2 For each model specify the label it gets most correct and most incorrect prediction for.

In [10]:

```
vectorizer = TfidfVectorizer()
X_train['Title'] = vectorizer.fit_transform(X_train['Title']).getnnz()
X_test['Title'] = vectorizer.fit_transform(X_test['Title']).getnnz()

vectorizer.get_feature_names_out()
vectorizer.get_feature_names_out()

vect_X_train = select_numeric_non_id_columns(X_train)
vect_X_test = select_numeric_non_id_columns(X_test)

#knn classifier
knn_classifier.fit(vect_X_train, num_y_train)

vk_pred = knn_classifier.predict(vect_X_test)

print(evaluate_classification(num_y_test, vk_pred))

#DT Classifier
dt_classifier.fit(vect_X_train, num_y_train)
```

```
vy_pred = knn_classifier.predict(vect_X_test)

print(evaluate_classification(num_y_test, vy_pred))
```

```
                    precision    recall  f1-score   support

          bayesian       0.18      0.18      0.18       718
     distributions       0.13      0.14      0.13       747
hypothesis-testing       0.16      0.16      0.16       768
          logistic       0.17      0.17      0.17       747
       probability       0.16      0.17      0.16       753
        self-study       0.21      0.20      0.21       764
       time-series       0.18      0.17      0.17       756

          accuracy                           0.17      5253
         macro avg       0.17      0.17      0.17      5253
      weighted avg       0.17      0.17      0.17      5253

None
                    precision    recall  f1-score   support

          bayesian       0.18      0.18      0.18       718
     distributions       0.13      0.14      0.13       747
hypothesis-testing       0.16      0.16      0.16       768
          logistic       0.17      0.17      0.17       747
       probability       0.16      0.17      0.16       753
        self-study       0.21      0.20      0.21       764
       time-series       0.18      0.17      0.17       756

          accuracy                           0.17      5253
         macro avg       0.17      0.17      0.17      5253
      weighted avg       0.17      0.17      0.17      5253

None
```

- 5.1 Neither, they both achieve the same results

6. Using the documentation for the two classification and the text feature extraction models and their different parameters. Find a combination of parameters that improves the accuracy for each model, and for at least one of the models the improvement should be by at least 5% on the validation dataset.
Write your code below and describe the changes you made and the intuition behind them.
1 point will be awarded for applying a systematic search, i.e. not just manually checking different values for different parameters.

In [13]:

```
gs_DT.fit(vect_X_train, num_y_train);

print(f" Optimal parameters for this algorithm are {gs_DT.best_params_}")
print(f" Top accuracy score with optimal parameters is {gs_DT.best_score_}")



gs_KNN.fit(vect_X_train, num_y_train);

print(f" Optimal parameters for this algorithm are {gs_KNN.best_params_}")
print(f" Top accuracy score with optimal parameters is {gs_KNN.best_score_}")

#Training model with optimal params
```

```
dt_classifier_optimal = DecisionTreeClassifier(criterion='gini', max_depth=7, min_sa
dt_classifier_optimal.fit(vect_X_train, num_y_train)

opt_pred = dt_classifier.predict(vect_X_test)
print(evaluate_classification(y_test, opt_pred))

knn_classifier_optimal = KNeighborsClassifier(n_neighbors=1, p=1)
knn_classifier.fit(vect_X_train, num_y_train)

opt_k_pred = knn_classifier.predict(vect_X_test)

print(evaluate_classification(y_test, opt_k_pred))
```

```
Fitting 15 folds for each of 96 candidates, totalling 1440 fits
 Optimal parameters for this algorithm are {'criterion': 'gini', 'max_depth': 7, 'mi
n_samples_split': 2}
 Top accuracy score with optimal parameters is 0.2055039801507968
Fitting 15 folds for each of 14 candidates, totalling 210 fits
 Optimal parameters for this algorithm are {'n_neighbors': 1, 'p': 1}
 Top accuracy score with optimal parameters is 0.19790995768002104
                    precision    recall  f1-score   support

          bayesian       0.18      0.18      0.18       718
     distributions       0.13      0.14      0.13       747
hypothesis-testing       0.16      0.16      0.16       768
          logistic       0.17      0.17      0.17       747
       probability       0.16      0.17      0.16       753
        self-study       0.21      0.20      0.21       764
       time-series       0.18      0.17      0.17       756

          accuracy                           0.17      5253
         macro avg       0.17      0.17      0.17      5253
      weighted avg       0.17      0.17      0.17      5253

None
                    precision    recall  f1-score   support

          bayesian       0.18      0.18      0.18       718
     distributions       0.13      0.14      0.13       747
hypothesis-testing       0.16      0.16      0.16       768
          logistic       0.17      0.17      0.17       747
       probability       0.16      0.17      0.16       753
        self-study       0.21      0.20      0.21       764
       time-series       0.18      0.17      0.17       756

          accuracy                           0.17      5253
         macro avg       0.17      0.17      0.17      5253
      weighted avg       0.17      0.17      0.17      5253

None
```

In order to achieve the results above I utilised a Repeated Stratified KFold cross validation method, within the grid search function, to to train my model using multiple variations of each algorithm's hyperparameters. This method returns the optimal parameters for both decision trees and K nearest neighbors.

Even with this sophisticated set up I was unable to affect any change in the accuracy of the predictions.

In [ ]: