KNN Model Selection

December 22, 2022

MAIN IMPORTS

[2]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

LOADING DATA

[3]:
```
# Loading the train.csv as the main dataset
data = pd.read_csv("../data/train.csv")

# Column Transformation to lowercase and underscored spaces
data.columns = data.columns.str.replace(' ', '_')
data.columns = data.columns.str.replace('-', '_')
data.columns = data.columns.str.lower()

X = data.loc[:, data.columns != 'lead']
y = data.loc[:, data.columns == 'lead']
```

SPLITTING DATA We split the dataset in to train: 75% and test: 25% using the default **train_test_split** function.

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=4045)
[X_train.shape, X_test.shape, y_train.shape, y_test.shape]
```

[4]:
```
[(779, 13), (260, 13), (779, 1), (260, 1)]
```

GET ALL FEATURE COMBINATIONS In this section, we create a function to produce sets of all possible feature combinations and save them in an array to be used in the model iteration. There will be at most $2^8 = 8192$ (including the empty set) feature combinations.

[5]:
```
# Function to produce an array of all feature combinations
def get_all_feature_combinations(data_columns):
    from itertools import chain, combinations
    "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)"
```

```
    feature_combinations = list(chain.from_iterable(combinations(data_columns,␣
 ↪r) for r in range(len(data_columns)+1)))

    feature_combinations_set = []
    for feature_combination in feature_combinations:
        feature_combination_set = []
        for feature in feature_combination:
            feature_combination_set.append(feature)

        feature_combinations_set.append(feature_combination_set)

    return feature_combinations_set

feature_combinations = get_all_feature_combinations(X.columns)
```

HYPERPARAMETER TUNING FUNCTION In this section, we create a function to find the best K value we could get by iterating thorugh given number of *k_iterations*. The input to this function will be training data **X** and **y** labels.

The function will then iterate through *k_iterations* which takes the data through a **GridSearchCV** pipeline which first scales the training data using **SandardScaler** and then fits a **KNeighborsClassifier** model to provide us the best K value along with it's accuracy.

Here, the **GridSearchCV** pipeline handles the cross validation search within itself.

[6]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# A function to produce an array with best K value along with it's accuracy -->␣
 ↪eg: returns [K = 10, 0.93]
def find_best_k_with_accuracy_cv(X, y, k_iterations, n_fold = 10):

    #knn = KNeighborsClassifier()
    k_range = list(range(1, k_iterations + 1))

    param_grid = {
        'knn__n_neighbors': k_range
    }

    pipe = Pipeline(
        [
            ('scaler', StandardScaler()),
            ('knn', KNeighborsClassifier(n_neighbors = k_range))
        ]
    )
```

```
    grid = GridSearchCV(pipe, param_grid, cv = n_fold, scoring = 'accuracy',␣
↪return_train_score = False, verbose=1)

    # fitting the model for grid search
    grid_search=grid.fit(X, y.to_numpy().reshape(-1, ))

    best_k = grid_search.best_params_.get('knn__n_neighbors')
    accuracy = grid_search.best_score_

    return [best_k, accuracy]
```

MODEL ITERATOR (PARAMETER TUNING) FUNCTION In this section, we have the code to produce a model performance report for each feature combination. This should ideally be run thorugh all feature combinations (i.e. $2^8 - 1 = 8191$ excluding the null set), and for each of the feature combination we run the above function **find_best_k_with_accuracy_cv** to give us the best K value along with it's accuracy. For computational convinience, we will be using all feature combinations which includes *at least 9* features for our model iteration.

This finally produces a report in csv format, which later can be used as an input for comparing how well each of the K-NN models would perform with an unseen test dataset.

[7]:
```
# Define number of iterations - max 8191
iterations = 1 # THIS VALUE NEEDS TO BE CHANGED TO MAXIMUM 8191 TO DO A FULL␣
 ↪ITERATION RUN

# Setting column names for iteration results
results_column_names = [
        'number_words_female',
        'total_words',
        'number_of_words_lead',
        'difference_in_words_lead_and_co_lead',
        'number_of_male_actors',
        'year',
        'number_of_female_actors',
        'number_words_male',
        'gross',
        'mean_age_male',
        'mean_age_female',
        'age_lead',
        'age_co_lead',
        'best_k',
        'accuracy',
        'iteration_no'
    ]

iteration_results = pd.DataFrame(columns=results_column_names)
```

```
for iteration in range(1, iterations + 1):
        if len(feature_combinations[iteration]) >= 9: # Any number within 0 to␣
 ↪13 - based on the minimum # of features we want to include
            best_k, accuracy = find_best_k_with_accuracy_cv(
                X_train[feature_combinations[iteration]], y_train, k_iterations␣
 ↪= 50, n_fold = 10
            )

            row = {
                'number_words_female': 0,
                'total_words': 0,
                'number_of_words_lead': 0,
                'difference_in_words_lead_and_co_lead': 0,
                'number_of_male_actors': 0,
                'year': 0,
                'number_of_female_actors': 0,
                'number_words_male': 0,
                'gross': 0,
                'mean_age_male': 0,
                'mean_age_female': 0,
                'age_lead': 0,
                'age_co_lead': 0,
                'best_k': best_k,
                'accuracy': accuracy,
                'iteration_no': iteration
            }

            for key, value in row.items():
                if key in feature_combinations[iteration]:
                    row[key] = 1
                else:
                    pass

            iteration_results = iteration_results.append(row, ignore_index=True)
            # This is the local path of the group member
            # iteration_results.to_csv(r'/Users/dininduseneviratne/Library/
 ↪CloudStorage/OneDrive-Uppsalauniversitet/Statistical Machine Learning/
 ↪project-results/results_8191.csv')

            # GIVE A LOCAL PATH IN THE FOLLOWING CODE BLOCK TO SAVE THE RESULTS
            iteration_results.to_csv(r'PATH.csv')
            print(str(iteration) + " OUT OF " + str(iterations) + " ITERATIONS␣
 ↪COMPLETED - " + str(iteration*100/iterations) + "%")

        else:
            pass
```