Machine learning Assignment

MATH2319 - Phase II report

Authors: Phil Steinke s3725547 Ash Olney s3686808

Methodology

The Google advertising dataset which we explored in detail in Phase I will be further transformed as appropriate for machine learning. This includes encoding the categorical variables, such as country_id and company_id, into binary variables by oneHotEncoding. The features and the target variable are then scaled between 0 and 1. As the original dataset is >200k rows, we have taken a random sample of 5000 rows and then split this sample further into training and testing data. The target variable, 'y', is a continuous numeric variable so we will be using regression algorithms to predict the outcome.

- neural network
- k-nearest neighbor regressior
- decision tree regressor

Each of these algorithms were optimised within a pipeline to fine tune the hyperparameters. This includes feature selection. As the encoded data had over 200 variables, we had to limit the range of features which could be selected during this process. The limited scope for hyperparameters to test was indended to manage our execution time. The best parameters as identified by the pipeline by the mean squared error are selected for the model and cross-validated.

Algorithm's tuning process

Feature selection is inlcuded in the pipeline process for algorithm fine tuning. Included is f-regression and mutual info regression. The pipeline iterates through 5, 10, 15, 20, and 100 variables. The entire set of variables is not included in the pipeline to reduce execution time. The pipeline algorithm for k-nearest neighbour includes the feature selection, and also the hyperparameters k, and p. k-neighbours runs from 1 through 10 and includes distance = 1, and 2. The pipeline algorithm for the decision tree regressor included feature selection, and hyperparameters max depth, and minimum sample split.

Algorithm Performance Analysis

Rank	Model	negative MSE	Execution time (min)
1	KNN	-0.0006579	359.1
2	DT	-0.0006866	278.8
3	NN	-0.0007560	618.6

As you can see above:

- the K-Nearest Neighbor has the best MSE (closest to 0)
- The execution time required to run the decision tree was most efficient

Setup

In [2]:
```
import io, joblib, math, numpy as np, os, pandas as pd, sklearn, warnings
from scipy import stats
warnings.filterwarnings("ignore")
# all of the sklearn libraries:
from sklearn import feature_selection as fs
from sklearn import preprocessing
from sklearn import preprocessing
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectKBest
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score, RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import Pipeline
import sklearn.metrics as metrics
from sklearn.tree import DecisionTreeRegressor
```

In [3]:
```
# from google.colab import files
# uploaded = files.upload()
```

In [4]:
```
# from google.colab import drive
# drive.mount('/content/drive')
# !ls "/content/drive/My Drive/" # this line will let you know if it's mounted
correctly
```

Import data in Google Colab

Import data in Jupyter / Atom / VS Code

In [6]:
```
# os.getcwd()
os.chdir('/Users/ashleigholney/Desktop/MATH2319-Machine-Learning/Course Project
') # ash's
__file__ = 'advertising_train.csv'
data = pd.read_csv(__file__)
```

consistent naming of columns (minus camelCase):

In [7]:
```
labelNames = [
    'case_id', 'company_id', 'country_id', 'device_type', 'day', 'dow',
    'price1', 'price2', 'price3', 'ad_area', 'ad_ratio', 'requests',
    'impression', 'cpc', 'ctr', 'viewability', 'ratio1', 'ratio2', 'ratio3',
    'ratio4', 'ratio5', 'y', ]
data.columns = labelNames
```

Common code/functions we reuse throughout code

In [140]:
```
categoricalFeatureList = [
    'company_id', 'country_id', 'device_type', 'day', 'dow'
    ]
```

source Pre-processing

Remove any index columns:

In [9]:
```
data = data.loc[:, data.nunique() != data.shape[0]]
data.shape
```

Out[9]:
```
(214128, 21)
```

Remove any constant features (that have only one unique value):

In [10]:
```
data = data.loc[:, data.nunique() != 1]
data.shape
```

Out[10]:
```
(214128, 21)
```

Check for nulls:

In [11]:
```
print(data.isnull().sum())
```

```
company_id     0
country_id     0
device_type    0
day            0
dow            0
price1         0
price2         0
price3         0
ad_area        0
ad_ratio       0
requests       0
impression     0
cpc            0
ctr            0
viewability    0
ratio1         0
ratio2         0
ratio3         0
ratio4         0
ratio5         0
y              0
dtype: int64
```

Encode Categorical Variables:

In [144]:
```
data['dow'] = data['dow'].str.lower()
data = pd.get_dummies(data, columns = categoricalFeatureList)
```

Our data is now 226 features wide

Outlier detection

Check if we have outliers

In [13]:
```
def get_outliers(df):
    absolute_normalized = np.abs(stats.zscore(df))
    return absolute_normalized > 3

def get_length_unique_outliers(df):
    # get boolean array of outliers:
    outliers = get_outliers(df)

    if (type(df).__module__ == np.__name__):
        # for numpy:
        # get boolean array of outliers:
        outliersIndexList = df[outliers]
    else:
        # for pandas:
        outliersIndexList = df[outliers].index.values.astype(int)

    # exclude records that include multiple outliers:
    uniqueOutliersIndexList = np.unique(outliersIndexList)
    uniqueOutliersLength = float(len(np.unique(outliersIndexList)))
    return uniqueOutliersLength

def get_proportion_unique_outliers(df):
    uniqueOutliersLength = get_length_unique_outliers(df)
    outlierPercent = round((uniqueOutliersLength/214128), 3)
    return outlierPercent
```

Split into target/data

Split `y` into separate dataset before normalisation

In [14]:
```
source = data.drop(columns='y')
target = data['y']
```

Normalise our source

In [15]:
```
sourceNormalised = source.copy()
sourceNormalised = preprocessing.MinMaxScaler().fit_transform(sourceNormalised)

targetNormalised = target.copy().to_frame(name=None)
targetNormalised = preprocessing.MinMaxScaler().fit_transform(targetNormalised)
```

Sample our data

Refactored samples to a set of 100 for testing pipelines, then in production set the sample = 5000

NOTE: This smaller sample size is limited by computing power. In a production environment, we would set this to a much larger sample

In [137]:
```
# sample = 100
# Setting random_state makes sampling reproducible
samples = 5000

# numpy version
sourceSample = pd.DataFrame(
    sourceNormalised,
    ).sample(n = samples, random_state = 8).values
targetSample = pd.DataFrame(
    targetNormalised,
    ).sample(n = samples, random_state = 8).values
```

Feature selection

- We refactored the parameter pipeline code (using DRY methodology) into the following dictionary:
- It is merged with the 4 pipeline dict's using `**` (requires python>3.5)

Parameters

- k - set to numerous options from 5 - 100
- Score function: tested the `f_regression` and `mutual_info_regression`

In [22]:
```
cv_method = RepeatedKFold(n_splits = 3, random_state = 999)
# parameter pipeline for `fselector` to use in 3 functions
params_pipe_fselector = {
    'fselector__score_func': [
        fs.f_regression,
        fs.mutual_info_regression
        ],
    'fselector__k': [5, 10, 15, 20, 100],
    # 'fselector__k': [5, 10, source.shape[1]],
}
```

Neural Network

Parameters

- hidden_layer_sizes : number of neurons in the ith hidden layer - 5 - 100
- activation: Activation function for the hidden layer (default 'relu')
- solver: The solver for weight optimization (default 'adam')
- learning_rate: Learning rate schedule for weight updates (default 'constant')

In [23]:
```
params_pipe_NN_fs = {
    **params_pipe_fselector, # pylint: disable=syntax-error,
    'nn__alpha': [0.001],
    'nn__hidden_layer_sizes': [(5,10,100,)],      # default: (100,)
    'nn__max_iter': [200],                    # default is 200
    'nn__activation': ['identity', 'logistic', 'tanh', 'relu'],# default 'relu'
    'nn__solver': ['lbfgs', 'sgd', 'adam',],   # default 'adam'
    'nn__verbose': [True],
    'nn__learning_rate': ['constant', 'invscaling', 'adaptive'],
}

steps_NN = [
    ('fselector', SelectKBest()),
    ('nn', MLPRegressor()),
    ]

pipe_NN_fs = Pipeline(steps_NN)

pipe_NN_fs = GridSearchCV(
    estimator = pipe_NN_fs,
    param_grid = params_pipe_NN_fs,
    cv = cv_method,
    n_jobs = -1,
    scoring = 'neg_mean_squared_error',
    refit = 'neg_mean_squared_error',
    verbose = 1,
)
```

In [24]:
```
pipe_NN_fs.fit(sourceSample, targetSample)
```

```
Fitting 30 folds for each of 360 candidates, totalling 10800 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    7.1s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   21.2s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   58.4s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 1234 tasks      | elapsed: 20.9min
[Parallel(n_jobs=-1)]: Done 1784 tasks      | elapsed: 82.7min
[Parallel(n_jobs=-1)]: Done 2434 tasks      | elapsed: 125.4min
[Parallel(n_jobs=-1)]: Done 3184 tasks      | elapsed: 127.1min
[Parallel(n_jobs=-1)]: Done 4034 tasks      | elapsed: 217.3min
[Parallel(n_jobs=-1)]: Done 4984 tasks      | elapsed: 250.2min
[Parallel(n_jobs=-1)]: Done 6034 tasks      | elapsed: 322.3min
[Parallel(n_jobs=-1)]: Done 7184 tasks      | elapsed: 372.8min
[Parallel(n_jobs=-1)]: Done 8434 tasks      | elapsed: 472.2min
[Parallel(n_jobs=-1)]: Done 9784 tasks      | elapsed: 504.7min
[Parallel(n_jobs=-1)]: Done 10800 out of 10800 | elapsed: 618.6min finished

Iteration 1, loss = 0.00270981
Iteration 2, loss = 0.00060439
Iteration 3, loss = 0.00048236
Iteration 4, loss = 0.00046257
Iteration 5, loss = 0.00045494
Iteration 6, loss = 0.00045074
Iteration 7, loss = 0.00044478
Iteration 8, loss = 0.00044358
Iteration 9, loss = 0.00043839
Iteration 10, loss = 0.00043618
Iteration 11, loss = 0.00043370
Iteration 12, loss = 0.00043098
Iteration 13, loss = 0.00042930
Iteration 14, loss = 0.00043095
Training loss did not improve more than tol=0.000100 for 10 consecutive epoch
s. Stopping.
```

Out[24]:
```
GridSearchCV(cv=<sklearn.model_selection._split.RepeatedKFold object at 0x1099
a7ef0>,
             error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('fselector',
                                        SelectKBest(k=10,
                                                    score_func=<function f_cla
ssif at 0x1234806a8>)),
                                       ('nn',
                                        MLPRegressor(activation='relu',
                                                     alpha=0.0001,
                                                     batch_size='auto',
                                                     beta_1=0.9, beta_2=0.999,
                                                     early_stopping=False,
                                                     epsilon=1...
                          'nn__activation': ['identity', 'logistic', 'tanh',
                                             'relu'],
                          'nn__alpha': [0.001],
                          'nn__hidden_layer_sizes': [(5, 10, 100)],
                          'nn__learning_rate': ['constant', 'invscaling',
                                                'adaptive'],
                          'nn__max_iter': [200],
                          'nn__solver': ['lbfgs', 'sgd', 'adam'],
                          'nn__verbose': [True]},
             pre_dispatch='2*n_jobs', refit='neg_mean_squared_error',
             return_train_score=False, scoring='neg_mean_squared_error',
             verbose=1)
```

In [25]:
```
joblib.dump(pipe_NN_fs.best_estimator_, 'pipe_NN_fs.pkl', compress=1)
# saved_nn = joblib.load('pipe_NN_fs.pkl')
```

Out[25]:
```
['pipe_NN_fs.pkl']
```

In [26]:
```
pipe_NN_fs.best_score_
```

Out[26]:
```
-0.0007555238411689335
```

In [27]:
```
pipe_NN_fs.best_params_
```

Out[27]:
```
{'fselector__k': 10,
 'fselector__score_func': <function sklearn.feature_selection.univariate_selec
tion.f_regression(X, y, center=True)>,
 'nn__activation': 'relu',
 'nn__alpha': 0.001,
 'nn__hidden_layer_sizes': (5, 10, 100),
 'nn__learning_rate': 'adaptive',
 'nn__max_iter': 200,
 'nn__solver': 'adam',
 'nn__verbose': True}
```

Nearest Neighbour Pipelines

In [30]:
```
steps_KNN = [
    ('fselector', SelectKBest()),
    ('knn', KNeighborsRegressor()),
    ]
pipe_KNN = Pipeline(steps_KNN)
```

the algorithms' tuning process,

`'fselector__k': [5, 10]` are feature selection `'knn__n_neighbors'` `'knn__p': [1, 2]` is distance

In [31]:
```
params_pipe_KNN = {
    **params_pipe_fselector,        # copies from above
    'knn__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'knn__p': [1, 2]}

pipe_KNN = GridSearchCV(
    estimator = pipe_KNN,
    param_grid = params_pipe_KNN,
    cv = cv_method,
    n_jobs = -1, # CHECKME: this was set to -2
    scoring = 'neg_mean_squared_error',
    verbose = 1)
```

In [32]:
```
pipe_KNN.fit(sourceSample, targetSample)
```

```
Fitting 30 folds for each of 200 candidates, totalling 6000 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  52 tasks      | elapsed:    1.0s
[Parallel(n_jobs=-1)]: Done 352 tasks      | elapsed:    4.9s
[Parallel(n_jobs=-1)]: Done 742 tasks      | elapsed: 16.7min
[Parallel(n_jobs=-1)]: Done 1092 tasks      | elapsed: 59.4min
[Parallel(n_jobs=-1)]: Done 1542 tasks      | elapsed: 72.2min
[Parallel(n_jobs=-1)]: Done 2092 tasks      | elapsed: 109.1min
[Parallel(n_jobs=-1)]: Done 2742 tasks      | elapsed: 146.5min
[Parallel(n_jobs=-1)]: Done 3492 tasks      | elapsed: 207.0min
[Parallel(n_jobs=-1)]: Done 4342 tasks      | elapsed: 235.5min
[Parallel(n_jobs=-1)]: Done 5292 tasks      | elapsed: 288.4min
[Parallel(n_jobs=-1)]: Done 6000 out of 6000 | elapsed: 359.1min finished
```

Out[32]:
```
GridSearchCV(cv=<sklearn.model_selection._split.RepeatedKFold object at 0x1099
a7ef0>,
             error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('fselector',
                                        SelectKBest(k=10,
                                                    score_func=<function f_cla
ssif at 0x1234806a8>)),
                                       ('knn',
                                        KNeighborsRegressor(algorithm='auto',
                                                            leaf_size=30,
                                                            metric='minkowski
',
                                                            metric_params=Non
e,
                                                            n_jobs=None,
                                                            n_neighbors=5, p=
2,
                                                            weights='uniform
'))],
                                verbose=False),
             iid='warn', n_jobs=-1,
             param_grid={'fselector__k': [5, 10, 15, 20, 100],
                         'fselector__score_func': [<function f_regression at 0
x123480840>,
                                                   <function mutual_info_regre
ssion at 0x123aec510>],
                         'knn__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'knn__p': [1, 2]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=1)
```

In [33]:
```
joblib.dump(pipe_KNN.best_estimator_, 'best_KNN.pkl', compress = 1)
# saved_knn = joblib.load('best_KNN.pkl')
```

Out[33]:
```
['best_KNN.pkl']
```

In [34]:
```
pipe_KNN.best_params_
```

Out[34]:
```
{'fselector__k': 10,
 'fselector__score_func': <function sklearn.feature_selection.univariate_selec
tion.f_regression(X, y, center=True)>,
 'knn__n_neighbors': 10,
 'knn__p': 1}
```

In [63]:
```
pipe_KNN.best_score_
```

Out[63]:
```
-0.000652080088372151
```

Decision Tree Pipelines

In [36]:
```
df_regressor = DecisionTreeRegressor(random_state=999)
```

In [46]:
```
params_pipe_DT_fs = {
    **params_pipe_fselector,        # copied from above
    'dt__criterion': ['mse'],
    'dt__max_depth': [1, 2, 3, 4],
    'dt__min_samples_split': [5, 50, 100, 150]
}

steps = [
    ('fselector', SelectKBest(score_func = fs.f_regression)),
    ('dt', df_regressor)
    ]

pipe_DT_fs = Pipeline(steps)

grid_DT_fs = GridSearchCV(
    pipe_DT_fs,
    params_pipe_DT_fs,
    cv = cv_method,
    n_jobs = -1,
    scoring = 'neg_mean_squared_error',
    refit = 'neg_mean_squared_error',
    verbose = 1
)
```

In [48]:
```
grid_DT_fs.fit(sourceSample, targetSample)
```

```
Fitting 30 folds for each of 160 candidates, totalling 4800 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   58.8s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 10.3min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed: 24.9min
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed: 45.7min
[Parallel(n_jobs=-1)]: Done 1234 tasks      | elapsed: 71.7min
[Parallel(n_jobs=-1)]: Done 1784 tasks      | elapsed: 104.2min
[Parallel(n_jobs=-1)]: Done 2434 tasks      | elapsed: 141.3min
[Parallel(n_jobs=-1)]: Done 3184 tasks      | elapsed: 185.5min
[Parallel(n_jobs=-1)]: Done 4034 tasks      | elapsed: 233.7min
[Parallel(n_jobs=-1)]: Done 4800 out of 4800 | elapsed: 278.8min finished
```

Out[48]:
```
GridSearchCV(cv=<sklearn.model_selection._split.RepeatedKFold object at 0x1099
a7ef0>,
             error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('fselector',
                                        SelectKBest(k=10,
                                                    score_func=<function f_reg
ression at 0x123480840>)),
                                       ('dt',
                                        DecisionTreeRegressor(criterion='mse',
                                                              max_depth=None,
                                                              max_features=Non
e,
                                                              max_leaf_nodes=N
one,
                                                              min_impurity_dec
rease=...
             param_grid={'dt__criterion': ['mse'],
                         'dt__max_depth': [1, 2, 3, 4],
                         'dt__min_samples_split': [5, 50, 100, 150],
                         'fselector__k': [5, 10, 15, 20, 100],
                         'fselector__score_func': [<function f_regression at 0
x123480840>,
                                                   <function mutual_info_regre
ssion at 0x123aec510>]},
             pre_dispatch='2*n_jobs', refit='neg_mean_squared_error',
             return_train_score=False, scoring='neg_mean_squared_error',
             verbose=1)
```

In [49]:
```
grid_DT_fs
```

Out[49]:
```
GridSearchCV(cv=<sklearn.model_selection._split.RepeatedKFold object at 0x1099
a7ef0>,
             error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('fselector',
                                        SelectKBest(k=10,
                                                    score_func=<function f_reg
ression at 0x123480840>)),
                                       ('dt',
                                        DecisionTreeRegressor(criterion='mse',
                                                              max_depth=None,
                                                              max_features=Non
e,
                                                              max_leaf_nodes=N
one,
                                                              min_impurity_dec
rease=...
             param_grid={'dt__criterion': ['mse'],
                         'dt__max_depth': [1, 2, 3, 4],
                         'dt__min_samples_split': [5, 50, 100, 150],
                         'fselector__k': [5, 10, 15, 20, 100],
                         'fselector__score_func': [<function f_regression at 0
x123480840>,
                                                   <function mutual_info_regre
ssion at 0x123aec510>]},
             pre_dispatch='2*n_jobs', refit='neg_mean_squared_error',
             return_train_score=False, scoring='neg_mean_squared_error',
             verbose=1)
```

In [50]:
```
joblib.dump(grid_DT_fs.best_estimator_, 'pipe_DT_fs.pkl', compress=1)
# saved_knn = joblib.load('pipe_DT_fs.pkl')
```

Out[50]:
```
['pipe_DT_fs.pkl']
```

In [52]:
```
grid_DT_fs.best_score_
```

Out[52]:
```
-0.0006999301498536244
```

In [53]:
```
grid_DT_fs.best_params_
```

Out[53]:
```
{'dt__criterion': 'mse',
 'dt__max_depth': 4,
 'dt__min_samples_split': 150,
 'fselector__k': 20,
 'fselector__score_func': <function sklearn.feature_selection.mutual_info_.mut
ual_info_regression(X, y, discrete_features='auto', n_neighbors=3, copy=True,
random_state=None)>}
```

NOTE: The best parameters are at the upper bounds of what was tested. the pipeline will be extended

In [54]:
```
params_pipe_DT2 = {
    'fselector__k': [26],
    'dt__criterion': ['mse'],
    'dt__max_depth': [5, 10, 15, 20],
    'dt__min_samples_split': [150, 200, 250, 300, 350, 400, 450, 500]}

steps = [
    ('fselector', SelectKBest(score_func = fs.f_regression)),
    ('dt', df_regressor)
]
pipe_DT2 = Pipeline(steps)

grid_DT2 = GridSearchCV(
    pipe_DT2,
    params_pipe_DT2,
    cv = cv_method,
    n_jobs = -1,
    scoring = 'neg_mean_squared_error',
    refit = 'neg_mean_squared_error',
    verbose = 1
)
```

In [55]:
```
grid_DT2.fit(sourceSample, targetSample)
```

```
Fitting 30 folds for each of 32 candidates, totalling 960 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.2s
[Parallel(n_jobs=-1)]: Done 319 tasks      | elapsed:    4.9s
[Parallel(n_jobs=-1)]: Done 819 tasks      | elapsed:   10.1s
[Parallel(n_jobs=-1)]: Done 945 out of 960 | elapsed:   11.5s remaining:    0.
2s
[Parallel(n_jobs=-1)]: Done 960 out of 960 | elapsed:   11.6s finished
```

Out[55]:
```
GridSearchCV(cv=<sklearn.model_selection._split.RepeatedKFold object at 0x1099
a7ef0>,
             error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('fselector',
                                        SelectKBest(k=10,
                                                    score_func=<function f_reg
ression at 0x123480840>)),
                                       ('dt',
                                        DecisionTreeRegressor(criterion='mse',
                                                              max_depth=None,
                                                              max_features=Non
e,
                                                              max_leaf_nodes=N
one,
                                                              min_impurity_dec
rease=...
                                                              presort=False,
                                                              random_state=99
9,
                                                              splitter='best
'))],
                                verbose=False),
             iid='warn', n_jobs=-1,
             param_grid={'dt__criterion': ['mse'],
                         'dt__max_depth': [5, 10, 15, 20],
                         'dt__min_samples_split': [150, 200, 250, 300, 350, 40
0,
                                                   450, 500],
                         'fselector__k': [26]},
             pre_dispatch='2*n_jobs', refit='neg_mean_squared_error',
             return_train_score=False, scoring='neg_mean_squared_error',
             verbose=1)
```

In [56]:
```
joblib.dump(grid_DT2.best_estimator_, 'grid_DT2.pkl', compress=1)
# saved_knn = joblib.load('grid_DT2.pkl')
```

Out[56]:
```
['grid_DT2.pkl']
```

In [57]:
```
grid_DT2.best_params_
```

Out[57]:
```
{'dt__criterion': 'mse',
 'dt__max_depth': 20,
 'dt__min_samples_split': 200,
 'fselector__k': 26}
```


```
{'dt__criterion': 'mse',
 'dt__max_depth': 20,
 'dt__min_samples_split': 200,
 'fselector__k': 26}
```


In [59]:
```
grid_DT2.best_score_
```

Out[59]:
```
-0.0007118538116185094
```

Performance Comparison of Algorithms

In [ ]:
```
# this is virul's code
cv_results_DT = cross_val_score(
    gs_pipe_DT_fs.best_estimator_,
    Data,
    target,
    cv=cv_method_ttest,
    scoring='accuracy')
cv_results_DT.mean().round(3)
```

In [118]:
```
cv_method_ttest = RepeatedKFold(n_splits = 10, random_state = 1)

def do_cross_val_score(estimator):
    """generates and compares the test to the real target (y)
    for the test data

    Parameters
    ----------
    estimator : type
        Description of parameter `estimator`.

    Returns
    -------
    type
        Mean Squared Error

    """
    cv_results = cross_val_score(
        estimator = estimator,
        X = sourceSample,
        y = targetSample,
        cv = cv_method_ttest,
        n_jobs = -2,
        scoring = 'neg_mean_squared_error')
    return(cv_results)
```

In [119]:
```
cv_results_KNN = do_cross_val_score(pipe_KNN.best_estimator_)
```

```
In [128]: cv_results_KNN.mean()
```

```
Out[128]: -0.0006579652267258794
```

```
In [132]: cv_results_NN = do_cross_val_score(pipe_NN_fs.best_estimator_)
```

```
In [127]: cv_results_NN.mean()
```

```
Out[127]: -0.0007560312310680465
```

```
In [131]: cv_results_DT = do_cross_val_score(grid_DT_fs.best_estimator_)
```

```
In [129]: cv_results_DT.mean()
```

```
Out[129]: -0.0007033477900923612
```

```
In [133]: cv_results_DT2 = do_cross_val_score(grid_DT2.best_estimator_)
```

```
In [130]: cv_results_DT2.mean()
```

```
Out[130]: -0.0006866844189859961
```

```
In [124]: print(stats.ttest_rel(cv_results_KNN, cv_results_NN))
          print(stats.ttest_rel(cv_results_KNN, cv_results_DT2))
          print(stats.ttest_rel(cv_results_DT2, cv_results_NN))
```

```
Ttest_relResult(statistic=15.181983879339706, pvalue=1.3470120147125353e-27)
Ttest_relResult(statistic=2.695891749078752, pvalue=0.008250336560359844)
Ttest_relResult(statistic=6.156974486867929, pvalue=1.591993301700683e-08)
```

```
In [125]: print(stats.ttest_rel(cv_results_KNN, cv_results_NN))
```

```
Ttest_relResult(statistic=15.181983879339706, pvalue=1.3470120147125353e-27)
```

Limitations of Report

Our methodology has several weaknesses and limitations:

- We did not know the context of the data, or it's target, as the company and what the target variable is, were not included in the Kaggle competition.
- We did not know what the target data was, as the company and what the target variable is, were not included in the Kaggle competition.
- We did not do in-depth feature selection and only included several options within the pipeline (i.e. 5 or 10 features).
- We used a small sample of the entire dataset in order to perform our analysis (5000 rows of a total possible ~200k).
- These limitations were put in place to reduce execution time; to improve the accuaracy of these models in future
- the hyperparameters of each algorithm could be further optimised
- As our data required us to perform regression our only performance metric was mean squared error.

Conclusion

This report examined an unknown companies Google Ads exported data. It compared three machine learning models to find the best parameters to maximise the ROI on advertising against an unknown parameter (y).

Ranked by MSE, the Nearest Neighbour (KNN) is the most performant model with the parameters: `n_neighbors: 10` and `p: 1` using the feature selection of `SelectKBest()`

The t-test p-value is significant ($p < 0.05$) when we compare it to both the Neural Network, and the Decision Tree model.