report.md

z5117408

Question 1

Part A

Decision Tree Results

Dataset	Default	0%	25%	50%	75%
australian	56.52% (2)	81.16% (7)	86.96% (2)	56.52% (2)	20.77% (7)
labor	66.67% (2)	94.44% (7)	44.44% (7)	66.67% (7)	50.00% (12)
diabetes	66.23% (2)	67.10% (7)	64.07% (12)	66.23% (2)	35.50% (27)
ionosphere	66.04% (2)	86.79% (7)	82.08% (27)	71.70% (7)	18.87% (12)

Part B

- answer: 4 - increase overfitting by increasing max depth of the decision tree
- rationale: the question states that max depth is being increased, so option 1 and 3 are invalid answers because they talk about max depth decreasing (and they are incorrect anyways). Overfitting increasing with depth is due to specificity on the data increasing as you go down levels in the tree and the amount of samples becoming smaller and smaller (incorporates too specific data and compounds too many assumptions for making general decisions).

Part C

- answer: 2 - yes, for 1/4 of the datasets
- rationale: the australian dataset had the same max depth and score in both situations, so no strict improvement. The labor dataset has a higher max depth for the 50% noise situation, but this did not offer any improvement. The diabetes dataset is the same situation as the australian dataset (no improvement). However, the ionosphere data set did see an improvement from 66.04% (default, max depth 2) to 71.70% (50% noise, max depth 7). Hence 1/4 of the datasets saw an improvement due to the grid search for max depth.

Question 2

Part A

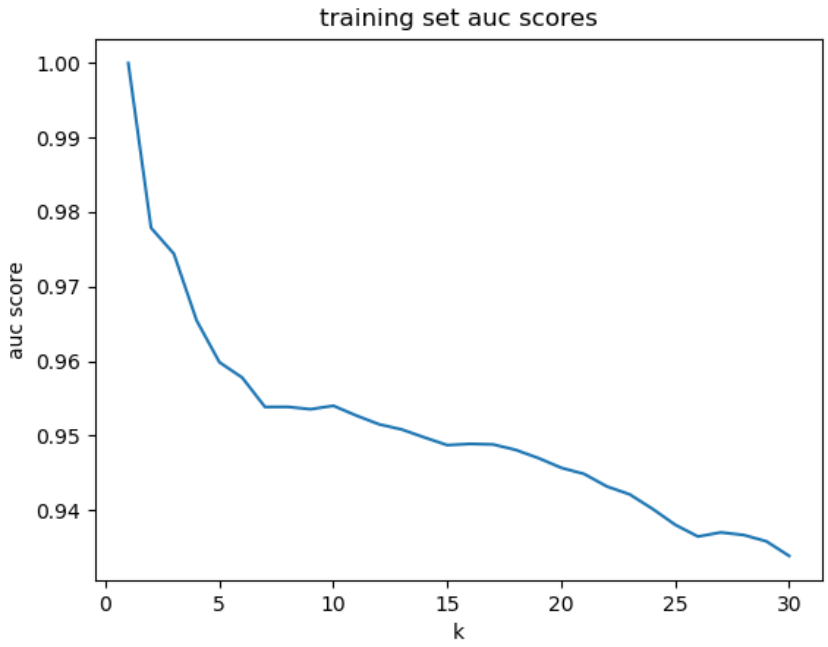
```
Accuracy score (k=2, train set):  0.8969404186795491
Accuracy score (k=2, test set):  0.7681159420289855
```

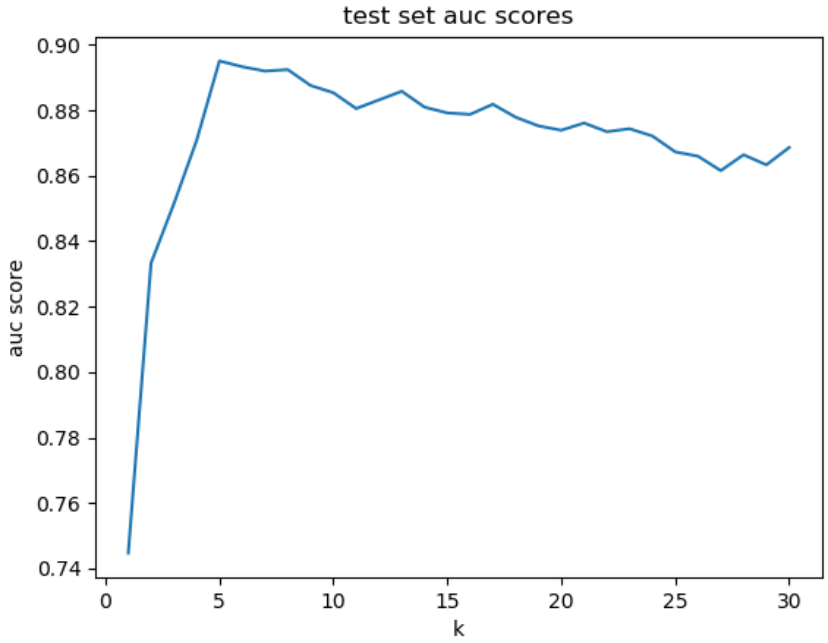
Part B

```
score for k=5 knn classifier was: 0.8950617283950617 (best AUC score)
```

- the best number of neighbours (k) is 5

Part C





Part D

```
precision score for k=5 on train set is:  0.8626760563380281
precision score for k=5 on test set is:  0.7666666666666667

precision score for k=2 on train set is:  1.0
precision score for k=2 on test set is:  0.7894736842105263

recall score for k=5 on train set is:  0.875
recall score for k=5 on test set is:  0.8518518518518519

recall score for k=2 on train set is:  0.7714285714285715
recall score for k=2 on test set is:  0.5555555555555556
```

- precision score comparison (TP/TP+FP):
  - it's not that suprising that the precision score for the train set is 1.0 when k=2. This is because we would be guaranteed for it to be 1.0 if k=1 (because the model was trained on the training data, and so when predicting samples from the training data it just finds that same sample), and so a value close or equal to 1.0 is expected for k=2 (given the data isn't super noisy). The sample

matching with itself, and then only 1 other neighbour (meaning that at least one of the k=2 classes has the correct guess) it why this has the highest precision overall.
    - k=2 performing worse on the test set than k=5 is because k is the optimal number of neighbours, meaning that k=2 has probably overfit whilst k=5 has fit as best as possible.
    - k=5 on the train set likely has a larger score than k=5 on test set because the model was trained on the train set, and so is better fitted for predicted on that set i.e. it's not suprising for the classifier to perform better on the training data than unseen data.
    - finally, k=5 on the train set likely has a larger score than k=2 on the test set because the k=2 is evaluating on unseen data and has an underfitting number of neighbours.
- recall score comparison (TP/TP+FN):
    - k=5 vs k=2: recall score is higher when k=5 (on both training and test sets) because the classifier is able to better fit (and thus avoid false positives due to noise) than the k=2 classifier (only 2 neighbours and so doesn't cope well with noise and thus overfits, leading to more false positives).
    - k=5 train vs k=5 test: slightly better recall on the train set just because this is what is was fitted to (the training data is unseen and faces the hard problem of generalising)
    - k=2 train vs k=2 test: the same reason as above, but here the problem is much more apparent because the number of neighbours is less.

CODE

```
import numpy as np
import sklearn as sk
import matplotlib.pyplot as plt
import sklearn.preprocessing
import sklearn.neighbors

# read in the complete data set, stripping the header
data_set = np.genfromtxt("./CreditCards.csv", delimiter=",", skip_header=1)

# tell numpy to not use scientific notation when printing small values
np.set_printoptions(suppress=True)
print(data_set) # observe the data_set to make sure it was read in correctly

print("\nsplitting attributes (x1, x2, ..., x14) and class value (Y)\n")
data_set, class_values = np.hsplit(data_set, [14]) # 14 columns of data, 1 column of predictions

# print data and class sets, as well as their RxC sizes
#print(data_set)
print("data set RxC size: ", np.size(data_set, 0), np.size(data_set, 1))
#print(class_values)
print("class values RxC size: ", np.size(class_values, 0), np.size(class_values, 1))

print("\nmin-max normalize the data\n")
min_max_normalizer = sk.preprocessing.MinMaxScaler(feature_range=(0,1), copy=False)
print(min_max_normalizer.fit_transform(data_set))

print("\ncheck min-max's\n")
print(data_set.min(axis=0))
print(data_set.max(axis=0))

print("\nsplit into training and data sets\n")
train_set, test_set = np.vsplit(data_set, [621])
train_classes = class_values[:621]
test_classes = class_values[621:]

# now print the lengths of these
print(len(train_classes))
print(len(test_classes))

print("train_set is;\n", train_set)
print("train set RxC size: ", np.size(train_set, 0), np.size(train_set, 1))

print("test set is;\n", test_set)
print("test set RxC size: ", np.size(test_set, 0), np.size(test_set, 1))

# Part A - k=2 knn classifier trained on training data, evaluated on both training and test data

# create the k=2 knn classifier
```

```
nn2 = sk.neighbors.KNeighborsClassifier(n_neighbors=2)
# fit it to the training set
# note: have to ravel the classes to make it into flat array instead of column array
print(nn2.fit(train_set, train_classes.ravel()))
# get predictions for both training and test sets
train_pred_nn2 = nn2.predict(train_set)
test_pred_nn2 = nn2.predict(test_set)

# print the AUC score for both training and test set predictions
# note: we don't get 1.0 (100%) for the train AUC score because we are using k=2
# if we used 1 it would perfectly fit each query to that data sample in the set and we get 100%
#print("AUC score (k=2, train set): ", sk.metrics.roc_auc_score(train_classes, train_pred_nn2))
#print("AUC score (k=2, test set): ", sk.metrics.roc_auc_score(test_classes, test_pred_nn2))

# EDIT, we actually wanted accuracy score for part A

print("Accuracy score (k=2, train set): ", sk.metrics.accuracy_score(train_classes, train_pred_nn2))
print("Accuracy score (k=2, test set): ", sk.metrics.accuracy_score(test_classes, test_pred_nn2))

# Part B - finding optical k using AOC score (train on train data, eval on test data only)

knn_scores = {}
for i in range(1, 31):
    nn_i = sk.neighbors.KNeighborsClassifier(n_neighbors=i)
    nn_i.fit(train_set, train_classes.ravel())
    nni_pred_score = nn_i.predict_proba(test_set)[:,1]
    knn_scores[i] = sk.metrics.roc_auc_score(test_classes, nni_pred_score)
    print("score for k=", i, " knn classifier was: ", knn_scores[i])

# find the best k (it is 5)
print("optimal k is: ", max(knn_scores, key=knn_scores.get))

# Part C - plotting auc score for all k, for both test and training sets, trained on training data

# generate the auc scores
knn_scores_train = {}
knn_scores_test = {}
for i in range(1, 31):
    nn_i = sk.neighbors.KNeighborsClassifier(n_neighbors=i)
    nn_i.fit(train_set, train_classes.ravel())
    nni_predscore_train = nn_i.predict_proba(train_set)[:,1]
    nni_predscore_test = nn_i.predict_proba(test_set)[:,1]
    knn_scores_train[i] = sk.metrics.roc_auc_score(train_classes, nni_predscore_train)
    knn_scores_test[i] = sk.metrics.roc_auc_score(test_classes, nni_predscore_test)
    print("train score for k=", i, " knn classifier was: ", knn_scores_train[i])
    print("test score for k=", i, " knn classifier was: ", knn_scores_test[i])

# unpacks the dictionary
tuples = sorted(knn_scores_train.items())
keys, values = zip(*tuples)

# plot it
plt.plot(keys, values)
plt.title('training set auc scores')
plt.xlabel('k');plt.ylabel('auc score')
plt.show()

# repeat for test set
tuples = sorted(knn_scores_test.items())
keys, values = zip(*tuples)

plt.plot(keys, values)
plt.title('test set auc scores')
plt.xlabel('k');plt.ylabel('auc score')
plt.show()

# Part D - compute precision metric and recall metric for k=5 (optimal k) and k=2 (part A)

nn5 = sk.neighbors.KNeighborsClassifier(n_neighbors=5)
nn5.fit(train_set, train_classes.ravel())
nn5_pred_train = nn5.predict(train_set)
nn5_pred_test = nn5.predict(test_set)
print("precision score for k=5 on train set is: ", sk.metrics.precision_score(train_classes, nn5_pred_train))
print("precision score for k=5 on test set is: ", sk.metrics.precision_score(test_classes, nn5_pred_test))
```

```
print("precision score for k=2 on train set is: ", sk.metrics.precision_score(train_classes, train_pred_nn2))
print("precision score for k=2 on test set is: ", sk.metrics.precision_score(test_classes, test_pred_nn2))

print("recall score for k=5 on train set is: ", sk.metrics.recall_score(train_classes, nn5_pred_train))
print("recall score for k=5 on test set is: ", sk.metrics.recall_score(test_classes, nn5_pred_test))

print("recall score for k=2 on train set is: ", sk.metrics.recall_score(train_classes, train_pred_nn2))
print("recall score for k=2 on test set is: ", sk.metrics.recall_score(test_classes, test_pred_nn2))
```