CPSC 330 - Applied Machine Learning

Homework 2: Decision trees and machine learning fundamentals

Associated lectures: Lectures 2 and 3

Due date: Tuesday, May 24, 2022 at 18:00

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

plt.rcParams["font.size"] = 16

from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
from sklearn.tree import DecisionTreeClassifier
```

Instructions

rubric={points:3}

Follow the homework submission instructions. In particular, **see the note about not pushing downloaded data to your repo**.

You are welcome to broadly discuss questions with your classmates but your final answers must be your own. **We are not allowing group submission for this homework assignment.**

Introducing the data set

For this assignment you'll be looking at Kaggle's Spotify Song Attributes dataset. The dataset contains a number of features of songs from 2017 and a binary variable `target` that represents whether the user liked the song (encoded as 1) or not (encoded as 0). See the documentation of all the features here.

This dataset is publicly available on Kaggle, and you will have to download it yourself. Follow the steps below to get the data CSV.

1. If you do not have an account with Kaggle, you will first need to create one (it's free).
2. Login to your account and download the dataset.
3. Unzip the data file if needed, then rename it to `spotify.csv`, and move it to the same directory as this notebook.

Exercise 1: Exploratory data analysis

1(a)

rubric={points:2}

Read in the data CSV and store it as a pandas dataframe named `spotify_df`. The first column of the .csv file should be set as the index.

In [2]:
```
spotify_df = pd.read_csv('spotify.csv', index_col=0)
```

1(b)

rubric={points:2}

Run the following line of code to split the data. How many training and test examples do we have?

> Note: we are setting the `random_state` so that everyone has the same split on their assignments. This will make it easier for the TAs to grade.

In [3]:
```
df_train, df_test = train_test_split(spotify_df, test_size=0.25, random_state=123)
df_train.shape, df_test.shape
```

Out[3]:
```
((1512, 16), (505, 16))
```

1512 training examples and 505 test examples

1(c)

rubric={points:3}

- Print out the output of `describe()` **on the training split**. This will compute some summary statistics of the numeric columns.
- Which feature has the smallest range?

> Hint: You can subtract the min value from the max value of the column to get the range.

Note that `describe` returns another DataFrame.

In [4]:
```
desc = df_train.describe()
desc
```

Out[4]:

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	lou
count	1512.000000	1512.000000	1512.000000	1512.000000	1512.000000	1512.000000	1512.000000	1512.0
mean	0.185337	0.617565	247606.240741	0.682022	0.139041	5.406746	0.189816	-7.0
std	0.257866	0.163086	81841.143743	0.211498	0.279536	3.635880	0.154115	3.8
min	0.000005	0.122000	16042.000000	0.014800	0.000000	0.000000	0.018800	-33.0
25%	0.009460	0.512000	200072.000000	0.564000	0.000000	2.000000	0.092200	-8.3
50%	0.063700	0.629500	230100.000000	0.713500	0.000076	6.000000	0.127000	-6.2
75%	0.253000	0.738250	274041.750000	0.847250	0.061100	9.000000	0.243000	-4.7
max	0.995000	0.984000	849960.000000	0.997000	0.976000	11.000000	0.969000	-0.3

In [5]:
```
ranges = desc.loc['max'] - desc.loc['min']
```

```
df_train.columns[np.argmin(ranges)]
```

Out[5]:
```
'speechiness'
```

speechiness

1(d)

rubric={points:5}

Let's focus on the following features:

- danceability
- tempo
- energy
- valence

For each of these features (in order), produce a histogram that shows the distribution of the feature values in the training set, **separated for positive and negative examples**. By "positive examples" we mean target = 1 (user liked the song, positive sentiment) and by "negative examples" we mean target = 0 (used disliked the song, negative sentiment). As an example, here is what the histogram would look like for a different feature, loudness:

(You don't have to match all the details exactly, such as colour, but your histograms should look something like this, with a reasonable number of bins to see the shape of the distribution.) As shown above, there are two different histograms, one for target = 0 and one for target = 1, and they are overlaid on top of each other. The histogram above shows that extremely quiet songs tend to be disliked (more blue bars than orange on the left) and very loud songs also tend to be disliked (more blue than orange on the far right).

To adhere to the DRY (Don't Repeat Yourself) principle, make sure you use a `for` loop for your plotting, rather than repeating the plotting code 4 times. For this to work, I used `plt.show()` at the end of your loop, which draws the figure and resets the canvas for your next plot.

Here is some code that separates out the dataset into positive and negative examples, to help you get started:

In [23]:
```
negative_examples = df_train.query("target == 0")
positive_examples = df_train.query("target == 1")
df_train[df_train['target'] == 1]
```

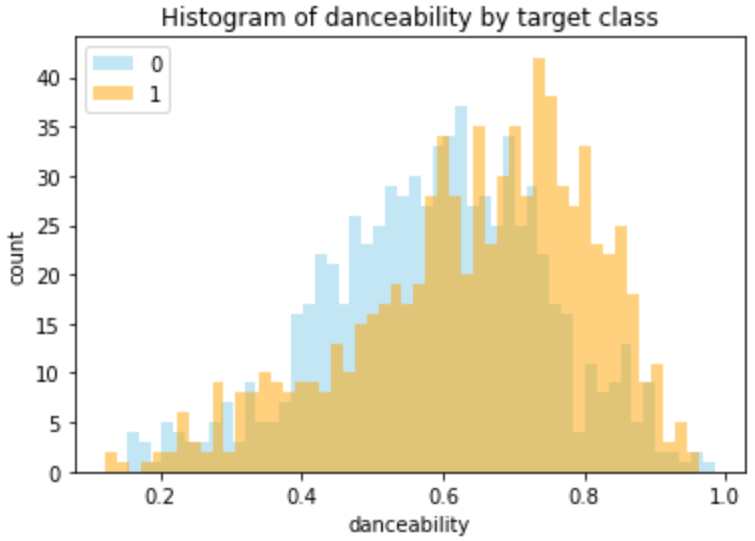
Out[23]:

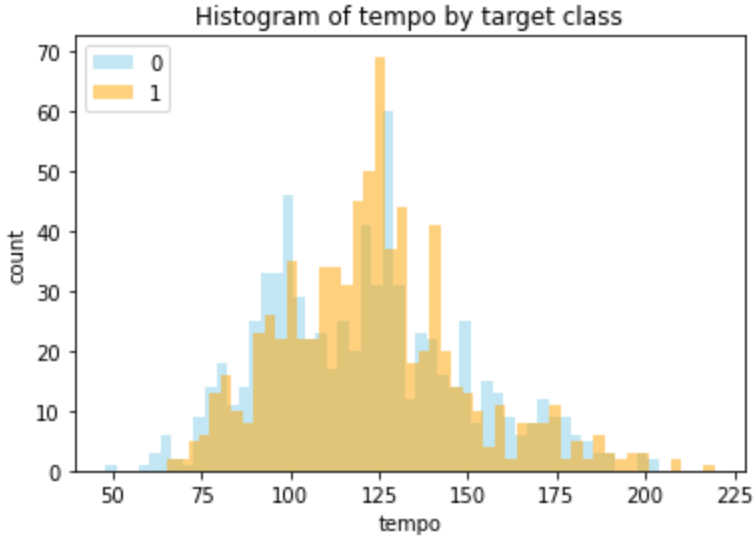
	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechine
641	0.030200	0.756	256933	0.595	0.000002	1	0.1460	-6.924	1	0.03
761	0.858000	0.741	227627	0.449	0.000522	5	0.1330	-6.621	1	0.05
561	0.032400	0.771	246573	0.416	0.000000	0	0.1110	-8.188	0	0.08
316	0.008380	0.587	235440	0.866	0.000036	9	0.1500	-6.444	1	0.08
715	0.017500	0.764	198907	0.547	0.000161	0	0.2300	-8.600	1	0.05
...	...	...	...	...	...	...	...	...	...	
47	0.092300	0.789	175240	0.494	0.000408	1	0.1440	-13.702	1	0.04
638	0.000423	0.723	358436	0.669	0.936000	11	0.0751	-5.026	0	0.05
113	0.009040	0.614	196352	0.907	0.000131	4	0.3600	-2.536	0	0.14
96	0.060400	0.799	333520	0.789	0.000000	0	0.0338	-8.208	1	0.03
106	0.378000	0.506	159253	0.813	0.029300	4	0.2760	-5.780	1	0.03

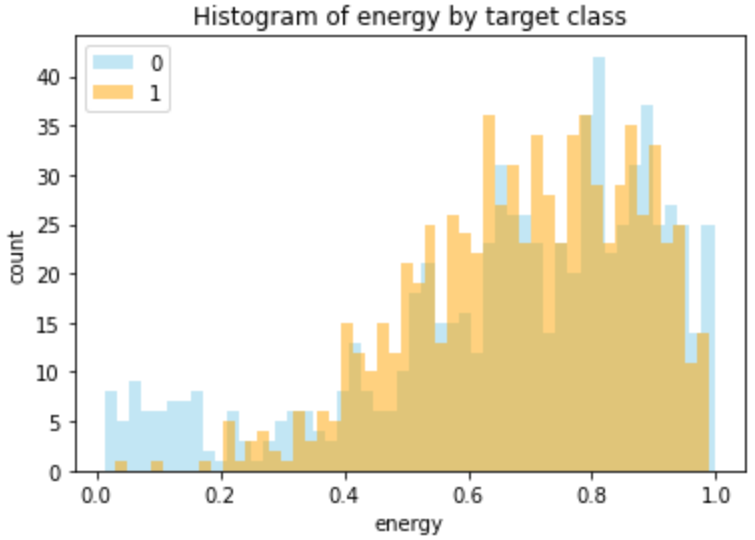
773 rows × 16 columns

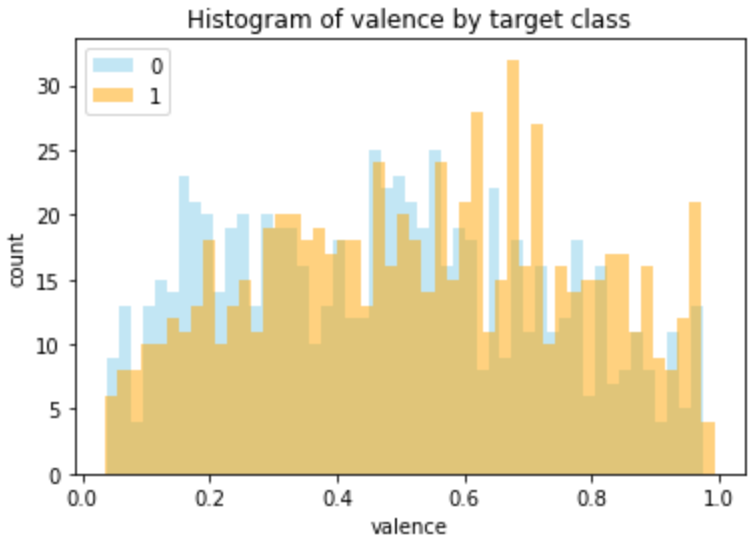
In [7]:

```
for feature in ['danceability', 'tempo', 'energy', 'valence']:
    plt.xlabel(feature)
    plt.ylabel('count')
    plt.title(f'Histogram of {feature} by target class')
    plt.hist(negative_examples[[feature]], bins=50, facecolor='skyblue', alpha=0.5, label="0")
    plt.hist(positive_examples[[feature]], bins=50, facecolor='orange', alpha=0.5, label="1")
    plt.legend(loc="upper left")
    plt.show()
```


Histogram of danceability by target class
0
1
count
danceability
0
5
10
15
20
25
30
35
40
0.2
0.4
0.6
0.8
1.0


Histogram of tempo by target class
0
1
count
tempo
0
10
20
30
40
50
60
70
50
75
100
125
150
175
200
225


Histogram of energy by target class
0
1
count
energy
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
1.0



1(e)

rubric={points:4}

Let's say you had to make a decision stump (decision tree with depth 1), *by hand*, to predict the target class. Just from looking at the plots above, describe a reasonable split (feature name and threshold) and what class you would predict in the two cases. For example, in the loudness histogram provided earlier on, it seems that very large values of loudness are generally disliked (more blue on the right side of the histogram), so you might answer something like this: "A reasonable split would be to predict 0 if loudness > -5 (and predict 1 otherwise)."

A reasonable split would be to predict 0 if energy < 0.2, and to predict 1 otherwise.

1(f)

rubric={points:2}

Let's say that, for a particular feature, the histograms of that feature are identical for the two target classes. Does that mean the feature is not useful for predicting the target class?

No. The feature may still be useful when used in conjunction with other features.

1(g)

rubric={points:2}

Note that the dataset includes two free text features labeled `song_title` and `artist`:

In [8]:
```
df_train[["song_title", "artist"]].head()
```

Out[8]:

	song_title	artist
1396	All in Vain	Mot
641	One of Those Nights	Juicy J
1765	Take Me As I Am	Wyclef Jean
1801	Save My Soul	JoJo
1624	Save the World - Radio Mix	Swedish House Mafia

- Do you think these features could be useful in predicting whether the user liked the song or not?
- Would there be any difficulty in using them in your model?

These features could be useful. People tend to have favorite artists, and like songs from their favorite artists. Title could also affect likeability of songs. But they cannot be used 'as is' since they are non-numerical. The cardinalities of these two features will likely get very high as song titles and artist names tend to be unique, and one-hot encoding might end up increasing the dimensionality too much.

Exercise 2: Using sklearn to build a decision tree classifier

2(a)

rubric={points:2}

- Create `X_train` and `y_train` and `X_test` and `y_test` from `df_train` and `df_test` above. Skip the `song_title` and `artist` features for now.
- Fit a `DecisionTreeClassifier` on the train set.

In [9]:
```
X_train = df_train.drop(['target', 'song_title', 'artist'], axis=1)
y_train = df_train[['target']]
X_test = df_test.drop(['target', 'song_title', 'artist'], axis=1)
y_test = df_test[['target']]

model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

Out[9]:
```
DecisionTreeClassifier()
```

2(b)

rubric={points:2}

Use the `predict` method to predict the class of the first example in your `X_train`. Is the prediction correct? That is, does it match with the corresponding class in `y_train`?

> Hint: you can grab the first example with `X_train.iloc[[0]]`.

In [10]:
```
model.predict(X_train.iloc[[0]])[0]
```

Out[10]:
```
0
```

In [11]:
```
y_train.iloc[0]['target']
```

Out[11]:
```
0
```

The prediction is correct.

2(c)

rubric={points:2}

Use the `cross_val_score` function on your training set to compute the 10-fold cross-validation accuracy of your tree.

In [12]:
```
cs = cross_val_score(model, X_train, y_train, cv=10)
pd.DataFrame(cs, columns=['validation_score'])
```

Out[12]:

	validation_score
0	0.664474
1	0.677632
2	0.715232
3	0.655629
4	0.655629
5	0.695364
6	0.668874
7	0.649007
8	0.721854
9	0.701987

In [13]:
```
print(f'Mean validation score {np.mean(cs):.3f}')
print(f'SD validation score {np.std(cs):.3f}')
```

```
Mean validation score 0.681
SD validation score 0.025
```

2(d)

rubric={points:2}

The above is useful, but we would like to see the training accuracy as well.

- Compute the 10-fold cross-validation again but this time using the `cross_validate` function with `return_train_score=True`.
- Print out both the cross-validation score and the training score.
- Is your cross-validation score exactly the same as what you got in the previous part? Very briefly discuss.

In [14]:
```
# Adapted from lecture 4
scores = cross_validate(model, X_train, y_train, cv=10, return_train_score=True)
```

```
print("Mean validation score %0.3f" % (np.mean(scores["test_score"])))
print("SD validation score %0.3f" % (np.std(scores["test_score"])))
print("Mean train score %0.3f" % (np.mean(scores["train_score"])))
pd.DataFrame(scores)
```

```
Mean validation score 0.674
SD validation score 0.030
Mean train score 0.999
```

Out[14]:

	fit_time	score_time	test_score	train_score
0	0.012688	0.000976	0.657895	0.999265
1	0.010712	0.000999	0.677632	1.000000
2	0.010736	0.001976	0.701987	1.000000
3	0.010738	0.001926	0.668874	0.999265
4	0.010736	0.001951	0.609272	0.999265
5	0.010708	0.001000	0.695364	0.999265
6	0.010735	0.000977	0.688742	0.999265
7	0.010764	0.001948	0.635762	0.999265
8	0.010736	0.000979	0.695364	0.999265
9	0.009732	0.001975	0.708609	0.999265

They are not the same. The splitting is done identically to the previous part, with StratifiedKFold with no shuffling, and the corresponding folds are the same as in the previous part. However, the features were permuted in different, "random", orders while fitting because `random_state` was not set. If the models were instantiated with a `random_state`, the model parameters and validation scores would have been the same as in the previous part.

2(e)

rubric={points:1}

Do you see a significant difference between the training score and the cross-validation score? Briefly discuss.

The difference is large. The large difference implies that the model is overfitted.

2(f)

rubric={points:1}

Inspect the 10 sub-scores from the 10 folds of cross-validation. How does this inform the trustworthiness of your cross validation score?

The standard deviation of the sub-scores is relatively small, and the low variance supports the trustworthiness of the cross validation score.

Exercise 3: Hyperparameters

rubric={points:10}

In this exercise, you'll experiment with the `max_depth` hyperparameter of the decision tree classifier. See the `DecisionTreeClassifier` documentation for more details.

- Explore the `max_depth` hyperparameter. Run 10-fold cross-validation for trees with different values of `max_depth` (at least 10 different values in the range 1 to 25).
- For each `max_depth`, get both the train accuracy and the cross-validation accuracy.
- Make a plot with `max_depth` on the *x*-axis and the train and cross-validation scores on the *y*-axis. That is, your plot should have two curves, one for train and one for cross-validation. Include a legend to specify which is which.
- Discuss how changing the `max_depth` hyperparameter affects the training and cross-validation accuracy. From these results, what depth would you pick as the optimal depth?
- Do you think that the depth you chose would generalize to other "spotify" datasets (i.e., data on other spotify users)?

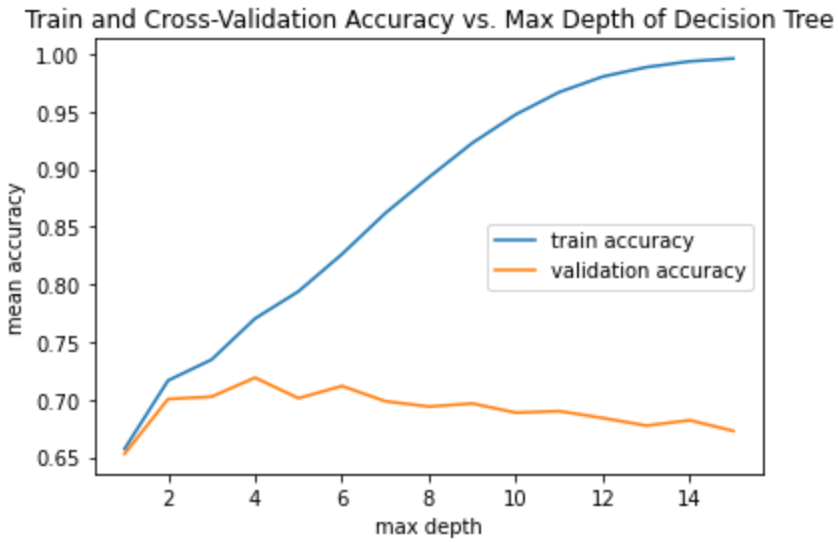
> Note: generally speaking (for all assignments) you are welcome to copy/paste code directly from the lecture notes, though I ask that you add a small citation (e.g. "Adapted from lecture 2") if you do so.

In [15]:
```
# Adapted from lecture 3

train_accuracies = []
validation_accuracies = []
max_depths = range(1, 16)

for max_depth in max_depths:
    model = DecisionTreeClassifier(max_depth=max_depth)
    scores = cross_validate(model, X_train, y_train, cv=10, return_train_score=True)
    validation_accuracies.append(np.mean(scores["test_score"]))
    train_accuracies.append(np.mean(scores["train_score"]))

pd.DataFrame(
    {"train_accuracies": train_accuracies, "validation_accuracies": validation_accuracies},
    index=max_depths
)
```

Out[15]:

	train_accuracies	validation_accuracies
1	0.657113	0.652771
2	0.716491	0.700379
3	0.734567	0.702366
4	0.770283	0.718935
5	0.793871	0.701046
6	0.826205	0.711650
7	0.861919	0.698432
8	0.893004	0.693835
9	0.922839	0.696467
10	0.947604	0.688493
11	0.967005	0.689840
12	0.980453	0.683884
13	0.988610	0.677283
14	0.993680	0.681932
15	0.996179	0.672630

In [16]:
```
plt.plot(max_depths, train_accuracies, label='train accuracy')
plt.plot(max_depths, validation_accuracies, label='validation accuracy')
plt.xlabel("max depth")
plt.ylabel("mean accuracy")
plt.title("Train and Cross-Validation Accuracy vs. Max Depth of Decision Tree")
plt.legend(loc="right")
plt.show()
```



I would pick 4 as the optimal depth as it maximizes the validation accuracy. At the depth of 4, the difference between the validation score and the train score is small, and the model would probably not overfit. Therefore the model would likely generalize to other datasets.

Exercise 4: Test set

rubric={points:4}

Remember the test set you created way back at the beginning of this assignment? Let's use it now to see if our cross-validation score from the previous exercise is trustworthy.

- Select your favorite `max_depth` from the previous part.
- Train a decision tree classifier using that `max_depth` on the *entire training set*.
- Compute and display the test score.
- How does it compare to the cross-validation score from the previous exercise? Briefly discuss.

In [17]:
```
model = DecisionTreeClassifier(max_depth=4)
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

Out[17]:
```
0.7029702970297029
```

It is similar to the score from the previous exercise. The similarity indicates that the model generalizes well.

Exercise 5: Conceptual questions

rubric={points:3}

Consider the dataset below, which has 6 examples and 2 features:

$$ X = \begin{bmatrix}5 & 2\\4 & 3\\ 2 & 2\\ 10 & 10\\ 9 & -1\\ 9& 9\end{bmatrix}, \quad y = \begin{bmatrix}-1\\-1\\+1\\+1\\+1\\+1\end{bmatrix}.$$

1. Say we fit a decision stump (depth 1 decision tree) and the first split is on the first feature (left column) being less than 5.5. What would we predict in the "true" and "false" cases here?
2. What training accuracy would the above stump get on this data set?
3. Can we obtain 100% accuracy with a single decision stump in this particular example?

1. true, true, true, false, false, false
2. 5/6; about 83.33%
3. No

Submission instructions

PLEASE READ: When you are ready to submit your assignment do the following:

1. Run all cells in your notebook to make sure there are no errors by doing `Kernel -> Restart Kernel and Clear All Outputs` and then `Run -> Run All Cells`.
2. Notebooks with cell execution numbers out of order or not starting from "1" will have marks deducted. Notebooks without the output displayed may not be graded at all (because we need to see the output in order to grade your work).
3. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission.