DSC 80 Discussion 8 Worksheet

1 FA22 Final Problem 7

	group	color	x	y
0	A	red	3	2
1	B	green	7	1
2	A	blue	2	5
3	A	red	5	3
4	B	blue	10	4
5	A	green	1	1

Consider the dataframe to the left. Suppose you wish to use this data in a linear regression model. To do so, the `color` column must be encoded numerically.

Problem 1.1. True or False: a meaningful way to numerically encode the `color` column is to replace each string by its index in the alphabetic ordering of the colors. That is, to replace `blue` by 1, `green` by 2, and `red` by 3.

[ ] True
[ ] False

Note that color is a nominal categorical varaible, meaning that there is no inherent ordering to the categories. Thus encoding the color variables by 1, 2, and 3 doesn't make any meaningful sense. I.e. it doesn't make sense to think of the color "red" as being "greater" than the color "blue" in any meaningful way.

Problem 1.2. `scikit-learn`'s `OneHotEncoder` module has a keyword called `drop=first`, which the documentation says will "drop the first category in each feature." What's the purpose of this keyword, and will using it lead to a worse linear classifier?

The purpose of drop='first' is to avoid perfectly collinear features, which can cause numerical problems in some models like unregularized linear regression. When you one-hot encode a categorical variable, all the columns sum to 1, creating redundancy. Dropping one category removes this redundancy.
Whether this leads to a worse classifier depends on the model type. For unregularized models, dropping a category helps and doesn't hurt performance. However, for penalized models (like regularized regression), dropping a category can introduce bias and may lead to slightly worse performance. The documentation notes that dropping "breaks the symmetry" of the encoding, which can affect models that use regularization.

2 FA22 Final Problem 8

Problem 2.1. Suppose you split a data set into a training set and a test set. You train a classifier on the training set and test it on the test set. **True or False**: the training accuracy must be higher than the test accuracy.

[ ] True [X] False

Training accuracy does NOT have to be higher than test accuracy, though we typically expect it to be. While models are optimized on the training data (which usually leads to higher training accuracy), it's entirely possible for test accuracy to exceed training accuracy due to random variation in how the data was split.
For example, the test set might happen to contain easier-to-classify examples, or the training set might contain more difficult or noisy examples. This is especially likely with small datasets or particular random splits. The word "must" in the question is the key—there's no guarantee that training accuracy will always be higher, only that we generally expect it to be higher on average across many random splits.

Problem 2.2. Suppose you train a model, but achieve much lower training and test accuracies than you expect. When you look at the data and make predictions yourself, you are easily able to achieve higher train and test accuracies. What should be done to improve the performance of the model?

Note: You haven't learned about decision trees yet (basically, just imagine a flow-chart), but for this question, all you need to know is that increasing `max_depth` *increases the complexity of your model.*

[ ] Decrease the `max_depth` hyperparameter; the model is "overfitting".
[X] Increase the `max_depth` hyperparameter; the model is "underfitting".

This is a clear case of underfitting. The key clue is that BOTH training accuracy and test accuracy are low—the model isn't even performing well on the data it was trained on. This indicates the model is too simple to capture the underlying patterns in the data.
The fact that you can manually achieve higher accuracies on both sets confirms that the patterns are learnable; the model just isn't complex enough to learn them. This is the definition of underfitting: when a model is too simple to capture the relationships in the data.

3 SP22 Final Problem 10

The DataFrame `new_releases` contains the following information for songs that were recently released. The first few rows are shown below.

	genre	rec_label	danceability	speechiness	first_month
0	Hip-Hop/Rap	EMI	0.39	0.84	12019896
1	Pop	UMG	0.91	0.65	9932385
2	Pop	EMI	0.65	0.71	10923584
3	Country	SME	0.45	0.93	8107742
4	Hip-Hop/Rap	UMG	0.39	0.86	9554136

- `genre`: one of the following five possibilities: `Hip-Hop/Rap`, `Pop`, `Country`, `Alternative`, or `International`
- `rec_label`: the label that released the song (one of the following 4: `EMI`, `SME`, `UMG`, or `WMG`)

- `danceability`: how easy the song is to dance to, according to the Spotify API (between 0 and 1)
- `speechiness`: what proportion of the song is made up of spoken words, according to the Spotify API
- `first_month`: the number of total streams the song had on Spotify in the first month it was released

To start, we conduct a train-test split, splitting `new_releases` into `X_train`, `X_test`, `y_train`, and `y_test`. We first fit a linear model to the training data that only uses `danceability`, and call this model `lr_one`.

Problem 3.1. True or False: If `lr_one.score(X_train, y_train)` is much lower than `lr_one.score(X_test, y_test)`, it is likely that `lr_one` overfit to the training data.

`[ ] True [X] False`

This is NOT overfitting. Overfitting occurs when training accuracy is HIGH but test accuracy is LOW—the model memorizes the training data instead of learning generalizable patterns. In this scenario, training score is LOWER than test score, which is the opposite. This could happen due to random variation in the train-test split (e.g., the test set happened to be easier), but it doesn't indicate overfitting.

```
>>> X_train.shape[0]
50
>>> np.sum((y_train - lr_one.predict(X_train)) ** 2)
500000 # five hundred thousand
```

Problem 3.2. Given this output, what is `lr_one`'s training RMSE? Give your answer as an integer.

RMSE (Root Mean Squared Error) is calculated by taking the sum of squared errors, dividing by the number of samples, and then taking the square root. We're given that the sum of squared errors is 500,000 and there are 50 training samples. So we calculate: 500,000 divided by 50 equals 10,000. The square root of 10,000 is 100. Therefore, the training RMSE is 100.

Now, suppose we fit one more linear model (with an intercept term) to the training data:

- Model 2 (`lr_no_drop`): Uses `danceability` and `speechiness` as-is, and one-hot encodes `genre` and `rec_label`, using `OneHotEncoder()`. (Note the lack of the `drop_first=True` keyword.)

Suppose we are given the following coefficients in Model 2:

- The coefficient on `genre_Pop` is 2000.
- The coefficient on `genre_Country` is 1000.
- The coefficient on `danceability` is $10^6 = 1,000,000$

Problem 3.3. Daisy and Billy are two artists signed to the same `rec_label` who each just released a new song with the same `speechiness`. Daisy is a `Pop` artist while Billy is a `Country` artist.

Model 2 predicted that Daisy's song and Billy's song will have the same `first_month` streams. What is the absolute difference between Daisy's song's `danceability` and Billy's song's `danceability`? Give your

answer as a simplified fraction.

Model 4 is made up of 11 features, i.e. 11 columns.

- 4 of the columns correspond to the different values of `"rec_label"`. Since Daisy and Billy have the same `"rec_label"`, their values in these four columns are all the same.
- One of the columns corresponds to `"speechiness"`. Since Daisy's song and Billy's song have the same `"speechiness"`, their values in this column are the same.
- 5 of the columns correspond to the different values of `"genre"`. Daisy is a `"Pop"` artist, so she has a 1 in the `"genre_Pop"` column and a 0 in the other four `"genre_"` columns, and similarly Billy has a 1 in the `"genre_Country"` column and 0s in the others.
- One of the columns corresponds to `"danceability"`, and Daisy and Billy have different quantitative values in this column.

Let d_1 and d_2.

The key is in recognizing that all features in Daisy's prediction and Billy's prediction are the same, other than the coefficients on `"genre_Pop"`, `"genre_Country"`, and `"danceability"`. Let's let d_1 be Daisy's song's `"danceability"`, and let d_2 be Billy's song's `"danceability"`. Then:

$$\begin{aligned} 2000 + 10^6 \cdot d_1 &= 1000 + 10^6 \cdot d_2 \\ 1000 &= 10^6 (d_2 - d_1) \\ \frac{1}{1000} &= d_2 - d_1 \end{aligned}$$

Thus, the absolute difference between their songs' `"danceability"`s is $\frac{1}{1000}$.