CPSC 330 - Applied Machine Learning

Homework 4: Logistic regression, hyperparameter optimization

Associated lectures: Lectures 7, 8

Due date: Wednesday, June 01, 2022 at 18:00

Table of Contents

Imports

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

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

from sklearn.dummy import DummyClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import (
    GridSearchCV,
    cross_val_score,
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.tree import DecisionTreeClassifier
```

Instructions

---

rubric={points:5}

Follow the homework submission instructions.

You may work with a partner on this homework and submit your assignment as a group. Below are some instructions on working as a group.

- The maximum group size is 2.
- Use group work as an opportunity to collaborate and learn new things from each other.
- Be respectful to each other and make sure you understand all the concepts in the assignment well.
- It's your responsibility to make sure that the assignment is submitted by one of the group members before the deadline.
- You can find the instructions on how to do group submission on Gradescope here.

Note: The assignments will get gradually more open-ended as we progress through the course. In many cases, there won't be a single correct solution. Sometimes you will have to make your own choices and your own decisions (for example, on what parameter values to use when they are not explicitly provided in the instructions). Use your own judgment in such cases and justify your choices, if necessary.

Exercise 1: implementing `DummyClassifier`

---

rubric={points:25}

In this course (unlike CPSC 340) you will generally **not** be asked to implement machine learning algorihtms (like logistic regression) from scratch. However, this exercise is an exception: you will implement the simplest possible classifier, `DummyClassifier`.

As a reminder, `DummyClassifier` is meant as a baseline and is generally the worst possible "model" you could "fit" to a dataset. All it does is predict the most popular class in the training set. So if there are more 0s than 1s it predicts 0 every time, and if there are more 1s than 0s it predicts 1 every time. For `predict_proba` it looks at the frequencies in the training set, so if you have 30% 0's 70% 1's it predicts `[0.3 0.7]` every time. Thus, `fit` only looks at `y` (not `X`).

Below you will find starter code for a class called `MyDummyClassifier`, which has methods `fit()`, `predict()`, `predict_proba()` and `score()`. Your task is to fill in those four functions. To get you started, I have given you a `return` statement in each case that returns the correct data type: `fit` can return nothing, `predict` returns an array whose size is the number of examples, `predict_proba` returns an array whose size is the number of examples x 2, and `score` returns a number.

The next code block has some tests you can use to assess whether your code is working.

I suggest starting with `fit` and `predict`, and making sure those are working before moving on to `predict_proba`. For `predict_proba`, you should return the frequency of each class in the training data, which is the behaviour of `DummyClassifier(strategy='prior')`. Your `score` function should call your `predict` function. Again, you can compare with `DummyClassifier` using the code below.

To simplify this question, you can assume **binary classification**, and furthermore that these classes are **encoded as 0 and 1**. In other words, you can assume that `y` contains only 0s and 1s. The real `DummyClassifier` works when you have more than two classes, and also works if the target values are encoded differently, for example as "cat", "dog", "mouse", etc.

In [2]:
```python
class MyDummyClassifier:
    """
    A baseline classifier that predicts the most common class.
    The predicted probabilities come from the relative frequencies
    of the classes in the training data.

    This implementation only works when y only contains 0s and 1s.
    """

    def fit(self, X, y):
        ### BEGIN SOLUTION
        self.prob_1 = np.mean(y == 1)
        ### END SOLUTION
        return None  # Replace with your code

    def predict(self, X):
        ### BEGIN SOLUTION
        if self.prob_1 >= 0.5:
            return np.ones(X.shape[0])
        else:
            return np.zeros(X.shape[0])
        ### END SOLUTION
        return np.zeros(X.shape[0])  # Replace with your code

    def predict_proba(self, X):
        ### BEGIN SOLUTION
        probs = np.zeros((X.shape[0], 2))
        probs[:, 0] = 1 - self.prob_1
        probs[:, 1] = self.prob_1
        return probs
        ### END SOLUTION
        return np.zeros((X.shape[0], 2))  # Replace with your code

    def score(self, X, y):
        ### BEGIN SOLUTION
        return np.mean(self.predict(X) == y)
        ### END SOLUTION
        return 0.0  # Replace with your code
```

Below are some tests for `predict` using randomly generated data. You may want to run the cell a few times to make sure you explore the different cases (or automate this with a loop or random seeds).

In [3]:
```python
# For testing, generate random data
n_train = 101
n_valid = 21
d = 5
X_train_dummy = np.random.randn(n_train, d)
X_valid_dummy = np.random.randn(n_valid, d)
y_train_dummy = np.random.randint(2, size=n_train)
y_valid_dummy = np.random.randint(2, size=n_valid)

my_dc = MyDummyClassifier()
sk_dc = DummyClassifier(strategy="prior")

my_dc.fit(X_train_dummy, y_train_dummy)
sk_dc.fit(X_train_dummy, y_train_dummy)

assert np.array_equal(my_dc.predict(X_train_dummy), sk_dc.predict(X_train_dummy))
assert np.array_equal(my_dc.predict(X_valid_dummy), sk_dc.predict(X_valid_dummy))
```

Below are some tests for `predict_proba`.

In [4]:
```
assert np.allclose(
    my_dc.predict_proba(X_train_dummy), sk_dc.predict_proba(X_train_dummy)
)
assert np.allclose(
    my_dc.predict_proba(X_valid_dummy), sk_dc.predict_proba(X_valid_dummy)
)
```

Below are some tests for `score`.

In [5]:
```
assert np.isclose(
    my_dc.score(X_train_dummy, y_train_dummy), sk_dc.score(X_train_dummy, y_train_dummy)
)
assert np.isclose(
    my_dc.score(X_valid_dummy, y_valid_dummy), sk_dc.score(X_valid_dummy, y_valid_dummy)
)
```

Exercise 2: Trump Tweets

For the rest of this assignment we'll be looking at a dataset of Donald Trump's tweets as of June 2020. You should start by downloading the dataset. Unzip it and move the file `realdonaldtrump.csv` into this directory. As usual, please do not submit the dataset when you submit the assignment.

In [6]:
```
tweets_df = pd.read_csv("realdonaldtrump.csv", index_col=0)
tweets_df.head()
```

Out[6]:

id	link	content	date	retweets	favorites	mentions	ha
1698308935	https://twitter.com/realDonaldTrump/status/169...	Be sure to tune in and watch Donald Trump on L...	2009-05-04 13:54:25	510	917	NaN	
1701461182	https://twitter.com/realDonaldTrump/status/170...	Donald Trump will be appearing on The View tom...	2009-05-04 20:00:10	34	267	NaN	
1737479987	https://twitter.com/realDonaldTrump/status/173...	Donald Trump reads Top Ten Financial Tips on L...	2009-05-08 08:38:08	13	19	NaN	
1741160716	https://twitter.com/realDonaldTrump/status/174...	New Blog Post: Celebrity Apprentice Finale and...	2009-05-08 15:40:15	11	26	NaN	
1773561338	https://twitter.com/realDonaldTrump/status/177...	"My persona will never be that of a wallflower...	2009-05-12 09:07:28	1375	1945	NaN	

In [7]:
```
tweets_df.shape
```

Out[7]:
```
(43352, 7)
```

We will be trying to predict whether a tweet will go "viral", defined as having more than 10,000 retweets:

In [8]:
```
y = tweets_df["retweets"] > 10_000
```

To make predictions, we'll be using only the content (text) of the tweet.

In [9]:
```
X = tweets_df["content"]
```

For the purpose of this assignment, you can ignore all the other columns in the original dataset.

2(a) ordering the steps

rubric={points:8}

Let's start by building a model using `CountVectorizer` and `LogisticRegression`. The code required to do this has been provided below, but in the wrong order.

- Rearrange the lines of code to correctly fit the model and compute the cross-validation score.

- Add a short comment to each block to describe what the code is doing.

YOUR COMMENT HERE countvec = CountVectorizer(stop_words="english") # YOUR COMMENT HERE X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=321) # YOUR COMMENT HERE cross_val_results = pd.DataFrame(cross_validate(pipe, X_train, y_train, return_train_score=True)) # YOUR COMMENT HERE pipe = make_pipeline(countvec, lr) # YOUR COMMENT HERE cross_val_results.mean() # YOUR COMMENT HERE lr = LogisticRegression(max_iter=1000)

In [10]:
```
### BEGIN SOLUTION
# 1. Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=321)

# 2. Create the vectorizer object
countvec = CountVectorizer(stop_words="english")

# 3. Create the log reg object (could switch with Step 2)
lr = LogisticRegression(max_iter=1000)

# 4. Create the Pipeline
pipe = make_pipeline(countvec, lr)

# 5. Compute the cross-validation scores across folds
cross_val_results = pd.DataFrame(
    cross_validate(pipe, X_train, y_train, return_train_score=True)
)

# 6. Average cross-validation scores
cross_val_results.mean()
### END SOLUTION
```

Out[10]:
```
fit_time       3.845708
score_time     0.407419
test_score     0.897890
train_score    0.967045
dtype: float64
```

2(b) Cross-validation fold sub-scores

rubric={points:5}

Above we averaged the scores from the 5 folds of cross-validation.

- Print out the 5 individual scores. Reminder: `sklearn` calls them `"test_score"` but they are really (cross-)validation scores.
- Are the 5 scores close to each other or spread far apart? (This is a bit subjective, answer to the best of your ability.)
- How does the size of this dataset (number of rows) compare to the cities dataset we have been using in class? How does this relate to the different sub-scores from the 5 folds?

BEGIN SOLUTION

In [11]:
```
cross_val_results["test_score"]
```

Out[11]:
```
0    0.899123
1    0.899739
2    0.896356
3    0.898201
4    0.896032
Name: test_score, dtype: float64
```

I would say they are quite close together, e.g. max-min =

In [12]:
```
cross_val_results["test_score"].max() - cross_val_results["test_score"].min()
```

Out[12]:
```
0.003706592035959244
```

is quite small. This is likely due to the fairly large dataset:

In [13]:
```
len(X_train)
```

Out[13]:
```
32514
```

it is much bigger than the cities dataset which has only 200 rows. Generally, larger datasets will give us more reliable validation results because the randomness of the splits plays less of a role.

END SOLUTION

2(c) baseline

rubric={points:3}

By the way, are these scores any good?

- Run `DummyClassifier` (or `MyDummyClassifier`!) on this dataset.
- Compare the `DummyClassifier` score to what you got from logistic regression above. Does logistic regression seem to be doing anything useful?
- Is it necessary to use `CountVectorizer` here? Briefly explain.

BEGIN SOLUTION

In [14]:
```
dc = DummyClassifier(strategy="prior")
pd.DataFrame(cross_validate(dc, X_train, y_train, return_train_score=True)).mean()
```

Out[14]:
```
fit_time       0.006978
score_time     0.001551
test_score     0.738543
train_score    0.738543
dtype: float64
```

- The train/cross-validation scores are around 74%, which represents the fact that about 26% of the tweets in the training set are viral and 74% non-viral.
- The logistic regression is getting around 90% accuracy so that is clearly an improvement over 74%.
- No, we didn't need `CountVectorizer` because `DummyClassifier` doesn't even look at X.

END SOLUTION

2(d) probability scores

rubric={points:5}

Here we train a logistic regression classifier on the entire training set:

(Note: this is relying on the `pipe` variable from 2(a) - you'll need to redefine it if you overwrote that variable in between.)

In [15]:
```
pipe.fit(X_train, y_train);
```

Using this model, find the tweet in the **test set** with the highest predicted probability of being viral. Print out the tweet and the associated probability score.

Reminder: you are free to reuse/adapt code from lecture. Please add in a small attribution, e.g. "From Lecture 7".

BEGIN SOLUTION

In [16]:
```
np.max(pipe.predict_proba(X_test)[:, 1])
```

Out[16]:
```
0.9999999325254757
```

In [17]:
```
most_positive_ind = np.argmax(pipe.predict_proba(X_test)[:, 1])

print(X_test.iloc[most_positive_ind])
```

```
Corrupt politician Adam Schiff wants people from the White House to testify in his and Pelosi’s disgraceful Witch Hunt, yet he will not allow a White House lawyer, nor will he allow ANY of our requested witnesses. This is a first in due process and Congressional history!
```

END SOLUTION

2(e) coefficients

rubric={points:4}

We can extract the `CountVectorizer` and `LogisticRegression` objects from the `make_pipeline` object as follows:

In [18]:
```
vec_from_pipe = pipe.named_steps["countvectorizer"]
lr_from_pipe = pipe.named_steps["logisticregression"]
```

Using these extracted components above, display the 5 words with the highest coefficients and the 5 words with the smallest coefficients.

BEGIN SOLUTION

In [19]:
```
words_weights_df = pd.DataFrame(
    data=lr_from_pipe.coef_.ravel(),
    index=vec_from_pipe.get_feature_names_out(),
    columns=["Coefficient"],
)
words_weights_df.sort_values(by="Coefficient", ascending=False)
```

Out[19]:

	Coefficient
harassment	2.731765
mini	2.712435
fake	2.692779
coronavirus	2.434255
transcripts	2.380497
...	...
1pic	-2.295103
trump2016	-2.316256
barackobama	-2.565427
trump2016pic	-2.637239
realdonaldtrump	-3.116974

40965 rows × 1 columns

END SOLUTION

2(f)

rubric={points:10}

scikit-learn provides a lot of useful tools like `make_pipeline` and `cross_validate`, which are awesome. But with these fancy tools it's also easy to lose track of what is actually happening under the hood. Here, your task is to "manually" (without `Pipeline` and without `cross_validate` or `cross_val_score`) compute logistic regression's validation score on one fold (that is, train on 80% and validate on 20%) of the training data.

You should start with the following `CountVectorizer` and `LogisticRegression` objects, as well as `X_train` and `y_train` (which you should further split):

In [20]:
```
countvec = CountVectorizer(stop_words="english")
lr = LogisticRegression(max_iter=1000)
```

Meta-comment: you might be wondering why we're going into "implementation" here if this course is about *applied* ML. In CPSC 340, we would go all the way down into `LogisticRegression` and understand how `fit` works, line by line. Here we're not going into that at all, but I still think this type of question (and

Exercise 1) is a useful middle ground. I do want you to know what is going on in `Pipeline` and in `cross_validate` even if we don't cover the details of `fit`. To get into logistic regression's `fit` requires a bunch of math; here, we're keeping it more conceptual and avoiding all those prerequisites.

BEGIN SOLUTION

In [21]:
```
X_train_fold, X_valid_fold, y_train_fold, y_valid_fold = train_test_split(
    X_train, y_train, test_size=0.2
)

X_train_fold_vec = countvec.fit_transform(X_train_fold)
X_valid_fold_vec = countvec.transform(X_valid_fold)

lr.fit(X_train_fold_vec, y_train_fold)
lr.score(X_valid_fold_vec, y_valid_fold)
```

Out[21]:
```
0.9009687836383208
```

END SOLUTION

Exercise 3: hyperparameter optimization

3(a)

rubric={points:4}

The following code varies the `max_features` hyperparameter of `CountVectorizer` and makes a plot (with the x-axis on a log scale) that shows train/cross-validation scores vs. `max_features`. It also prints the results. Based on the plot/output, what value of `max_features` seems best? Briefly explain.

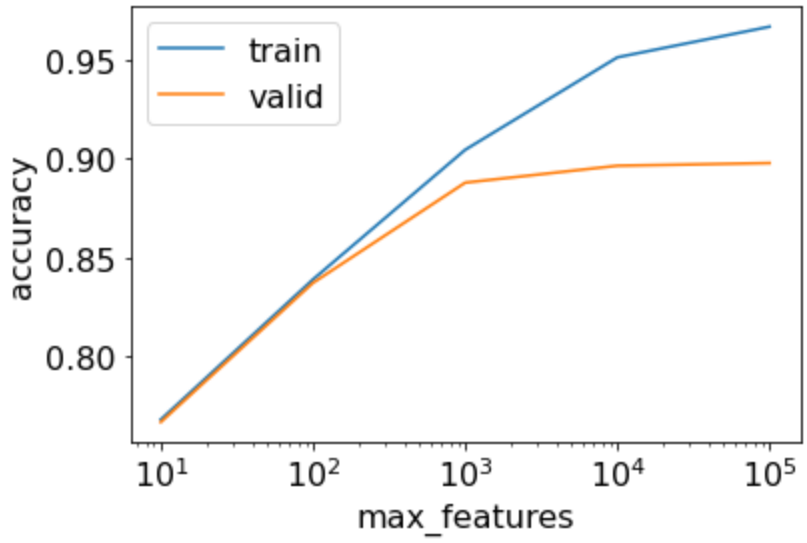
Note: the code may take a minute or two to run. You can uncomment the `print` statement if you want to see it show the progress.

In [22]:
```
train_scores = []
cv_scores = []

max_features = [10, 100, 1000, 10_000, 100_000]

for mf in max_features:
    #     print(mf)
    pipe = make_pipeline(
        CountVectorizer(stop_words="english", max_features=mf),
        LogisticRegression(max_iter=1000),
    )
    cv_results = cross_validate(pipe, X_train, y_train, return_train_score=True)
    train_scores.append(cv_results["train_score"].mean())
    cv_scores.append(cv_results["test_score"].mean())
```

```
plt.semilogx(max_features, train_scores, label="train")
plt.semilogx(max_features, cv_scores, label="valid")
plt.legend()
plt.xlabel("max_features")
plt.ylabel("accuracy");
```



In [23]:
```
pd.DataFrame({"max_features": max_features, "train": train_scores, "cv": cv_scores})
```

Out[23]:

	max_features	train	cv
0	10	0.767854	0.766593
1	100	0.838900	0.837147
2	1000	0.904618	0.887956
3	10000	0.951506	0.896537
4	100000	0.967045	0.897890

BEGIN SOLUTION

In terms of cross-validation score, it looks like the best is `max_features=100_000`. In this case that means using all the words, since the total number of words is less than 100,000:

In [24]:
```
len(CountVectorizer(stop_words="english").fit(X_train, y_train).get_feature_names_out())
```

Out[24]:
```
40965
```

END SOLUTION

3(b)

rubric={points:4}

The following code varies the `C` hyperparameter of `LogisticRegression` and makes a plot (with the x-axis on a log scale) that shows train/cross-validation scores vs. `C`. Based on the plot, what value of `C` seems best?

Note: the code may take a minute or two to run. You can uncomment the `print` statement if you want to see it show the progress.

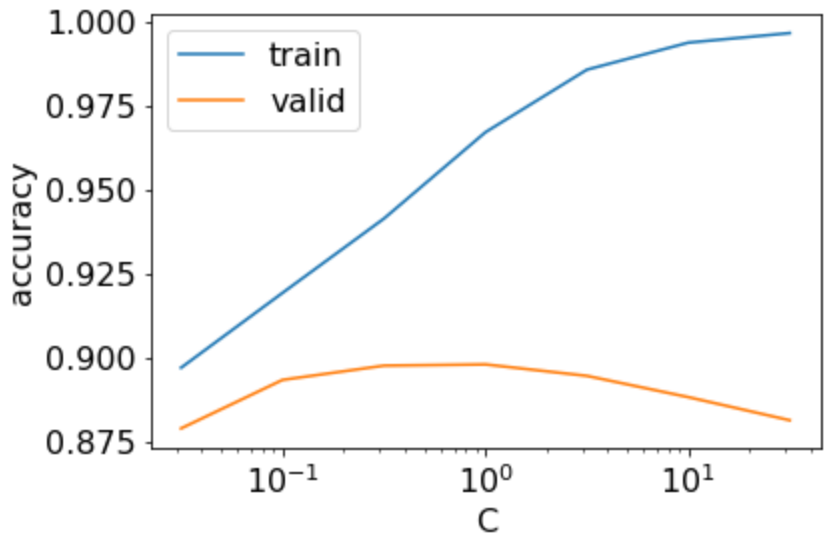
In [25]:
```
train_scores = []
cv_scores = []

C_vals = 10.0 ** np.arange(-1.5, 2, 0.5)

for C in C_vals:
    #     print(C)
    pipe = make_pipeline(
        CountVectorizer(stop_words="english", max_features=None),
        LogisticRegression(max_iter=1000, C=C),
    )
    cv_results = cross_validate(pipe, X_train, y_train, return_train_score=True)

    train_scores.append(cv_results["train_score"].mean())
    cv_scores.append(cv_results["test_score"].mean())

plt.semilogx(C_vals, train_scores, label="train")
plt.semilogx(C_vals, cv_scores, label="valid")
plt.legend()
plt.xlabel("C")
plt.ylabel("accuracy");
```



In [26]:
```
pd.DataFrame({"C": C_vals, "train": train_scores, "cv": cv_scores})
```

Out[26]:

	C	train	cv
0	0.031623	0.896898	0.878821
1	0.100000	0.919196	0.893277
2	0.316228	0.941333	0.897521
3	1.000000	0.967045	0.897890
4	3.162278	0.985675	0.894476
5	10.000000	0.993733	0.888140
6	31.622777	0.996578	0.881251

BEGIN SOLUTION

In [27]:
```
pd.DataFrame({"C": C_vals, "train": train_scores, "valid": cv_scores}).sort_values(
    by="valid", ascending=False
)
```

Out[27]:

	C	train	valid
3	1.000000	0.967045	0.897890
2	0.316228	0.941333	0.897521
4	3.162278	0.985675	0.894476
1	0.100000	0.919196	0.893277
5	10.000000	0.993733	0.888140
6	31.622777	0.996578	0.881251
0	0.031623	0.896898	0.878821

It looks like the best value of `C` is 1, though $C \approx 0.3$ gives a very similar cross-validation score.

END SOLUTION

3(c)

rubric={points:12}

- Using `GridSearchCV`, jointly optimize `max_features` and `C` across all the combinations of values we tried above.
  - Note: the code might be a bit slow here.
  - Setting `n_jobs=-1` should speed it up if you have a multi-core processor.
  - You can reduce the number of folds (e.g. `cv=2`) to speed it up if necessary.
- What are the best values of `max_features` and `C` according to your grid search?
- Do these best values agree with what you found in parts (a) and (b)?
- Generally speaking, *should* these values agree with what you found in parts (a) and (b)? Explain.

BEGIN SOLUTION

In [28]:
```
pipeline = make_pipeline(
    CountVectorizer(stop_words="english"),
    LogisticRegression(max_iter=1000),
)

param_grid = {
    "countvectorizer__max_features": max_features,
    "logisticregression__C": C_vals,
}
```

In [29]:
```
grid_search = GridSearchCV(pipeline, param_grid, verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 35 candidates, totalling 175 fits
```

In [30]:
```
grid_search.best_params_
```

Out[30]:
```
{'countvectorizer__max_features': 100000, 'logisticregression__C': 1.0}
```

In [31]:
```
grid_search.best_score_
```

Out[31]:
```
0.8978900824847041
```

They do agree. The situation here is quite nuanced. In general there is no reason they need to agree - by jointly optimizing the hyperparameters you might find something better. So why did they agree in this case - just luck? Well, no, not quite!! Here, as it turns out, the optimal values of the hyperparameters turned out to be the default values of `max_features=None` and `C=1.0`. (I guess the developers of scikit-learn picked good defaults!) So, that means when we were individually optimizing `max_features` we were already using the optimal (default) value of `C` and likewise for optimizing `C`. This made the one-at-a-time hyperparameter optimization more effective than it would be in general.

END SOLUTION

3(d)

rubric={points:5}

- Evaluate your final model on the test set.
- How does your test accuracy compare to your validation accuracy?
- If they are different: do you think this is because you "overfitted on the validation set", or simply random luck?

BEGIN SOLUTION

In [32]:
```
grid_search.score(X_test, y_test)
```

Out[32]:
```
0.8992434028418528
```

The test score is very close to the cross-validation score, and in fact the test score is slightly higher, which confirms that this is due to luck/randomness.

END SOLUTION

Exercise 4: Very short answer questions

rubric={points:10}

Each question is worth 2 points. Max 2 sentences per answer.

1. What is the problem with calling `fit_transform` on your test data with `CountVectorizer`?
2. Why is it important to follow the Golden Rule? If you violate it, will that give you a worse classifier?
3. If you could only access one of `predict` or `predict_proba`, which one would you choose? Briefly explain.
4. What are two advantages of using sklearn `Pipeline`s?
5. What are two advantages of `RandomizedSearchCV` over `GridSearchCV`?

BEGIN SOLUTION

1. You need to perform the same transformations on the train and test data, otherwise the results will not make sense.
2. Not necessarily a worse classifier, but you'll get an overly optimistic estimate of your model performance when you compute test accuracy - which is bad.
3. `predict_proba`. From here you can get the output of `predict`, but not the other way around.
4. (1) prevents violating the Golden Rule, (2) helps keep track of all your transformations in one place.
5. (1) you can directly choose the number of experiments, (2) avoids the "irrelevant hyperparameter" issue where experiments are wasted (see lecture).

END SOLUTION

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.