hyperparameter tuning

November 2, 2022

1 Mudcard

- **How do we find the range for C and gamma for hyperparameter tuning of SVMs? Do we need to visualize the data first?**
- **For SVR, how do we know what the width of the gaussian should be? Is it better if the width is as low as possible?**
  - it's a hyperparameter (gamma) so as usual, you calculate train and validation scores
- **In the case of regression, are SVM's exactly the same as kernel density estimation?**
  - it's very similar but the goal is different
  - in KDE, your goal is to plot a smooth distribution instead of a histogram
  - in SVR rbf, your goal is to predict the regression target variable for previously unseen points
- **Muddiest part was understanding how summing different gaussian functions result in the final prediction function for SVR. Is this summing similar to the Taylor series of a function?**
  - nope
  - it's quite literally just replacing each point with a gaussian and the model prediction is the sum of the gaussians
- **I am still confused how widening the Gaussian predictions creates such a smooth curve for predictions in SVMs.**
  - implement the algorithm yourself to figure it out
  - it's not too difficult and it's a great exercise to deepen your understanding
- **Which library or method would you recommend if we want to check the memory used?**
  - as usual, ask stackoverflow
- **Could you please post the codes for the quiz 2?**
  - once you submit your solution, the code should show up in canvas
- **I'm still unclear about the quiz question:¬†The random forest run-time scales linearly with n_samples? I couldn't tell the linear scaling from the graph or the run time values.**
  - you might need to average the runtime of multiple fits or use more datapoints to see it well

1.1 The supervised ML pipeline

The goal: Use the training data (X and y) to develop a model which can accurately predict the target variable (y_new') for previously unseen data (X_new).

1. Exploratory Data Analysis (EDA): you need to understand your data and verify that it doesn't contain errors - do as much EDA as you can!

2. Split the data into different sets: most often the sets are train, validation, and test (or holdout) - practitioners often make errors in this step! - you can split the data randomly, based on groups, based on time, or any other non-standard way if necessary to answer your ML question

3. Preprocess the data: ML models only work if X and Y are numbers! Some ML models additionally require each feature to have 0 mean and 1 standard deviation (standardized features) - often the original features you get contain strings (for example a gender feature would contain 'male', 'female', 'non-binary', 'unknown') which needs to transformed into numbers - often the features are not standardized (e.g., age is between 0 and 100) but it needs to be standardized

4. Choose an evaluation metric: depends on the priorities of the stakeholders - often requires quite a bit of thinking and ethical considerations

5. Choose one or more ML techniques: it is highly recommended that you try multiple models - start with simple models like linear or logistic regression - try also more complex models like nearest neighbors, support vector machines, random forest, etc.

6. Tune the hyperparameters of your ML models (aka cross-validation) - ML techniques have hyperparameters that you need to optimize to achieve best performance - for each ML model, decide which parameters to tune and what values to try - loop through each parameter combination - train one model for each parameter combination - evaluate how well the model performs on the validation set - take the parameter combo that gives the best validation score - evaluate that model on the test set to report how well the model is expected to perform on previously unseen data

7. Interpret your model: black boxes are often not useful - check if your model uses features that make sense (excellent tool for debugging) - often model predictions are not enough, you need to be able to explain how the model arrived to a particular prediction (e.g., in health care)

1.2 Let's put everything together

- IID data first!
- the adult dataset
- the next two cells were copied from the week 3 material and slightly rewritten

import packages

load your dataset

create feature matrix and target variable

for i in random_states:

- split the data
- preprocess it
- decide which hyperparameters you'll tune and what values you'll try
- for combo in hyperparameters:
  - train your ML algo
  - calculate validation scores
- select best model based on the mean and std validation scores
- predict the test set using the best model

- return your test score (generalization error)

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder,␣
↪OrdinalEncoder, MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import ParameterGrid
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

df = pd.read_csv('data/adult_data.csv')

# let's separate the feature matrix X, and target variable y
y = df['gross-income'] # remember, we want to predict who earns more than 50k␣
↪or less than 50k
X = df.loc[:, df.columns != 'gross-income'] # all other columns are features

# collect which encoder to use on each feature
# needs to be done manually
ordinal_ftrs = ['education']
ordinal_cats = [[' Preschool',' 1st-4th',' 5th-6th',' 7th-8th',' 9th',' 10th','␣
↪11th',' 12th',' HS-grad',\
                ' Some-college',' Assoc-voc',' Assoc-acdm',' Bachelors','␣
↪Masters',' Prof-school',' Doctorate']]
onehot_ftrs =␣
↪['workclass','marital-status','occupation','relationship','race','sex','native-country']
minmax_ftrs = ['age','hours-per-week']
std_ftrs = ['capital-gain','capital-loss']

# collect all the encoders into one preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('ord', OrdinalEncoder(categories = ordinal_cats), ordinal_ftrs),
        ('onehot', OneHotEncoder(sparse=False,handle_unknown='ignore'),␣
↪onehot_ftrs),
        ('minmax', MinMaxScaler(), minmax_ftrs),
        ('std', StandardScaler(), std_ftrs)])

prep = Pipeline(steps=[('preprocessor', preprocessor)]) # for now we only␣
↪preprocess, later we will add other steps here
```

1.3 Quiz

Let's recap preprocessing. Which of these statements are true?

1.4 Basic hyperparameter tuning

[2]:
```
# let's train a random forest classifier

# we will loop through nr_states random states so we will return nr_states test␣
↪scores and nr_states trained models
nr_states = 5
test_scores = np.zeros(nr_states)
final_models = []

# loop through the different random states
for i in range(nr_states):
    print('randoms state '+str(i+1))

    # first split to separate out the training set
    X_train, X_other, y_train, y_other = train_test_split(X,y,train_size = 0.
↪6,random_state=42*i)

    # second split to separate out the validation and test sets
    X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,train_size␣
↪= 0.5,random_state=42*i)

    # preprocess the sets
    X_train_prep = prep.fit_transform(X_train)
    X_val_prep = prep.transform(X_val)
    X_test_prep = prep.transform(X_test)

    # decide which parameters to tune and what values to try
    # the default value of any parameter not specified here will be used
    param_grid = {
                  'max_depth': [1, 3, 10, 30, 100], # no upper bound so the␣
↪values are evenly spaced in log
                  'max_features': [0.25, 0.5,0.75,1.0] # linearly spaced␣
↪because it is between 0 and 1, 0 is omitted
                  }

    # we save the train and validation scores
    # the validation scores are necessary to select the best model
    # it's optional to save the train scores, it can be used to identify high␣
↪bias and high variance models
    train_score = np.zeros(len(ParameterGrid(param_grid)))
    val_score = np.zeros(len(ParameterGrid(param_grid)))
    models = []
```

```python
    # loop through all combinations of hyperparameter combos
    for p in range(len(ParameterGrid(param_grid))):
        params = ParameterGrid(param_grid)[p]
        print('   ',params)
        clf = RandomForestClassifier(**params,random_state = 42*i,n_jobs=-1) # initialize the classifier
        clf.fit(X_train_prep,y_train) # fit the model
        models.append(clf) # save it
        # calculate train and validation accuracy scores
        y_train_pred = clf.predict(X_train_prep)
        train_score[p] = accuracy_score(y_train,y_train_pred)
        y_val_pred = clf.predict(X_val_prep)
        val_score[p] = accuracy_score(y_val,y_val_pred)
        print('   ',train_score[p],val_score[p])

    # print out model parameters that maximize validation accuracy
    print('best model parameters:',ParameterGrid(param_grid)[np.argmax(val_score)])
    print('corresponding validation score:',np.max(val_score))
    # collect and save the best model
    final_models.append(models[np.argmax(val_score)])
    # calculate and save the test score
    y_test_pred = final_models[-1].predict(X_test_prep)
    test_scores[i] = accuracy_score(y_test,y_test_pred)
    print('test score:',test_scores[i])
```

```
randoms state 1
    {'max_features': 0.25, 'max_depth': 1}
    0.7599815724815725 0.7581388206388207
    {'max_features': 0.5, 'max_depth': 1}
    0.7599815724815725 0.7581388206388207
    {'max_features': 0.75, 'max_depth': 1}
    0.7599815724815725 0.7581388206388207
    {'max_features': 1.0, 'max_depth': 1}
    0.7599815724815725 0.7581388206388207
    {'max_features': 0.25, 'max_depth': 3}
    0.8408579033579033 0.8413697788697788
    {'max_features': 0.5, 'max_depth': 3}
    0.8433149058149059 0.8465909090909091
    {'max_features': 0.75, 'max_depth': 3}
    0.842956592956593 0.8459766584766585
    {'max_features': 1.0, 'max_depth': 3}
    0.8421375921375921 0.8456695331695332
    {'max_features': 0.25, 'max_depth': 10}
    0.8746928746928747 0.8616400491400491
    {'max_features': 0.5, 'max_depth': 10}
```

```
    0.8763308763308764 0.8627149877149877
    {'max_features': 0.75, 'max_depth': 10}
    0.8761261261261262 0.8614864864864865
    {'max_features': 1.0, 'max_depth': 10}
    0.8761773136773137 0.8614864864864865
    {'max_features': 0.25, 'max_depth': 30}
    0.9780917280917281 0.8547297297297297
    {'max_features': 0.5, 'max_depth': 30}
    0.9797809172809173 0.8541154791154791
    {'max_features': 0.75, 'max_depth': 30}
    0.9807534807534808 0.850583538083538
    {'max_features': 1.0, 'max_depth': 30}
    0.9805487305487306 0.8495085995085995
    {'max_features': 0.25, 'max_depth': 100}
    0.9819819819819819 0.8521191646191646
    {'max_features': 0.5, 'max_depth': 100}
    0.9819819819819819 0.851044226044226
    {'max_features': 0.75, 'max_depth': 100}
    0.9819819819819819 0.8511977886977887
    {'max_features': 1.0, 'max_depth': 100}
    0.9819819819819819 0.8487407862407862
best model parameters: {'max_features': 0.5, 'max_depth': 10}
corresponding validation score: 0.8627149877149877
test score: 0.8624289881774911
randoms state 2
    {'max_features': 0.25, 'max_depth': 1}
    0.7588554463554463 0.7547604422604423
    {'max_features': 0.5, 'max_depth': 1}
    0.7904381654381655 0.788544226044226
    {'max_features': 0.75, 'max_depth': 1}
    0.7588554463554463 0.7547604422604423
    {'max_features': 1.0, 'max_depth': 1}
    0.7588554463554463 0.7547604422604423
    {'max_features': 0.25, 'max_depth': 3}
    0.8409602784602784 0.836916461916462
    {'max_features': 0.5, 'max_depth': 3}
    0.8458742833742834 0.8398341523341524
    {'max_features': 0.75, 'max_depth': 3}
    0.8447481572481572 0.839527027027027
    {'max_features': 1.0, 'max_depth': 3}
    0.8448505323505323 0.8396805896805897
    {'max_features': 0.25, 'max_depth': 10}
    0.8752047502047502 0.8567260442260443
    {'max_features': 0.5, 'max_depth': 10}
    0.8781224406224406 0.8602579852579852
    {'max_features': 0.75, 'max_depth': 10}
    0.8779176904176904 0.8616400491400491
    {'max_features': 1.0, 'max_depth': 10}
```

```
    0.8778153153153153 0.859490171990172
    {'max_features': 0.25, 'max_depth': 30}
    0.9798321048321048 0.8485872235872236
    {'max_features': 0.5, 'max_depth': 30}
    0.9816748566748567 0.8508906633906634
    {'max_features': 0.75, 'max_depth': 30}
    0.9817772317772318 0.8498157248157249
    {'max_features': 1.0, 'max_depth': 30}
    0.9816236691236692 0.8487407862407862
    {'max_features': 0.25, 'max_depth': 100}
    0.9830569205569205 0.8482800982800983
    {'max_features': 0.5, 'max_depth': 100}
    0.9830569205569205 0.8468980343980343
    {'max_features': 0.75, 'max_depth': 100}
    0.9830569205569205 0.847512285012285
    {'max_features': 1.0, 'max_depth': 100}
    0.983005733005733 0.8459766584766585
best model parameters: {'max_features': 0.75, 'max_depth': 10}
corresponding validation score: 0.8616400491400491
test score: 0.8615077537233226
randoms state 3
    {'max_features': 0.25, 'max_depth': 1}
    0.7600839475839476 0.7530712530712531
    {'max_features': 0.5, 'max_depth': 1}
    0.7705773955773956 0.7627457002457002
    {'max_features': 0.75, 'max_depth': 1}
    0.7600839475839476 0.7530712530712531
    {'max_features': 1.0, 'max_depth': 1}
    0.7600839475839476 0.7530712530712531
    {'max_features': 0.25, 'max_depth': 3}
    0.8442362817362817 0.8353808353808354
    {'max_features': 0.5, 'max_depth': 3}
    0.846027846027846 0.8379914004914005
    {'max_features': 0.75, 'max_depth': 3}
    0.8456183456183456 0.8372235872235873
    {'max_features': 1.0, 'max_depth': 3}
    0.8456183456183456 0.8372235872235873
    {'max_features': 0.25, 'max_depth': 10}
    0.8738738738738738 0.856418918918919
    {'max_features': 0.5, 'max_depth': 10}
    0.8778153153153153 0.8593366093366094
    {'max_features': 0.75, 'max_depth': 10}
    0.8767403767403767 0.859029484029484
    {'max_features': 1.0, 'max_depth': 10}
    0.8759213759213759 0.8588759213759214
    {'max_features': 0.25, 'max_depth': 30}
    0.9781941031941032 0.8556511056511057
    {'max_features': 0.5, 'max_depth': 30}
```

```
    0.9801392301392301 0.8541154791154791
    {'max_features': 0.75, 'max_depth': 30}
    0.9804463554463555 0.8539619164619164
    {'max_features': 1.0, 'max_depth': 30}
    0.9805999180999181 0.8507371007371007
    {'max_features': 0.25, 'max_depth': 100}
    0.9813677313677314 0.8542690417690417
    {'max_features': 0.5, 'max_depth': 100}
    0.9813677313677314 0.8516584766584766
    {'max_features': 0.75, 'max_depth': 100}
    0.9813165438165438 0.8507371007371007
    {'max_features': 1.0, 'max_depth': 100}
    0.9813677313677314 0.8482800982800983
best model parameters: {'max_features': 0.5, 'max_depth': 10}
corresponding validation score: 0.8593366093366094
test score: 0.8635037617073545
randoms state 4
    {'max_features': 0.25, 'max_depth': 1}
    0.7657145782145782 0.754914004914005
    {'max_features': 0.5, 'max_depth': 1}
    0.7657145782145782 0.754914004914005
    {'max_features': 0.75, 'max_depth': 1}
    0.7657145782145782 0.754914004914005
    {'max_features': 1.0, 'max_depth': 1}
    0.7657145782145782 0.754914004914005
    {'max_features': 0.25, 'max_depth': 3}
    0.8441850941850941 0.8347665847665847
    {'max_features': 0.5, 'max_depth': 3}
    0.8479217854217854 0.8356879606879607
    {'max_features': 0.75, 'max_depth': 3}
    0.846488533988534 0.8361486486486487
    {'max_features': 1.0, 'max_depth': 3}
    0.846488533988534 0.836455773955774
    {'max_features': 0.25, 'max_depth': 10}
    0.877968877968878 0.859490171990172
    {'max_features': 0.5, 'max_depth': 10}
    0.8811936936936937 0.8584152334152334
    {'max_features': 0.75, 'max_depth': 10}
    0.8822686322686323 0.8591830466830467
    {'max_features': 1.0, 'max_depth': 10}
    0.883087633087633 0.8582616707616708
    {'max_features': 0.25, 'max_depth': 30}
    0.9804463554463555 0.8531941031941032
    {'max_features': 0.5, 'max_depth': 30}
    0.9817260442260443 0.8495085995085995
    {'max_features': 0.75, 'max_depth': 30}
    0.9822891072891073 0.8499692874692875
    {'max_features': 1.0, 'max_depth': 30}
```

```
    0.9823402948402948 0.847051597051597
    {'max_features': 0.25, 'max_depth': 100}
    0.9829545454545454 0.8484336609336609
    {'max_features': 0.5, 'max_depth': 100}
    0.9829545454545454 0.8476658476658476
    {'max_features': 0.75, 'max_depth': 100}
    0.9829545454545454 0.8481265356265356
    {'max_features': 1.0, 'max_depth': 100}
    0.9829545454545454 0.8462837837837838
best model parameters: {'max_features': 0.25, 'max_depth': 10}
corresponding validation score: 0.859490171990172
test score: 0.8582834331337326
randoms state 5
    {'max_features': 0.25, 'max_depth': 1}
    0.756961506961507 0.7590601965601965
    {'max_features': 0.5, 'max_depth': 1}
    0.7872133497133497 0.7926904176904177
    {'max_features': 0.75, 'max_depth': 1}
    0.756961506961507 0.7590601965601965
    {'max_features': 1.0, 'max_depth': 1}
    0.756961506961507 0.7590601965601965
    {'max_features': 0.25, 'max_depth': 3}
    0.833947583947584 0.8381449631449631
    {'max_features': 0.5, 'max_depth': 3}
    0.842495904995905 0.8465909090909091
    {'max_features': 0.75, 'max_depth': 3}
    0.8420864045864046 0.8467444717444718
    {'max_features': 1.0, 'max_depth': 3}
    0.8420864045864046 0.8468980343980343
    {'max_features': 0.25, 'max_depth': 10}
    0.8734643734643734 0.8617936117936118
    {'max_features': 0.5, 'max_depth': 10}
    0.8764332514332515 0.8608722358722358
    {'max_features': 0.75, 'max_depth': 10}
    0.8766380016380017 0.860411547911548
    {'max_features': 1.0, 'max_depth': 10}
    0.8754095004095004 0.85995085995086
    {'max_features': 0.25, 'max_depth': 30}
    0.9787571662571662 0.8548832923832924
    {'max_features': 0.5, 'max_depth': 30}
    0.980497542997543 0.8527334152334153
    {'max_features': 0.75, 'max_depth': 30}
    0.9809070434070434 0.8495085995085995
    {'max_features': 1.0, 'max_depth': 30}
    0.9808046683046683 0.8482800982800983
    {'max_features': 0.25, 'max_depth': 100}
    0.9816748566748567 0.8487407862407862
    {'max_features': 0.5, 'max_depth': 100}
```

```
    0.9816748566748567 0.8502764127764127
    {'max_features': 0.75, 'max_depth': 100}
    0.9816748566748567 0.8490479115479116
    {'max_features': 1.0, 'max_depth': 100}
    0.9816236691236692 0.847972972972973
best model parameters: {'max_features': 0.25, 'max_depth': 10}
corresponding validation score: 0.8617936117936118
test score: 0.8641179180101336
```

1.5 Things to look out for

- are the ranges of the hyperparameters wide enough?
  - if you are unsure, save the training scores and plot the train and val scores!
  - do you see underfitting? model performs poorly on both training and validation sets?
  - do you see overfitting? model performs very good on training but worse on validation?
  - if you don't see both, expand the range of the parameters and you'll likely find a better model
  - read the manual and make sure you understand what the hyperparameter does in the model
    * some parameters (like regularization parameters) should be evenly spaced in log because there is no upper bound
    * some parameters (like max_features) should be linearly spaced because they have clear lower and upper bounds
  - if the best hyperparameter is at the edge of your range, you definitely need to expand the range if you can
- not every hyperparameter is equally important
  - some parameters have little to no impact on train and validation scores
  - in the example above, max_depth is much more important than max_features
  - visualize the results if in doubt
- is the best validation score similar to the test score?
  - it's usual that the validation score is a bit better than the test score
  - but if the difference between the two scores is significant over multiple random states, something could be off
- traiv/val/test split is usually a safe bet for any splitting strategy

1.6 Quiz

1.7 Hyperparameter tuning with folds

- the steps are a bit different

[3]:
```python
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline

df = pd.read_csv('data/adult_data.csv')

# let's separate the feature matrix X, and target variable y
```

```
y = df['gross-income'] # remember, we want to predict who earns more than 50k␣
↪or less than 50k
X = df.loc[:, df.columns != 'gross-income'] # all other columns are features

ordinal_ftrs = ['education']
ordinal_cats = [[' Preschool',' 1st-4th',' 5th-6th',' 7th-8th',' 9th',' 10th','␣
↪11th',' 12th',' HS-grad',\
                ' Some-college',' Assoc-voc',' Assoc-acdm',' Bachelors','␣
↪Masters',' Prof-school',' Doctorate']]
onehot_ftrs =␣
↪['workclass','marital-status','occupation','relationship','race','sex','native-country']
minmax_ftrs = ['age','hours-per-week']
std_ftrs = ['capital-gain','capital-loss']

# collect all the encoders
preprocessor = ColumnTransformer(
    transformers=[
        ('ord', OrdinalEncoder(categories = ordinal_cats), ordinal_ftrs),
        ('onehot', OneHotEncoder(sparse=False,handle_unknown='ignore'),␣
↪onehot_ftrs),
        ('minmax', MinMaxScaler(), minmax_ftrs),
        ('std', StandardScaler(), std_ftrs)])

# all the same up to this point
```

[4]:
```
# we will use GridSearchCV and the parameter names need to contain the ML␣
↪algorithm you want to use
# the parameters of some ML algorithms have the same name and this is how we␣
↪avoid confusion
param_grid = {
              'randomforestclassifier__max_depth': [1, 3, 10, 30, 100], # the␣
↪max_depth should be smaller or equal than the number of features roughly
              'randomforestclassifier__max_features': [0.5,0.75,1.0] # linearly␣
↪spaced between 0.5 and 1
              }

nr_states = 3
test_scores = np.zeros(nr_states)
final_models = []

for i in range(nr_states):
    # first split to separate out the test set
    # we will use kfold on other
    X_other, X_test, y_other, y_test = train_test_split(X,y,test_size = 0.
↪2,random_state=42*i)
```

```
    # splitter for other
    kf = KFold(n_splits=4,shuffle=True,random_state=42*i)

    # the classifier
    clf = RandomForestClassifier(random_state = 42*i) # initialize the␣
↪classifier

    # let's put together a pipeline
    # the pipeline will fit_transform the training set (3 folds), and transform␣
↪the last fold used as validation
    # then it will train the ML algorithm on the training set and evaluate it␣
↪on the validation set
    # it repeats this step automatically such that each fold will be an␣
↪evaluation set once
    pipe = make_pipeline(preprocessor,clf)

    # use GridSearchCV
    # GridSearchCV loops through all parameter combinations and collects the␣
↪results
    grid = GridSearchCV(pipe, param_grid=param_grid,scoring = 'accuracy',
                        cv=kf, return_train_score = True, n_jobs=-1,␣
↪verbose=True)

    # this line actually fits the model on other
    grid.fit(X_other, y_other)
    # save results into a data frame. feel free to print it and inspect it
    results = pd.DataFrame(grid.cv_results_)
    #print(results)

    print('best model parameters:',grid.best_params_)
    print('validation score:',grid.best_score_) # this is the mean validation␣
↪score over all iterations
    # save the model
    final_models.append(grid)
    # calculate and save the test score
    y_test_pred = final_models[-1].predict(X_test)
    test_scores[i] = accuracy_score(y_test,y_test_pred)
    print('test score:',test_scores[i])
```

```
Fitting 4 folds for each of 15 candidates, totalling 60 fits
best model parameters: {'randomforestclassifier__max_depth': 10,
'randomforestclassifier__max_features': 0.75}
validation score: 0.8628685503685503
test score: 0.8576692768309535
Fitting 4 folds for each of 15 candidates, totalling 60 fits
best model parameters: {'randomforestclassifier__max_depth': 10,
'randomforestclassifier__max_features': 0.75}
```

```
validation score: 0.8601428132678133
test score: 0.865806847842776
Fitting 4 folds for each of 15 candidates, totalling 60 fits
best model parameters: {'randomforestclassifier__max_depth': 10,
'randomforestclassifier__max_features': 0.5}
validation score: 0.8624846437346437
test score: 0.8590511285122063
```

[5]:
```
results
```

[5]:
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0        2.181422      0.038313         0.205140        0.020082
1        2.758087      0.027262         0.151116        0.013382
2        3.350122      0.079547         0.196142        0.014781
3        4.463053      0.027966         0.217251        0.015049
4        6.143098      0.089503         0.237941        0.015617
5        7.667399      0.059537         0.226158        0.010623
6        9.119380      0.099135         0.228989        0.010201
7       12.822704      0.115817         0.214337        0.013214
8       16.088822      0.071157         0.234474        0.013608
9       11.278563      0.125260         0.248746        0.019503
10      15.842812      0.263227         0.242607        0.019124
11      20.856934      0.608109         0.335563        0.025797
12      11.742924      0.217466         0.289400        0.018626
13      16.258868      0.193122         0.244635        0.020889
14      16.597886      0.199887         0.145838        0.015063

   param_randomforestclassifier__max_depth  \
0                                        1
1                                        1
2                                        1
3                                        3
4                                        3
5                                        3
6                                       10
7                                       10
8                                       10
9                                       30
10                                      30
11                                      30
12                                     100
13                                     100
14                                     100

   param_randomforestclassifier__max_features  \
0                                         0.5
1                                        0.75
```

```
2                                              1.0
3                                              0.5
4                                             0.75
5                                              1.0
6                                              0.5
7                                             0.75
8                                              1.0
9                                              0.5
10                                            0.75
11                                             1.0
12                                             0.5
13                                            0.75
14                                             1.0

                                               params  split0_test_score  \
0   {'randomforestclassifier__max_depth': 1, 'rand…           0.772881
1   {'randomforestclassifier__max_depth': 1, 'rand…           0.754300
2   {'randomforestclassifier__max_depth': 1, 'rand…           0.754300
3   {'randomforestclassifier__max_depth': 3, 'rand…           0.838299
4   {'randomforestclassifier__max_depth': 3, 'rand…           0.837684
5   {'randomforestclassifier__max_depth': 3, 'rand…           0.837684
6   {'randomforestclassifier__max_depth': 10, 'ran…           0.857647
7   {'randomforestclassifier__max_depth': 10, 'ran…           0.857801
8   {'randomforestclassifier__max_depth': 10, 'ran…           0.857033
9   {'randomforestclassifier__max_depth': 30, 'ran…           0.850276
10  {'randomforestclassifier__max_depth': 30, 'ran…           0.849509
11  {'randomforestclassifier__max_depth': 30, 'ran…           0.847819
12  {'randomforestclassifier__max_depth': 100, 'ra…           0.848741
13  {'randomforestclassifier__max_depth': 100, 'ra…           0.847973
14  {'randomforestclassifier__max_depth': 100, 'ra…           0.845516

    split1_test_score  split2_test_score  split3_test_score  mean_test_score  \
0            0.787469           0.763514           0.792690         0.779139
1            0.754607           0.763514           0.793305         0.766431
2            0.754607           0.763514           0.764281         0.759175
3            0.843366           0.847973           0.851505         0.845286
4            0.841984           0.847205           0.850891         0.844441
5            0.842138           0.847359           0.851044         0.844556
6            0.859644           0.865479           0.867168         0.862485
7            0.858876           0.864711           0.868090         0.862369
8            0.858569           0.863329           0.868704         0.861909
9            0.852273           0.848434           0.863329         0.853578
10           0.849662           0.849048           0.857955         0.851543
11           0.850737           0.846437           0.857801         0.850699
12           0.850430           0.847666           0.860104         0.851735
13           0.847973           0.843827           0.856265         0.849010
14           0.848741           0.844441           0.856419         0.848779
```

	std_test_score	rank_test_score	split0_train_score	split1_train_score	\
0	0.011580	13	0.780405	0.793561	
1	0.015951	14	0.760801	0.760698	
2	0.004731	15	0.760801	0.760698	
3	0.004960	10	0.847871	0.846233	
4	0.005023	12	0.846847	0.845311	
5	0.005075	11	0.846847	0.845362	
6	0.003949	1	0.880170	0.880989	
7	0.004221	2	0.880477	0.881347	
8	0.004558	3	0.879863	0.881245	
9	0.005791	4	0.980446	0.979986	
10	0.003708	6	0.980907	0.980242	
11	0.004384	7	0.981112	0.980293	
12	0.004931	5	0.982136	0.981061	
13	0.004518	8	0.982136	0.981061	
14	0.004686	9	0.982084	0.980958	

	split2_train_score	split3_train_score	mean_train_score	std_train_score
0	0.757729	0.786701	0.779599	0.013456
1	0.757729	0.787469	0.766674	0.012069
2	0.757729	0.757473	0.759175	0.001577
3	0.844390	0.843776	0.845567	0.001608
4	0.843622	0.842496	0.844569	0.001653
5	0.843622	0.842394	0.844556	0.001692
6	0.878225	0.875819	0.878801	0.001993
7	0.879453	0.876024	0.879325	0.002021
8	0.878788	0.874898	0.878698	0.002361
9	0.980498	0.980242	0.980293	0.000202
10	0.980805	0.980395	0.980587	0.000277
11	0.980753	0.980446	0.980651	0.000313
12	0.981828	0.981214	0.981560	0.000439
13	0.981828	0.981214	0.981560	0.000439
14	0.981828	0.981214	0.981521	0.000454

1.8 Things to look out for

- less code but more stuff is going on in the background hidden from you
  - looping over multiple folds
  - .fit_transform and .transform is hidden from you
- nevertheless, GridSearchCV and pipelines are pretty powerful
- working with folds is a bit more robust because the best hyperparameter is selected based on the average score of multiple trained models

1.9 Quiz

Can we use GridSearchCV with sets prepared by train_test_split in advance? Use the sklearn manual or stackoverflow to answer the question.

1.10 Mud card

```
[ ]:
```