Mud card answers

- **how do we know if we should gather more data for SVM? I believe you mentioned something about this, but I didn't understand†!**
  - it's not just for SVMs but generally for any ML model
  - you create a learning curve
  - if the curve is steep, it makes sense to collect more data if you can
  - if the learning curve is saturated, more data will likely not improve model performance
- **What does RBF kernel look like in high dimension**
- **bit confused about how gaussians are 2d in svm classification!**
  - it's still a gaussian but in higher dimensions :)
  - it's called the multivariate normal distribution
- **What is the danger of overfitting with trees? Even if the decision boundary is some complicated fractal-esc line it seems like you would have similar accuracy to a less complex decision boundary.**
  - not really, a model that overfits performs very well on the training set but poorly on the validation set
  - that's still true for trees
  - the complex fractal-esc line occurs because the model is trying to fit each individual point so it wiggles around each training sample
  - such a model will not do well on validation samples that are at different locations
- **I still don't understand the max_depth in the RandomForest algorithms.¬† If the max_depth is 3, does it mean we have 3 turning points in the graph you plotted in the lecture notes?**
  - nope, max 8 but it can be less
  - max depth of 1 means 2 turning points
  - max depth of 2 means 4 turning points
  - max depth of n means 2^n turning points
  - as we discussed in class, trees cannot be arbitrarily deep so the 2^n is just an upper limit, the actual number of turning points can be less but it cannot be more
- **For max_depth,...these parameters, we just try some values and pick the best one?**
  - well yes
  - but you need to be mindful what values you try
- **"With sklearn random forests,¬† we tune n_estimators after other best hyperparameters are found.**
  - basically yes
- **is it possible that the best hyperparameters change when the n_estimator increase? then the model would not be the best one"
  - in my experience, that doesn't happen
  - but write code and verify it on you project dataset
- **Still don't quite understand how the Gaussian kernel comes into play for the SVR and SVC: I know the points are "smeared" with the density of the kernel, but I don't see how that drastically changes the behavior of the SVM.**

        - check out this page and the examples
- **How many hyperparameter values do you recommend looping through when working with larger models?**
    - I assume you mean larger datasets
    - I know you expect a number like 20 but there is no such number
    - as many as your computing resources allow
    - if you need more guidance, come to my office hours
- **what happens if we have a high gamma in SVC? In the class note, you plotted 4 graphs of different gamma in SVC, for gamma =1e4, there is no "sorted line" in the graph like other 3, where is the "sorted line"?**
    - I don't know what you mean by 'sorted line'
    - can you come to the office hours or post on ed discussion?
- **while tuning models in real data science projects, is it correct that what we need to do is just make a list of alpha and plot all accuracy scores and pick the best one?**
    - you might have more than one hyperparameters to tune
    - you might not use accuracy but a different evaluation metric
    - you need to choose the best model based on the validation scores not just any scores
- **When creating a learning curve, would we change the size of our training set multiple times and calculate the eval metric for each to fill in the points on the x-axis for sample sizes up to the size of our dataset?**
    - yes
    - and ideally you would calculate the eval metric multiple times for each size value and you'd plot the mean and std of the scores

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)

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

In [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,
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
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','
                ' Some-college',' Assoc-voc',' Assoc-acdm',' Bachelors',' Master
onehot_ftrs = ['workclass','marital-status','occupation','relationship','race','
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_f
        ('minmax', MinMaxScaler(), minmax_ftrs),
        ('std', StandardScaler(), std_ftrs)])

prep = Pipeline(steps=[('preprocessor', preprocessor)]) # for now we only prepro
```

Quiz

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

Basic hyperparameter tuning

In [2]:

```
# let's train a random forest classifier

# decide which parameters to tune and what values to try
# all parameters not specified here will be the default
param_grid = {
              'max_depth': [1, 3, 10, 30, 100], # the max_depth should be smalle
              'max_features': [0.5,0.75,1.0] # linearly spaced between 0.5 and 1
              }

# we will loop through nr_states random states so we will return nr_states test
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,r

    # 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 =

    # 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)

    # we save the train and validation scores
    # the validation scores are necessary to select the best model
    # we save the train score just to check things
    train_score = np.zeros(len(ParameterGrid(param_grid)))
    val_score = np.zeros(len(ParameterGrid(param_grid)))
    models = []

    # 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) # i
        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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.75, 'max_depth': 10}
corresponding validation score: 0.8591830466830467
test score: 0.8601259020420697
randoms state 5
    {'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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, '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': 0.5, 'max_depth': 10}
corresponding validation score: 0.8608722358722358
test score: 0.8648856133886074
```

Things to look out for

- are the ranges of the hyperparameters wide enough?
  - 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 linearly spaced in log [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2]
    - some parameters (like max_features) should be linearly spaced
- 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 larger than the test score
  - but if the difference between the two scores is significant over multiple random states, something is off
- traiv/val/test split is usually a safe bet for any splitting strategy

Quiz

Hyperparameter tuning with folds

- the steps are a bit different

```
In [3]: 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
        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','
                         ' Some-college',' Assoc-voc',' Assoc-acdm',' Bachelors',' Master
        onehot_ftrs = ['workclass','marital-status','occupation','relationship','race','
        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_f
                ('minmax', MinMaxScaler(), minmax_ftrs),
                ('std', StandardScaler(), std_ftrs)])

        # all the same up to this point

        # we will use GridSearchCV and the parameter names need to contain the ML algori
        # the parameters of some ML algorithms have the same name and this is how we avo
        param_grid = {
                      'randomforestclassifier__max_depth': [1, 3, 10, 30, 100], # the ma
                      'randomforestclassifier__max_features': [0.5,0.75,1.0] # linearly
                      }

        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,rand

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

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

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

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

            # 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 sc
    # 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, 'randomforestcl
assifier__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, 'randomforestcl
assifier__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, 'randomforestcl
assifier__max_features': 0.5}
validation score: 0.8624846437346437
test score: 0.8590511285122063
```

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

Quiz

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

Mud card

```
In [ ]:
```