EVML3

SUPERVISED ML

HANDS-ON

JEROEN VEEN

HAN_UNIVERSITY OF APPLIED SCIENCES

CONTENTS

- Sci-kit learn interface
- Training a classifier
- Thinking about performance
- Hyperparameter tuning
- Nonlinear SVM
- Visualizing a Decision Tree
- Linear regression exercise
- Polynomial regression exercise

SCIKIT LEARN INTERFACE DESIGN

- Estimators
  - estimation performed by the fit() method
  - dataset as a parameter (or two for supervised learning)
  - Any other parameter is considered a hyperparameter
- Transformers
  - performed by the transform() method
  - fit_transform() is equivalent to calling fit() and then transform() (but sometimes fit_transform() is optimized and runs much faster
- Predictors
  - prediction method performed by predict() method
  - quality of the predictions measured by score() method

MORE ON THE INTERFACE

- See: https://youtu.be/84gqSbLcBFE

SCIKIT-LEARN

- a few standard datasets available, e.g.

```
import cv2 as cv
from sklearn import datasets
from sklearn import svm
```

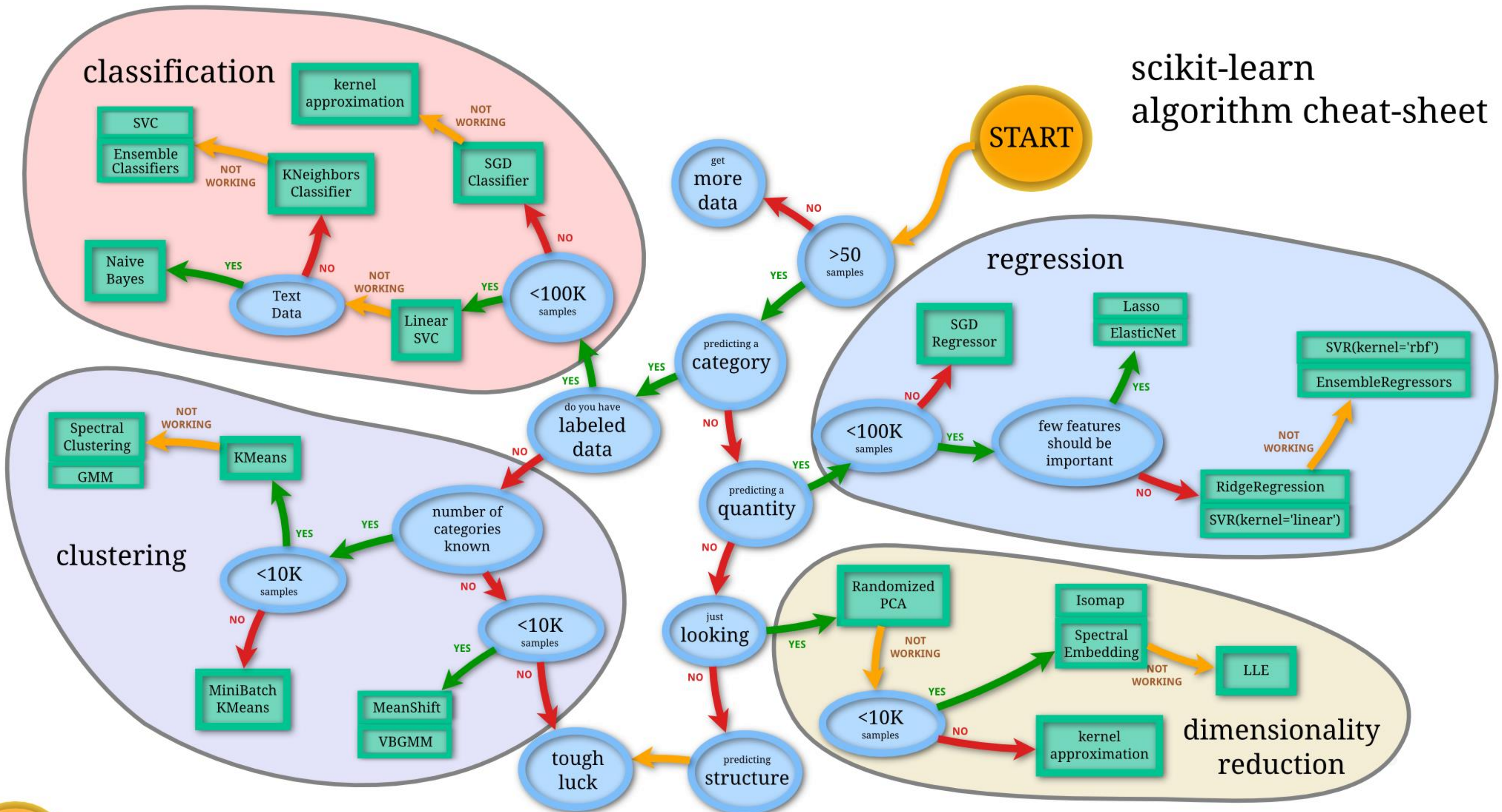
```
digits = datasets.load_digits()
```

- .data member: (n_samples, n_features) sized array of feature
- .target member: (n_samples,) sized array of labels

See supervised_01.py

See: https://scikit-learn.org/stable/tutorial/basic/tutorial.html

ALGORITHMS, ALGORITHMS



TRAINING A CLASSIFIER WITH SCIKIT-LEARN

- Goal: predict which digit (0..9) an image represents

- Load a model and instantiate a classifier

```
from sklearn import svm
```

```
clf = svm.SVC(gamma=0.001, C=100.)
```

- given samples of each of the 10 possible classes on which we **fit** an estimator to be able to **predict** the classes

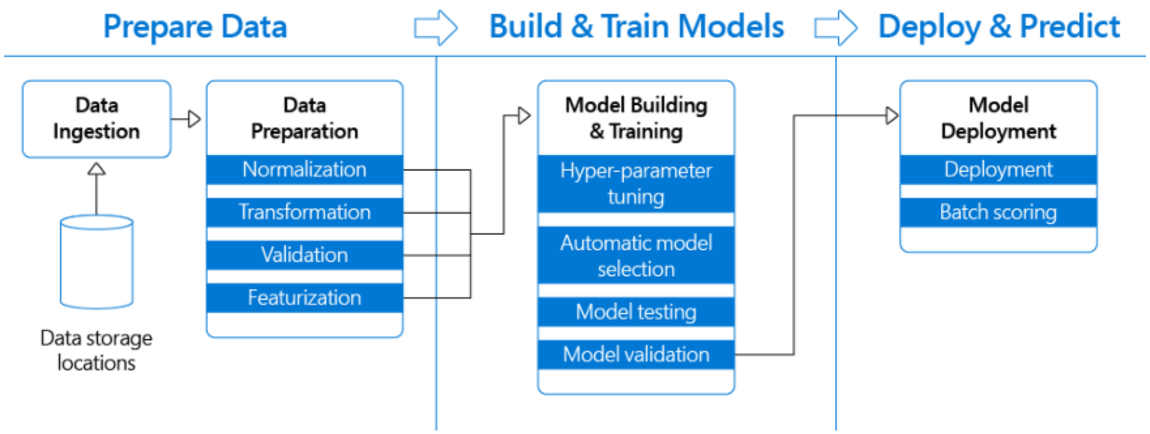
```
# train the model on all images except the last
clf.fit(digits.data[:-1], digits.target[:-1])

## predict using the last image
prediction = clf.predict(digits.data[-1:])
```

See supervised_01.py

PIPELINES

- Sequence of processing components
- Preparation pipeline is the same for training and prediction
  - Cleaner code
  - Fewer bugs



Source: https://www.kaggle.com/alexisbcook/pipelines

PREPARATION PIPELINE EXAMPLE

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

list of name/estimator pairs defining a sequence of steps

calls fit_transform() sequentially on all transformers

Source: Géron, ed2, page 70

BUILD YOUR OWN PIPELINE OBJECTS

```python
class RemoveBackground(BaseEstimator, TransformerMixin):
    """ Segment the image and return a
        BW image with foreground (white) and background (black) """
    def __init__(self):
        pass

    def fit(self, X, y=None):
        # fit is not used, so returns itself
        return self

    def transform(self, X, y=None):
        X_ = []
        for img in X:
            # Change image color space
            img_hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)

            #Mark pixels outside background color range
            result = cv.bitwise_not(cv.inRange(img_hsv, Config.BG_HSV_LOWER_BOUND, Config.BG_HSV_UPPER_BOUND))
            X_.append(result)

        return X_
```

NOW, TRY IT OUT

```
# Bundle preprocessing and modeling code in a pipeline
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('model', model)
                             ])

# Preprocessing of training data, fit model
my_pipeline.fit(X_train, y_train)

# Preprocessing of validation data, get predictions
preds = my_pipeline.predict(X_valid)
```

MEASURING ACCURACY

- How well is your classifier performing?
  Use cross-validation!

```
# compute accuracy over n-folds
X_train, X_test = digits.data[:1000,], digits.data[1000:,]
y_train, y_test = digits.target[:1000,], digits.target[1000:,]
score = cross_val_score(clf, X_train, y_train, cv=3, scoring="accuracy")
print("score: {}".format(score))
```

- split the training set into n distinct subsets called *folds*, then train and evaluate the model n times, picking a different fold for evaluation every time and training on the other folds
- Note that the data is already shuffled, which is good because this guarantees that all cross-validation folds will be similar!

SVC HYPERPARAMETERS

- SVM C-parameter controls trade-off between smooth decision boundary and classifying training points correctly. Large C results in more training datapoints correctly classified, so less smooth boundary
- SVM gamma-parameter

HYPERPARAMETER TUNING BY GRID SEARCH

- Typical hyperparameters include c, kernel, gamma for SVC
- Search the hyper-parameter space for the best score
- Exhaustive of randomized

```
# Split the dataset in two equal parts
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, digits.target, test_size=0.5, random_state=0)

# Set the parameters by cross-validation
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

# Tuning hyper-parameters for accuracy
clf = GridSearchCV(
    SVC(), tuned_parameters, scoring='accuracy'
)
clf.fit(X_train, y_train)
```

See supervised_02.py

EXERCISE

- Try to build another classifier (KNeighborsClassifier) for the digit dataset that achieves over 97% accuracy on the test set.
- you just need to find good hyperparameter values (try a grid search on the weights and n_neighbors hyperparameters).

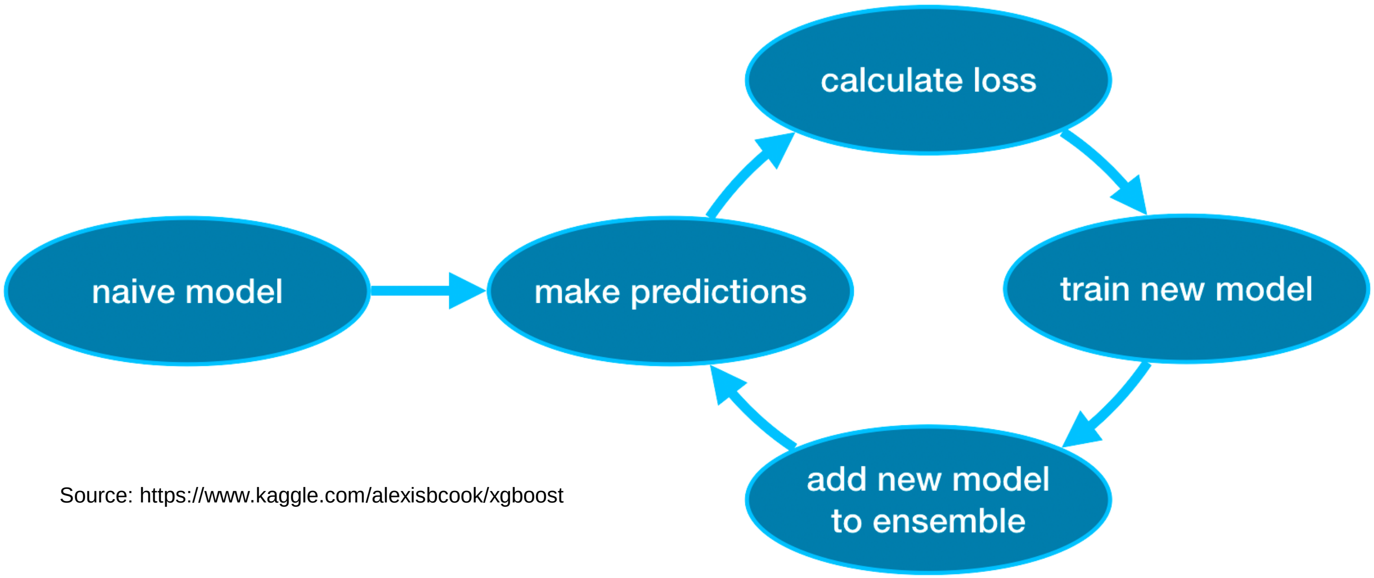
VISUALIZING A DECISION TREE

```
# show tree
plt.figure()
tree.plot_tree(clf, feature_names=iris.feature_names,
               class_names=iris.target_names,
               rounded=True,
               filled=True)
plt.show()
```

See supervised_04.py

GRADIENT BOOSTING

- Applied to ensemble methods, such as random forest



Source: https://www.kaggle.com/alexisbcook/xgboost

- Gradient descent along the loss function to determine the parameters in this new model

XGBOOST

- **extreme gradient boosting**,
- Scikit-learn API for XGBoost (xgboost.XGBRegressor)

- Parameters:
  - Number of estimators, typically 100-1000, depending on learning rate
  - Use early stopping causing the model to stop iterating when the validation score stops improving,
  - Learning rate

https://www.youtube.com/watch?v=OQKQHNCVf5k