UFRN

Open Data Day
Natal, Brazil

01 a 03 de março de 2018

Minicourse

Machine Learning Fundamentals in Python - Hyperparameter Optimization & Cross Validation

Speaker:
Prof. Ivanovitch Silva (ivan@imd.ufrn.br)

Prepare & Cleaning the Data

```
def prepare_cleaning(df):

    # format the date
    stripped_commas = df['price'].str.replace(',', '')
    stripped_dollars = stripped_commas.str.replace('$', '')
    df['price'] = stripped_dollars.astype('float')

    # drop the unless columns
    columns = ["room_type", "city", "state","latitude","longitude","zipcode",
          "host_response_rate","host_acceptance_rate","host_listings_count",
          "cleaning_fee", "security_deposit"]
    df.drop(columns,axis=1,inplace=True)

    # drop null rows
    df.dropna(axis=0,inplace=True)

    return df

dc_listings = prepare_cleaning(dc_listings)
```

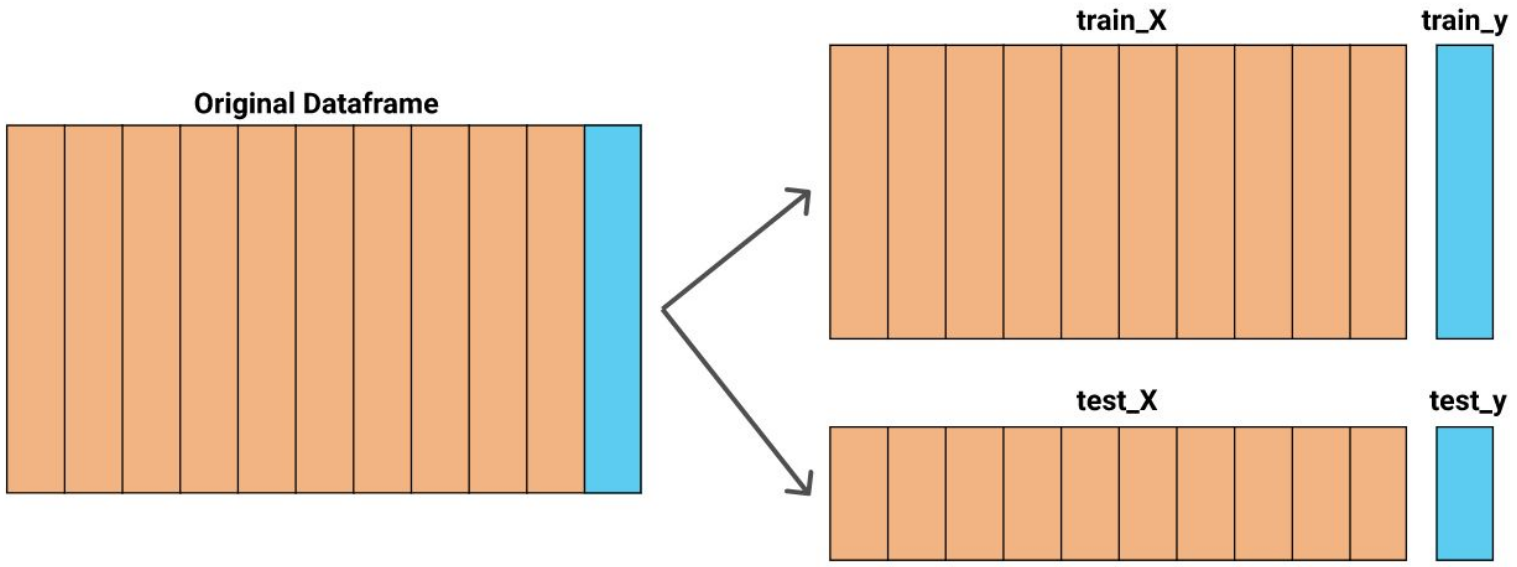
Standardization

	accommodates	bedrooms	bathrooms	beds	price	minimum_nights	maximum_nights	number_of_reviews
0	0.401420	-0.249501	-0.439211	0.297386	160.0	-0.341421	-0.016575	-0.516779
1	1.399466	2.129508	2.969551	1.141704	350.0	-0.065047	-0.016606	1.706767
2	-1.095648	-0.249501	1.265170	-0.546933	50.0	-0.065047	-0.016575	-0.482571
3	-0.596625	-0.249501	-0.439211	-0.546933	95.0	-0.341421	-0.016575	-0.516779
4	0.401420	-0.249501	-0.439211	-0.546933	50.0	1.316824	-0.016575	-0.516779

Standardization

```
from sklearn.preprocessing import StandardScaler

# store the target column
target = dc_listings.price

# scale the features
scaled_features = StandardScaler().fit_transform(dc_listings.values)
scaled_features_df = pd.DataFrame(scaled_features,
                                  index=dc_listings.index,
                                  columns=dc_listings.columns)

# update the original price column
scaled_features_df.price = target

# show the five first rows
scaled_features_df.head()
```

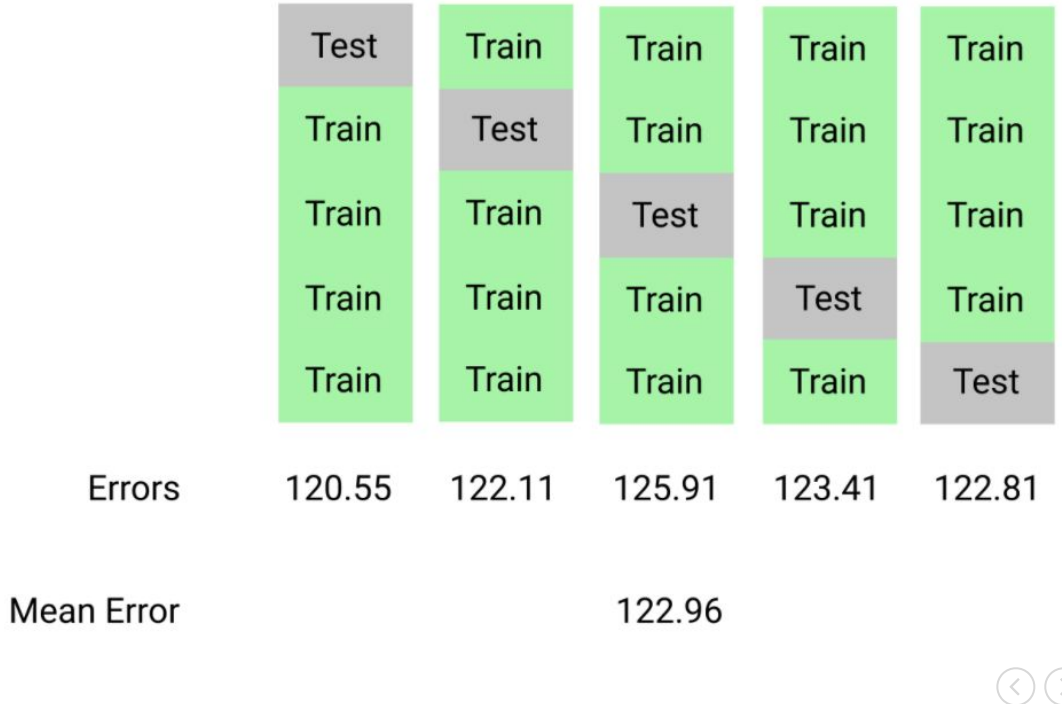
Splitting out training data



Splitting out training data

```
from sklearn.model_selection import train_test_split

features = ["accommodates", "bedrooms", "bathrooms", "beds",
            "minimum_nights", "maximum_nights", "number_of_reviews"]

X = dc_listings[features]
y = dc_listings['price']

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.20,random_state=1)
```

Holdout & Cross Validation



Goal #1

- We'll focus on the impact of increasing k, the number of nearby neighbors the model uses to make predictions.

Hyperparameters

- When we **vary the feature**s that are used in the model, we're **affecting the data** that the model uses.
- On the other hand, **varying the k value affects the behavior of the model independently of the actual data** that's used when making predictions.
- Values that affect the behavior and performance of a model that are unrelated to the data that's used are referred to as hyperparameters.

Hyperparameter Optimization

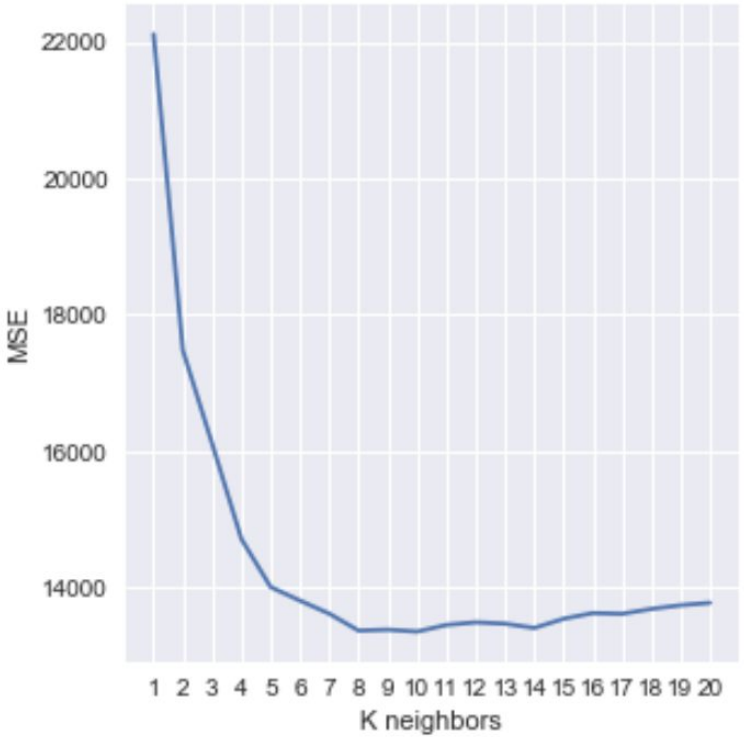
A simple but common hyperparameter optimization technique is known as grid search:

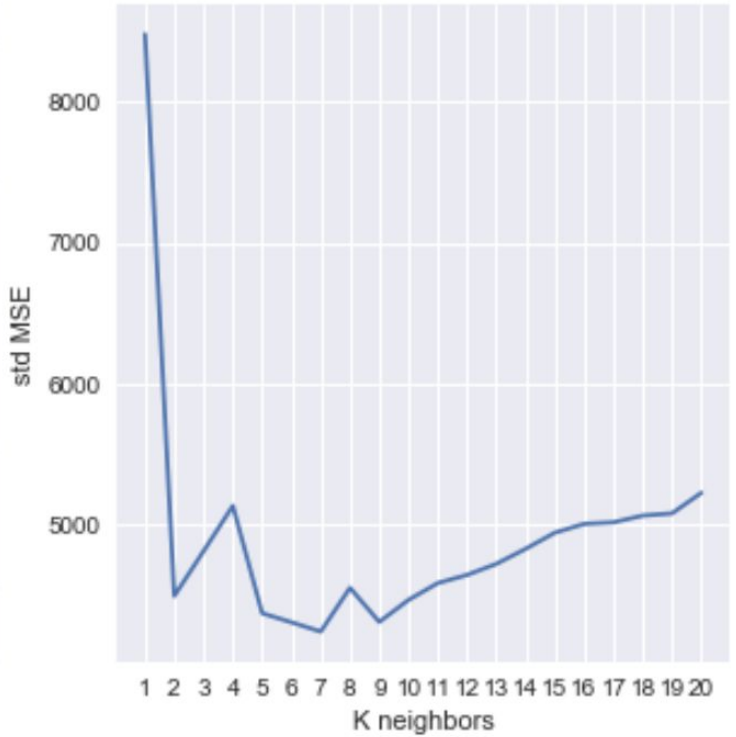
- selecting a subset of the possible hyperparameter values,
- training a model using each of these hyperparameter values,
- evaluating each model's performance,
- selecting the hyperparameter value that resulted in the lowest error value.

GridSearchCV

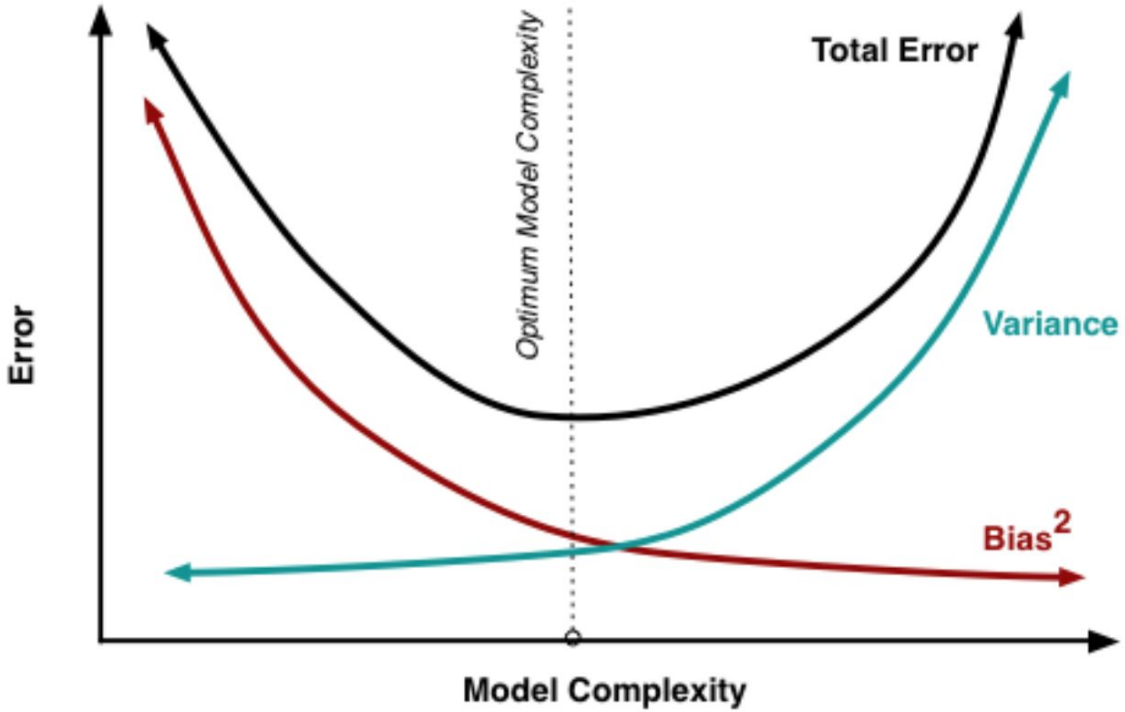

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor

# configure the hyperparameters
hyperparameters = {
    "n_neighbors": range(1,21,1)
}

# instantiate a KNN model
knn = KNeighborsRegressor()

# execute a gridsearch procedure
grid = GridSearchCV(knn,
                    param_grid=hyperparameters,
                    cv=10,
                    scoring="neg_mean_squared_error",
                    return_train_score=True)

# fit using train data
grid.fit(train_X, train_y)

# look the best results
best_params = grid.best_params_
best_score = grid.best_score_
```

Hyperparameter Optimization





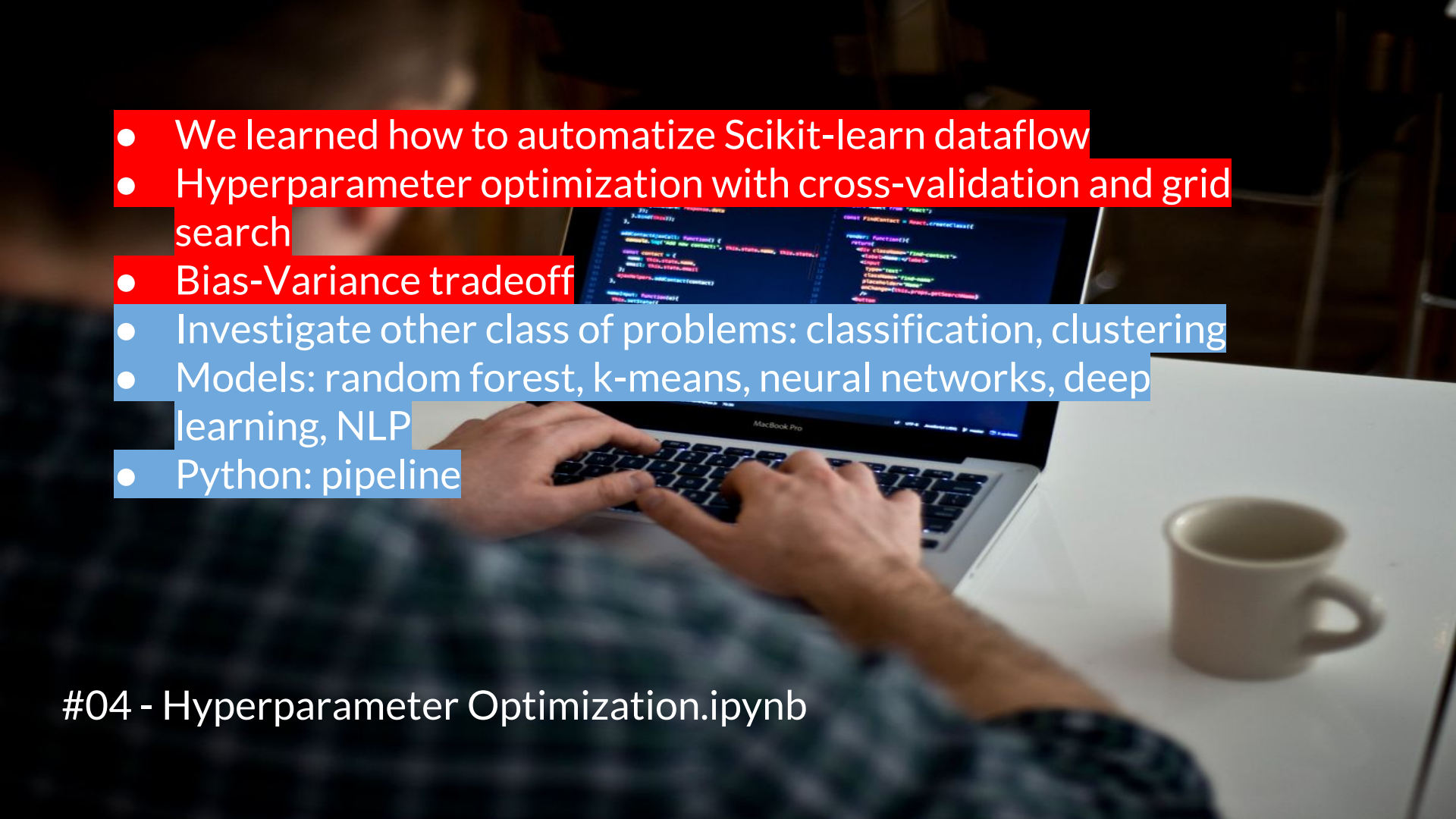
Bias-Variance Tradeoff



Making prediction os unseen data

```
from sklearn.metrics import mean_squared_error

# best model
best_knn = grid.best_estimator_

# predict using the best model with unseen X
predictions = best_knn.predict(test_X)

mse = mean_squared_error(test_y,predictions)

print("mse: {}\nrmse: {}".format(mse,np.sqrt(mse)))
```

```
mse: 10894.830489795919
rmse: 104.37830468922131
```



- We learned how to automatize Scikit-learn dataflow
- Hyperparameter optimization with cross-validation and grid search
- Bias-Variance tradeoff
- Investigate other class of problems: classification, clustering
- Models: random forest, k-means, neural networks, deep learning, NLP
- Python: pipeline

#04 - Hyperparameter Optimization.ipynb