Big Data - Foundations and Applications

Lesson #14 - Machine Learning Fundamentals Cont.

Ivanovitch Silva

November, 2017

Agenda

- Hyperparameter optimization
- Cross validation

Goal #1

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

Collecting data

- We exported both the training (train_df) and test sets (test_df) from the last missions to CSV files, dc_airbnb_train.csv and dc_airbnb_test.csv respectively.

Hyperparameters

- When we **vary the features** 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:

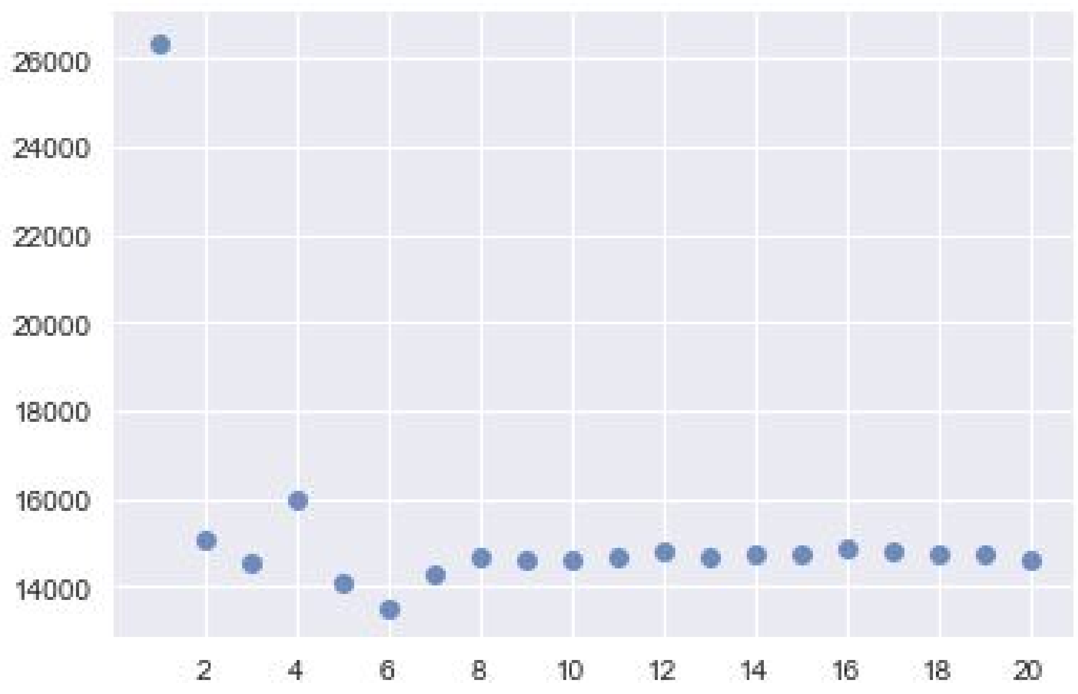
- selecting a subset of the possible hyperparameter values,
- training a model using each of these hyperparameter values,
- evaluating each model's performance,
- selecting the hyperparameter value that resulted in the lowest error value.

Grid Search

- accommodates
- bedrooms
- bathrooms
- number_of_reviews

k	MSE
1	26364.928327645051
2	15100.522468714449
3	14579.597901655923
4	16006.955844709897
5	14114.812468714448

Visualizing hyperparameters values



Varying features and hyperparameters

```
two_features = ['accommodates', 'bathrooms']
three_features = ['accommodates', 'bathrooms', 'bedrooms']
hyper_params = [x for x in range(1,21)]
# Append the first model's MSE values to this list.
two_mse_values = list()
# Append the second model's MSE values to this list.
three_mse_values = list()
two_hyp_mse = dict()
three_hyp_mse = dict()
```

Varying features and hyperparameters

```
for hp in hyper_params:
    knn = KNeighborsRegressor(n_neighbors=hp, algorithm='brute')
    knn.fit(train_df[two_features], train_df['price'])
    predictions = knn.predict(test_df[two_features])
    mse = mean_squared_error(test_df['price'], predictions)
    two_mse_values.append(mse)
```

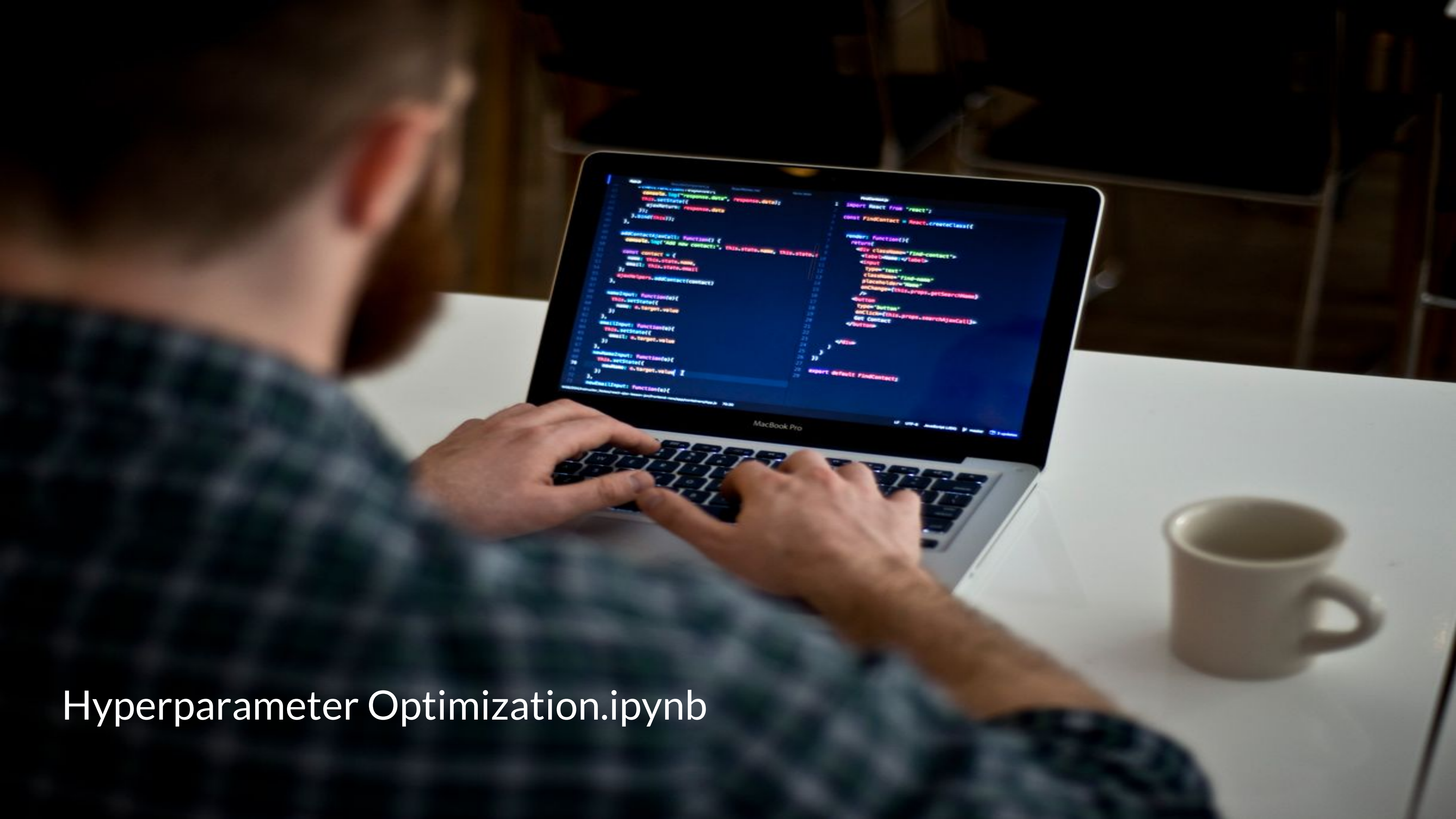
Varying features and hyperparameters

```
two_lowest_mse = two_mse_values[0]
two_lowest_k = 1

for k,mse in enumerate(two_mse_values):
    if mse < two_lowest_mse:
        two_lowest_mse = mse
        two_lowest_k = k + 1
```

Varying features and hyperparameters

```
two_hyp_mse[two_lowest_k] = two_lowest_mse
three_hyp_mse[three_lowest_k] = three_lowest_mse

print(two_hyp_mse)
print(three_hyp_mse)
```

```
{5: 14787.264345847556}
{7: 13518.769009310208}
```

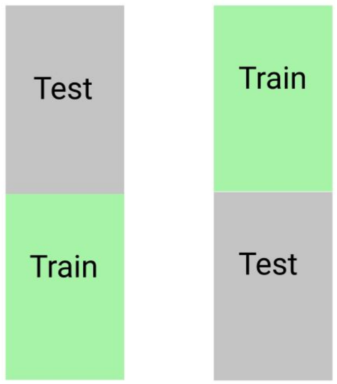


Hyperparameter Optimization.ipynb

Goal #2

- We'll focus on more robust techniques for testing a machine learning model's accuracy

Holdout validation



Error 123.64 123.21

Mean Error 123.43

K-Fold Cross Validation

Holdout validation is actually a specific example of a larger class of validation techniques called **k-fold cross-validation.**

K-Fold Cross Validation

	Test	Train	Train	Train	Train
	Train	Test	Train	Train	Train
	Train	Train	Test	Train	Train
	Train	Train	Train	Test	Train
	Train	Train	Train	Train	Test
Errors	120.55	122.11	125.91	123.41	122.81
Mean Error			122.96		

Function for training models

```
# Use np.mean to calculate the mean.
import numpy as np
fold_ids = [1,2,3,4,5]
def train_and_validate(df, folds):
    fold_rmses = []
    for fold in folds:
        # Train
        model = KNeighborsRegressor()
        train = df[df["fold"] != fold]
        test = df[df["fold"] == fold]
        model.fit(train[["accommodates"]], train["price"])
        # Predict
        labels = model.predict(test[["accommodates"]])
        test["predicted_price"] = labels
        mse = mean_squared_error(test["price"], test["predicted_price"])
        rmse = mse**(1/2)
        fold_rmses.append(rmse)
    return(fold_rmses)

rmses = train_and_validate(dc_listings, fold_ids)
```

K-Fold Cross Validation using Scikit-Learn

```
from sklearn.model_selection import KFold
kf = KFold(n_folds, shuffle=False, random_state=None)
```

- n_folds is the number of folds you want to use,
- shuffle is used to toggle shuffling of the ordering of the observations in the dataset,
- random_state is used to specify the random seed value if shuffle is set to True.

K-Fold Cross Validation using Scikit-Learn

```
from sklearn.model_selection import cross_val_score
cross_val_score(estimator, X, Y, scoring=None, cv=None)
```

K-Fold Cross Validation using Scikit-Learn

Full Example

```
from sklearn.model_selection import cross_val_score, KFold

# kfold instance
kf = KFold(5, shuffle=True, random_state=1)

# knn model
model = KNeighborsRegressor()

# cross validation (knn,x,y,scoring,kfold)
mses = cross_val_score(model, dc_listings[["accommodates"]],
                       dc_listings["price"], scoring="neg_mean_squared_error", cv=kf)
```

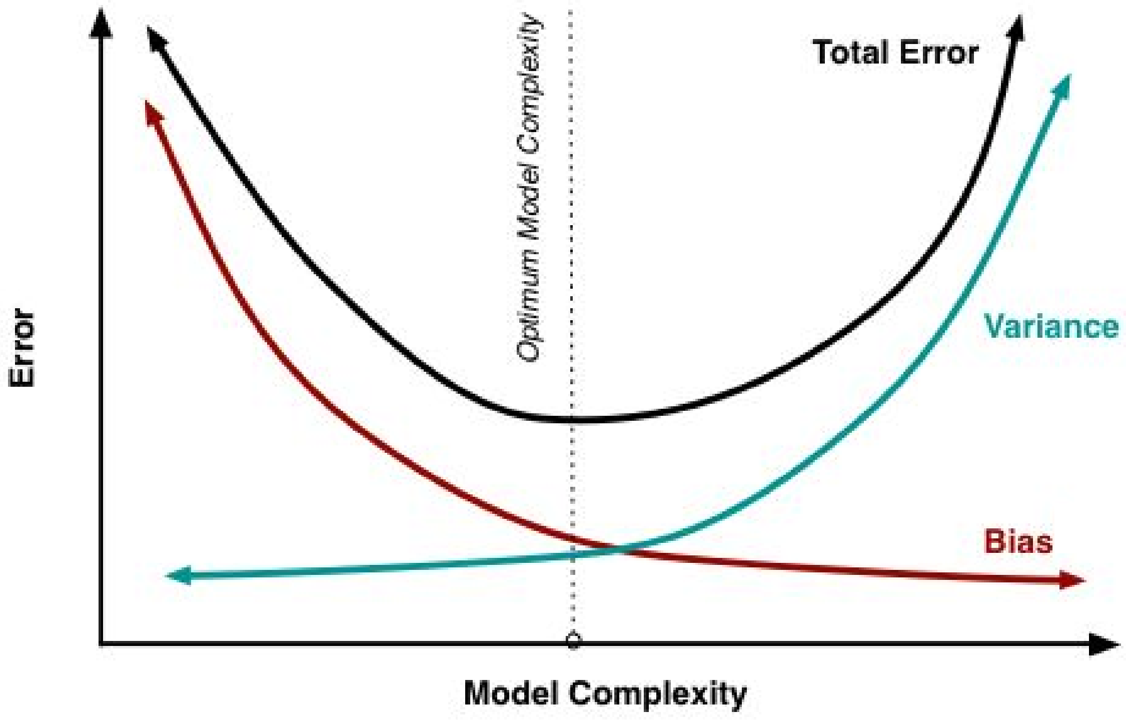
Exploring Different K Values

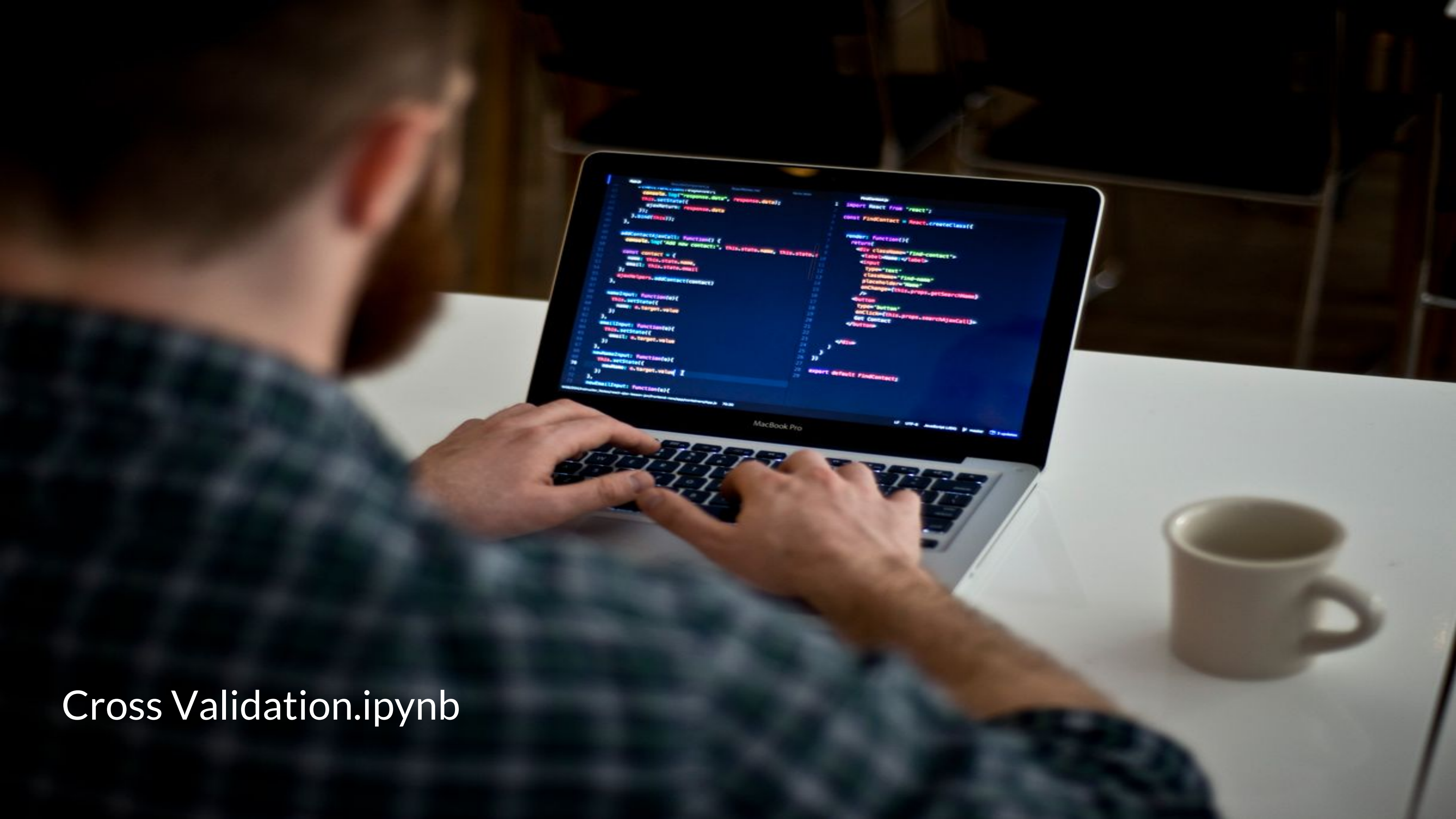
```
from sklearn.model_selection import cross_val_score, KFold
import numpy as np

num_folds = [3, 5, 7, 9, 10, 11, 13, 15, 17, 19, 21, 23]

for fold in num_folds:
    kf = KFold(fold, shuffle=True, random_state=1)
    model = KNeighborsRegressor()
    mses = cross_val_score(model, dc_listings[["accommodates"]],
                           dc_listings["price"], scoring="neg_mean_squared_error", cv=kf)
    rmses = np.sqrt(np.absolute(mses))
    avg_rmse = np.mean(rmses)
    std_rmse = np.std(rmses)
    print(str(fold), "folds: ", "avg RMSE: ", str(avg_rmse), "std RMSE: ", str(std_rmse))
```

Exploring Different K Values

```
3 folds:  avg RMSE:  130.241113525 std RMSE:  33.8224588753
5 folds:  avg RMSE:  142.995566567 std RMSE:  49.511835738
7 folds:  avg RMSE:  138.798092396 std RMSE:  43.9755423505
9 folds:  avg RMSE:  126.785310282 std RMSE:  36.1877009487
10 folds:  avg RMSE:  131.817845321 std RMSE:  34.3853202147
11 folds:  avg RMSE:  128.520346424 std RMSE:  37.1935930162
13 folds:  avg RMSE:  122.197051186 std RMSE:  41.2948951891
15 folds:  avg RMSE:  129.832684903 std RMSE:  43.4306418425
17 folds:  avg RMSE:  131.421694504 std RMSE:  43.0233643077
19 folds:  avg RMSE:  126.726422355 std RMSE:  48.8562172175
21 folds:  avg RMSE:  123.156701843 std RMSE:  49.4378544412
23 folds:  avg RMSE:  122.696083298 std RMSE:  43.145630879
```

Bias-Variance tradeoff




MacBook Pro
Cross Validation.ipynb