Overfitting and Cross validation

Part 1

Cesar Acosta Ph.D.

COMPLEXITY AND OVERFITTING

MODEL COMPLEXITY

- A model becomes more complex if new terms are included (polynomial terms, interactions, nonlinear terms, etc.)
- Adding complexity helps the model to identify new patterns useful for prediction
- We should increase the model complexity but not beyond the point where the model starts overfitting

COMPLEXITY AND OVERFITTING

- UNDERFITTING

  The model has not yet identified all important patterns from the training data set

- OVERFITTING

  The model has started to learn patterns that are specific to the training data but not to new data sets

Example 1
Polynomial Regression

EXAMPLE 1

The *Auto.csv* file contains information about 392 cars

- mpg (miles per gallon)
- cylinders (between 4 and 8 cylinders)
- displacement (engine displacement in cubic inches)
- horsepower (engine horsepower)
- weight (pounds)
- acceleration (number of seconds to accelerate from 0 to 69 mph)
- year (Model year)
- origin (1. American, 2. European, 3. Japanese)
- name (vehicle name)

EXAMPLE 1

The *Auto.csv* file contains the following information about 392 cars

- mpg (miles per gallon)
- cylinders (between 4 and 8 cylinders)
- displacement (engine displacement in cubic inches)
- horsepower (engine horsepower)
- weight (pounds)
- acceleration (number of seconds to accelerate from 0 to 69 mph)
- year (Model year)
- origin (1. American, 2. European, 3. Japanese)
- name (vehicle name)

Use cross validation to find the best polynomial model to predict the car’s mileage (mpg) using predictor horsepower

POLYNOMIAL REGRESSION

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

```
auto = pd.read_csv('Auto.csv')
auto[:5]
```

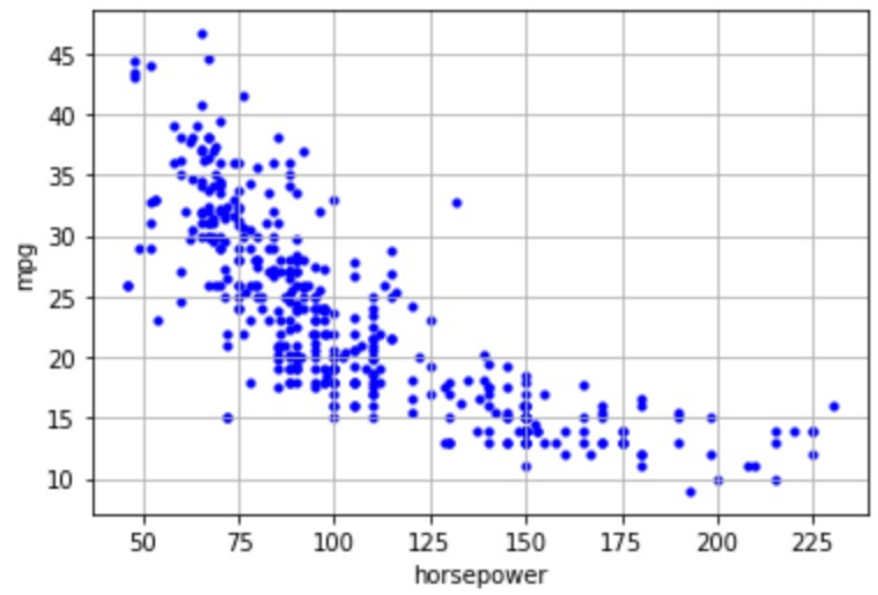
	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

POLYNOMIAL REGRESSION

```
hp = auto.horsepower
mpg = auto.mpg
```

```
plt.scatter(hp,mpg,c='b',s=10)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()
```

What polynomial degree best predicts the data?

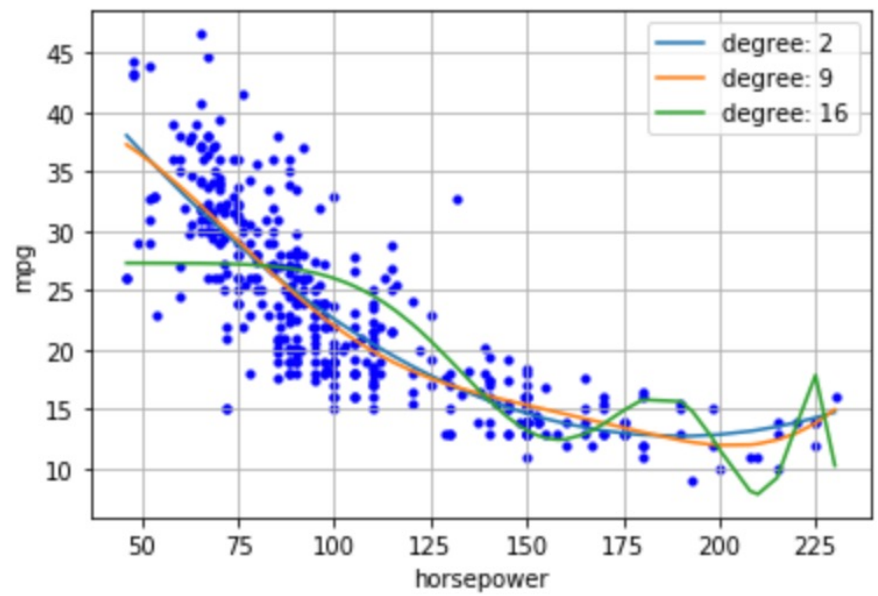


POLYNOMIAL REGRESSION

What polynomial degree best predicts the data?

```
hp = auto.horsepower
mpg = auto.mpg
```

```
plt.scatter(hp,mpg,c='b',s=10)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()
```



POLYNOMIAL REGRESSION – LINEAR MODEL

```
hp = auto.horsepower
mpg = auto.mpg
```

```
# predictors must be an array
# (not vector or series)
hp.shape
```

```
(392,)
```

```
hp1 = hp.values.reshape(-1,1)
hp1.shape
```

```
(392, 1)
```

```
hp1[:5]
```

```
array([[130],
       [165],
       [150],
       [150],
       [140]])
```

POLYNOMIAL REGRESSION – LINEAR MODEL

```
hp = auto.horsepower
mpg = auto.mpg
```

```
# predictors must be an array
# (not vector or series)
hp.shape
```

```
(392,)
```

```
hp1 = hp.values.reshape(-1,1)
hp1.shape
```

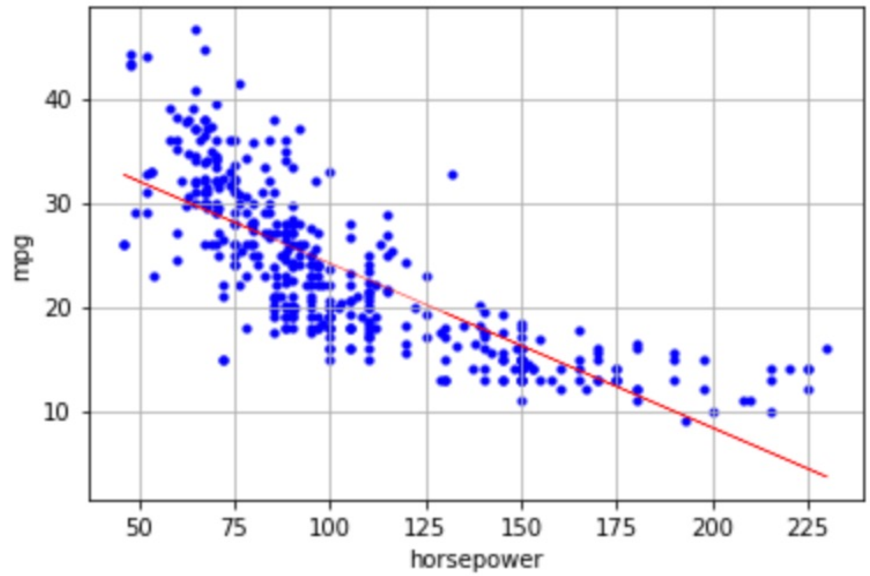
```
(392, 1)
```

```
hp1[:5]
```

```
array([[130],
       [165],
       [150],
       [150],
       [140]])
```

```
model1 = LinearRegression().fit(hp1,mpg)
yhat   = model1.predict(hp1)
```

```
plt.scatter(hp,mpg,c='b',s=10)
plt.plot(hp,yhat,c='r',lw=0.5)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()
```



POLYNOMIAL REGRESSION – LINEAR MODEL

```
hp = auto.horsepower
mpg = auto.mpg
```

```
# predictors must be an array
# (not vector or series)
hp.shape
```

```
(392,)
```

```
hp1 = hp.values.reshape(-1,1)
hp1.shape
```

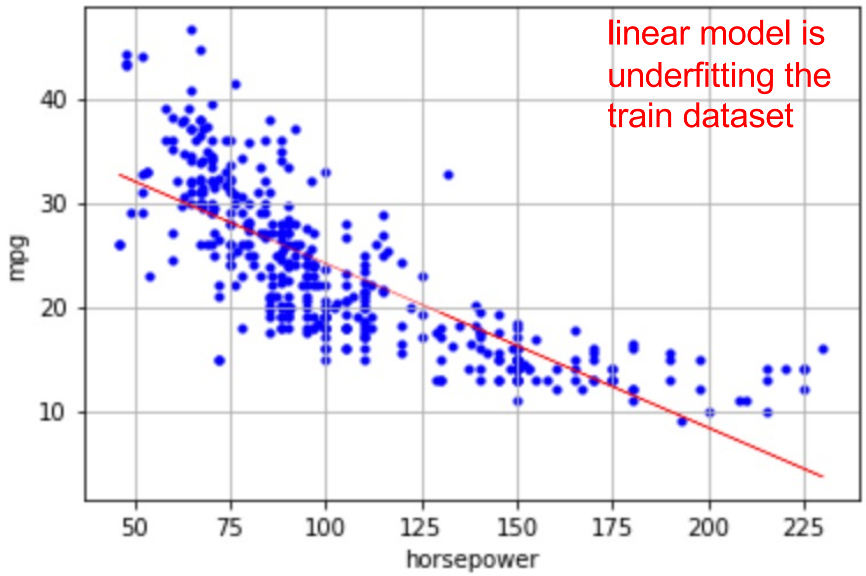
```
(392, 1)
```

```
hp1[:5]
```

```
array([[130],
       [165],
       [150],
       [150],
       [140]])
```

```
model1 = LinearRegression().fit(hp1,mpg)
yhat   = model1.predict(hp1)
```

```
plt.scatter(hp,mpg,c='b',s=10)
plt.plot(hp,yhat,c='r',lw=0.5)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()
```



POLYNOMIAL REGRESSION – QUADRATIC MODEL

```
hp1[:5]
```

```
array([[130],
       [165],
       [150],
       [150],
       [140]])
```

```
poly2 = PolynomialFeatures(degree=2)
hp2 = poly2.fit_transform(hp1)
```

```
# hp2 is array (no need to reshape)
```

POLYNOMIAL REGRESSION – QUADRATIC MODEL

```
hp1[:5]
```

```
array([[130],
       [165],
       [150],
       [150],
       [140]])
```

```
poly2 = PolynomialFeatures(degree=2)
hp2 = poly2.fit_transform(hp1)
hp2[:5]
```

```
array([[1.0000e+00, 1.3000e+02, 1.6900e+04]
       [1.0000e+00, 1.6500e+02, 2.7225e+04]
       [1.0000e+00, 1.5000e+02, 2.2500e+04]
       [1.0000e+00, 1.5000e+02, 2.2500e+04]
       [1.0000e+00, 1.4000e+02, 1.9600e+04]
```

```
# hp2 is array (no need to reshape)
```

fit_transform( ) adds
a column of ones and
a column of squared values
to hp1

POLYNOMIAL REGRESSION – QUADRATIC MODEL

```
hp1[:5]
```

```
array([[130],
       [165],
       [150],
       [150],
       [140]])
```

```
poly2 = PolynomialFeatures(degree=2)
hp2 = poly2.fit_transform(hp1)
hp2[:5]
```

```
array([[1.0000e+00, 1.3000e+02, 1.6900e+04]
       [1.0000e+00, 1.6500e+02, 2.7225e+04]
       [1.0000e+00, 1.5000e+02, 2.2500e+04]
       [1.0000e+00, 1.5000e+02, 2.2500e+04]
       [1.0000e+00, 1.4000e+02, 1.9600e+04]
```

```
# hp2 is array (no need to reshape)
```

```
df2 = pd.DataFrame(hp2)
df2[:5]
```

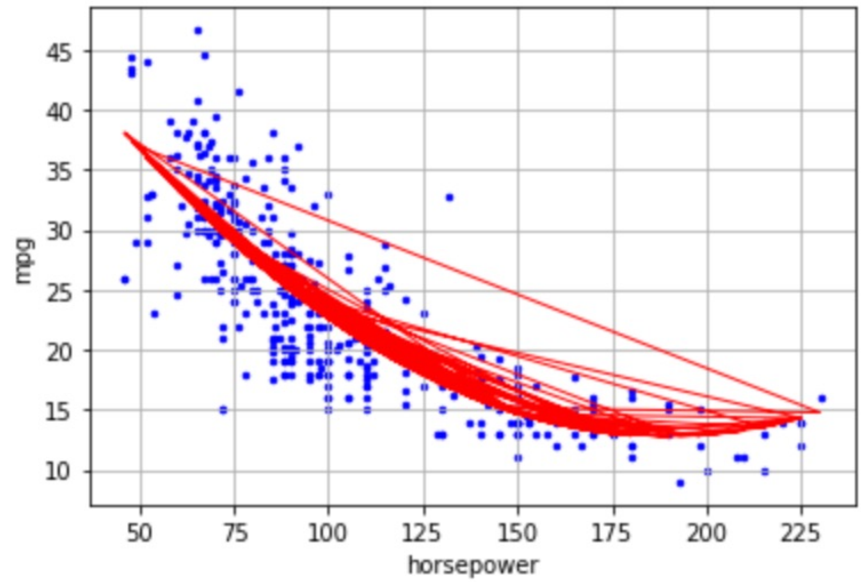
	0	1	2
0	1.0	130.0	16900.0
1	1.0	165.0	27225.0
2	1.0	150.0	22500.0
3	1.0	150.0	22500.0
4	1.0	140.0	19600.0

POLYNOMIAL REGRESSION – QUADRATIC MODEL

.

```
model2 = LinearRegression().fit(hp2,mpg)
yhat2 = model2.predict(hp2)
```

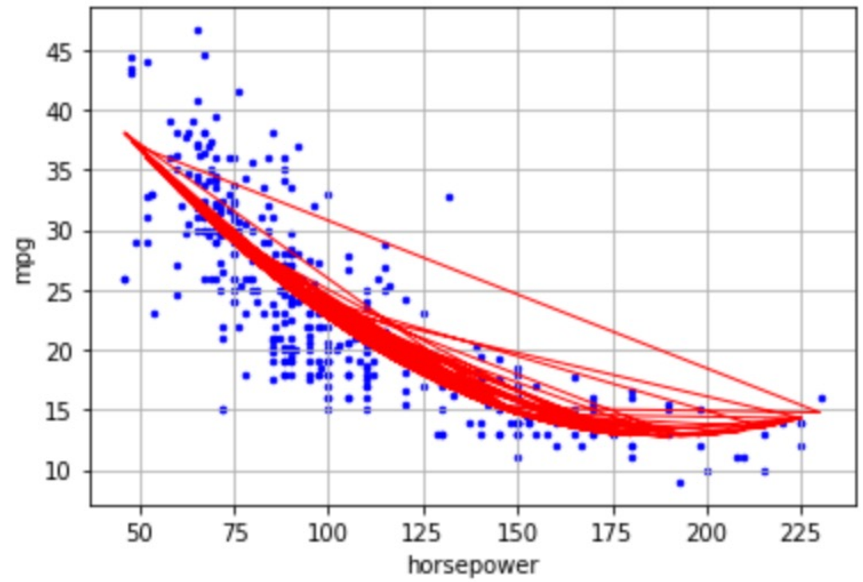
```
plt.scatter(hp,mpg,c='b',s=6)
plt.plot(hp,yhat2,c='r',lw=1)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()
```



POLYNOMIAL REGRESSION – QUADRATIC MODEL

```
model2 = LinearRegression().fit(hp2,mpg)
yhat2 = model2.predict(hp2)
```

```
plt.scatter(hp,mpg,c='b',s=6)
plt.plot(hp,yhat2,c='r',lw=1)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()
```



hp (and hp2) is not sorted.
Connecting unsorted points yields this plot
(see previous slide)

POLYNOMIAL REGRESSION – QUADRATIC MODEL

sort by *horsepower*, then transform, fit, and plot again

```
d2 = auto.sort_values('horsepower')
```

```
d2[:5]
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
19	26.0	4	97.0	46	1835	20.5	70	2
101	26.0	4	97.0	46	1950	21.0	73	2
324	43.4	4	90.0	48	2335	23.7	80	2
323	44.3	4	90.0	48	2085	21.7	80	2
242	43.1	4	90.0	48	1985	21.5	78	2

```
mpg = d2.mpg
hp = d2.horsepower
hp1 = hp.values.reshape(-1,1)
```

now hp1 is sorted

POLYNOMIAL REGRESSION – QUADRATIC MODEL

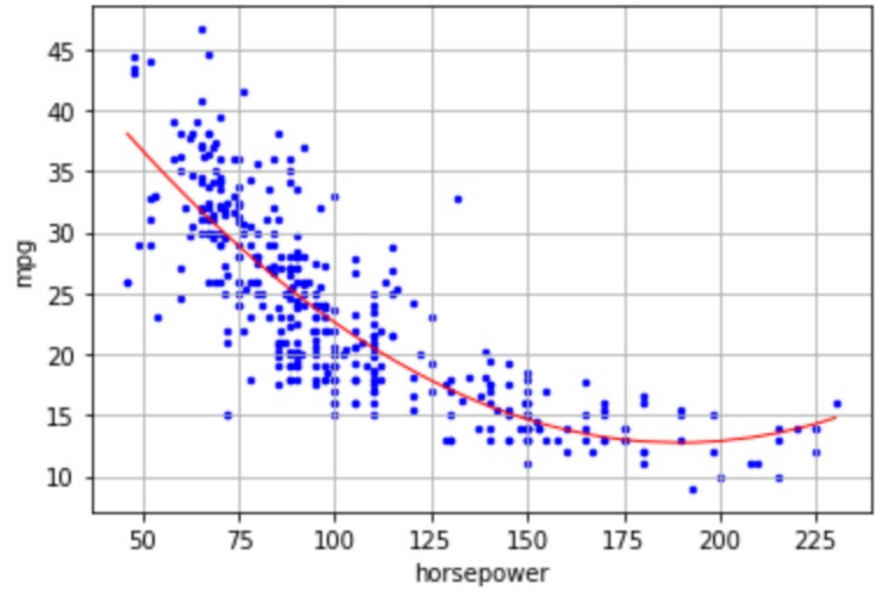
.

```
mpg = d2.mpg
hp = d2.horsepower
hp1 = hp.values.reshape(-1,1)
```

```
poly2 = PolynomialFeatures(degree=2)
hp2 = poly2.fit_transform(hp1)
```

```
model2 = LinearRegression().fit(hp2,mpg)
yhat2 = model2.predict(hp2)
```

```
plt.scatter(hp,mpg,c='b',s=6)
plt.plot(hp,yhat2,c='r',lw=1)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()
```



POLYNOMIAL REGRESSION – MODEL DEGREE 5

Model (degree 5)

```
poly5 = PolynomialFeatures(degree=5)
hp5 = poly5.fit_transform(hp1)
hp5.shape
```

```
(392, 6)
```

```
df5 = pd.DataFrame(hp5)
df5[:5]
```

	0	hp	2	3	4	5
0	1.0	46.0	2116.0	97336.0	4477456.0	205962976.0
1	1.0	46.0	2116.0	97336.0	4477456.0	205962976.0
2	1.0	48.0	2304.0	110592.0	5308416.0	254803968.0
3	1.0	48.0	2304.0	110592.0	5308416.0	254803968.0
4	1.0	48.0	2304.0	110592.0	5308416.0	254803968.0

POLYNOMIAL REGRESSION – MODEL DEGREE 5

Model (degree 5)

```
poly5 = PolynomialFeatures(degree=5)
hp5 = poly5.fit_transform(hp1)
hp5.shape
```

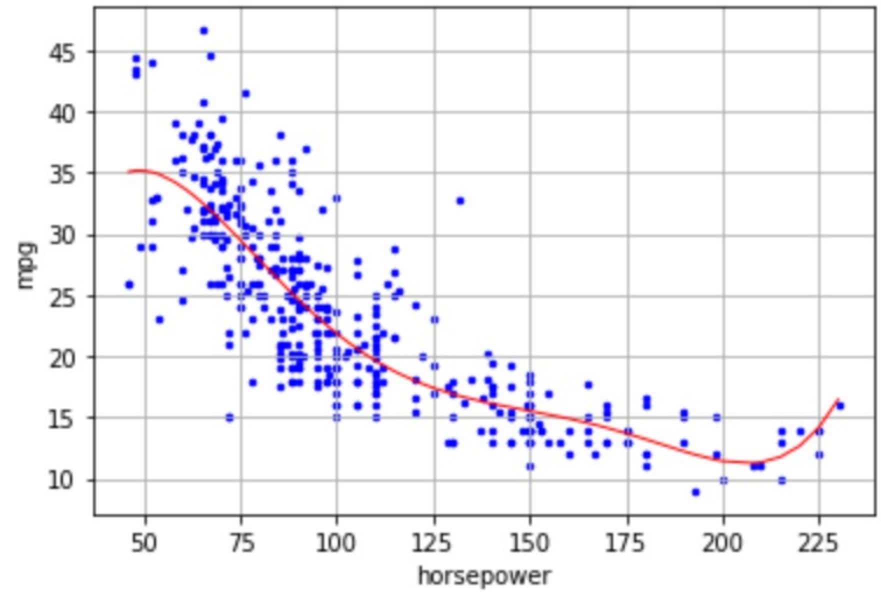
```
(392, 6)
```

```
df5 = pd.DataFrame(hp5)
df5[:5]
```

	0	1	2	3	4	5
0	1.0	46.0	2116.0	97336.0	4477456.0	205962976.0
1	1.0	46.0	2116.0	97336.0	4477456.0	205962976.0
2	1.0	48.0	2304.0	110592.0	5308416.0	254803968.0
3	1.0	48.0	2304.0	110592.0	5308416.0	254803968.0
4	1.0	48.0	2304.0	110592.0	5308416.0	254803968.0

```
model5 = LinearRegression().fit(hp5,mpg)
yhat5 = model5.predict(hp5)
```

```
plt.scatter(hp,mpg,c='b',s=6)
plt.plot(hp,yhat5,c='r',lw=1)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()
```



POLYNOMIAL REGRESSION – MODELS DEGREE 1 TO 4 (LOOP)

Model (degree 5)

```
poly5 = PolynomialFeatures(degree=5)
hp5 = poly5.fit_transform(hp1)

model5 = LinearRegression().fit(hp5,mpg)
yhat5 = model5.predict(hp5)
```

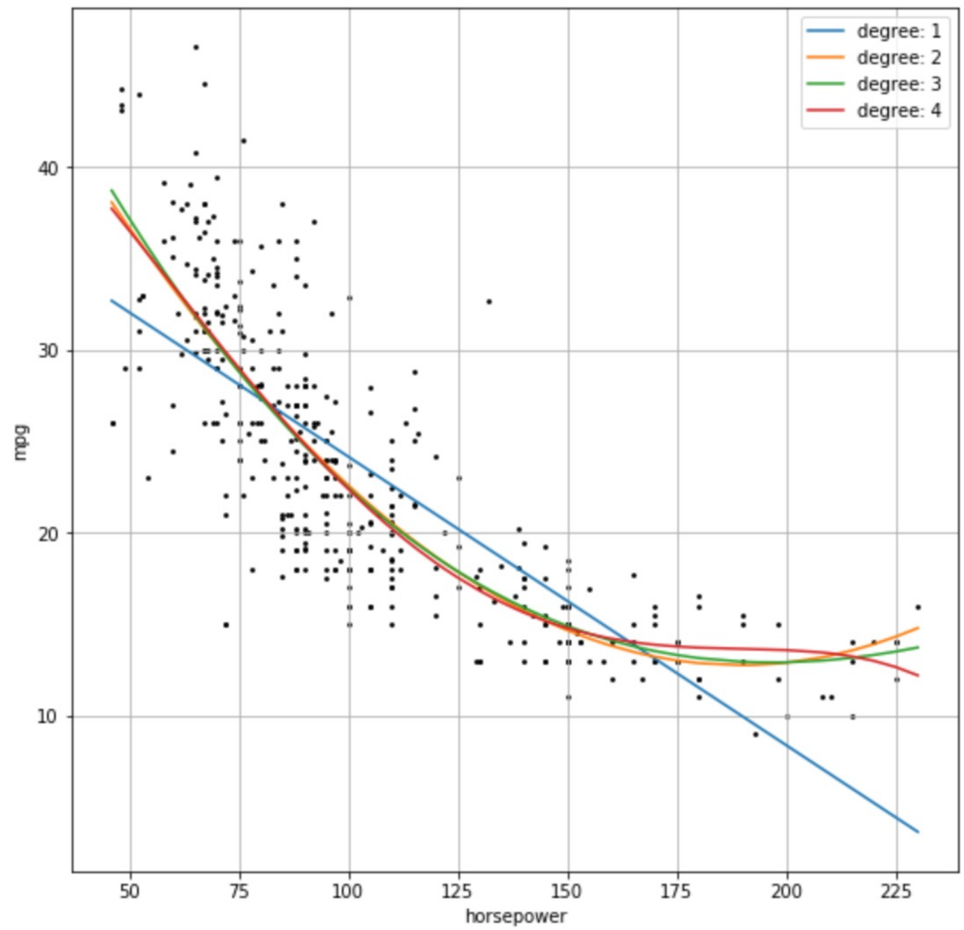
```
for j in range(1,5):
    hp_j = PolynomialFeatures(degree = j).\
                    fit_transform(hp1)
    model = LinearRegression().fit(hp_j,mpg)
    yhat = model.predict(hp_j)
```

POLYNOMIAL REGRESSION

Models (all up to degree 4)

```
plt.figure(figsize=(9,9))
plt.scatter(hp,mpg,color='k',s = 3)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()

for j in range(1,5):
    hp_j = PolynomialFeatures(degree = j).\
                    fit_transform(hp1)
    model = LinearRegression().fit(hp_j,mpg)
    yhat = model.predict(hp_j)
    plt.plot(hp,yhat,label = 'degree: '+str(j))
    plt.legend()
```



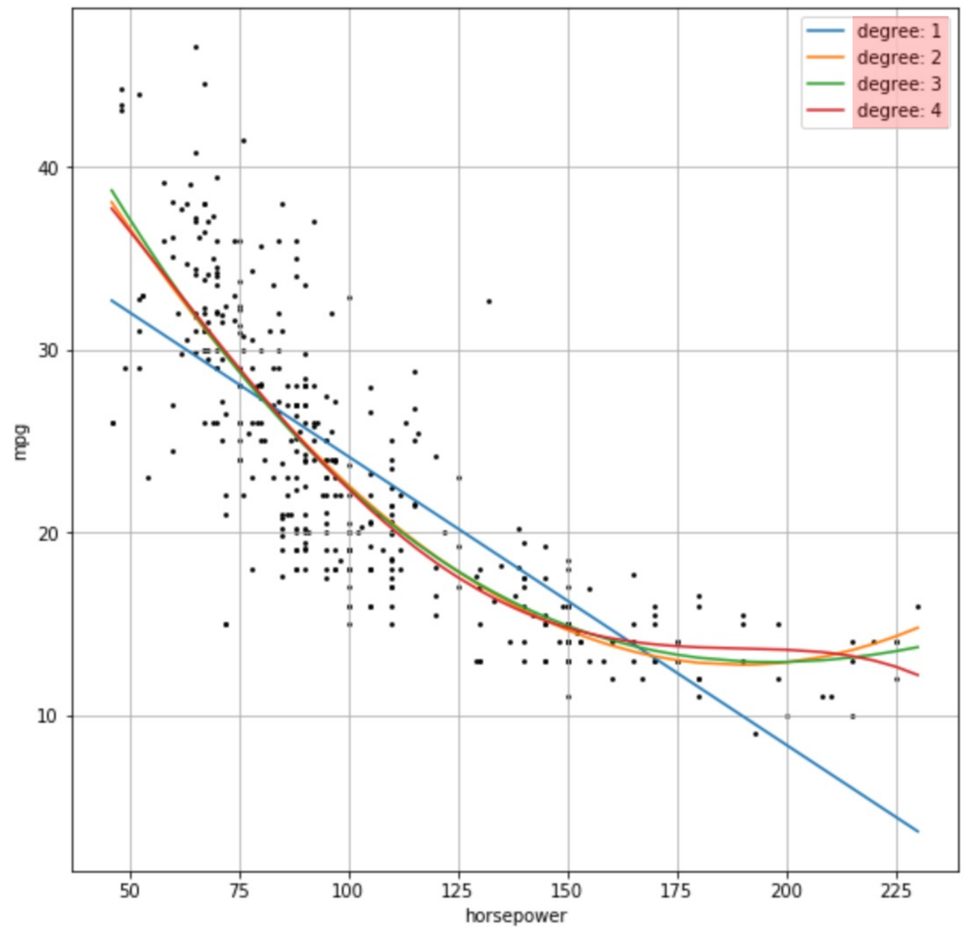
POLYNOMIAL REGRESSION

Models (all up to degree 4)

```
plt.figure(figsize=(9,9))
plt.scatter(hp,mpg,color='k',s = 3)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.grid()

for j in range(1,5):
    hp_j = PolynomialFeatures(degree = j).\
                    fit_transform(hp1)
    model = LinearRegression().fit(hp_j,mpg)
    yhat = model.predict(hp_j)
    plt.plot(hp,yhat,label = 'degree: '+str(j))
    plt.legend()
```

labels for the legend items



POLYNOMIAL REGRESSION

Holdout cross-validation to select the best polynomial model

CROSS VALIDATION

.

DataFrame with variables

```
list1 = ['horsepower','mpg']
df = auto[list1]
df[:5]
```

	horsepower	mpg
0	130	18.0
1	165	15.0
2	150	18.0
3	150	16.0
4	140	17.0

Select response and predictor(s)

```
mpg = df.mpg
hp1 = df.drop(['mpg'],axis = 1)
hp1[:5]
```

	horsepower
0	130
1	165
2	150
3	150
4	140

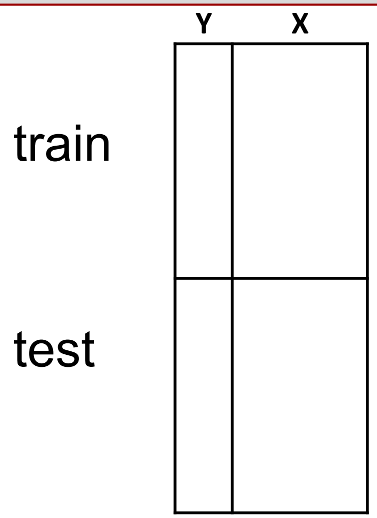
```
mpg[:5]
```

```
0     18.0
1     15.0
2     18.0
3     16.0
4     17.0
```

HOLDOUT CROSS VALIDATION

Split the data into

- Train set 50%
- Test set 50%



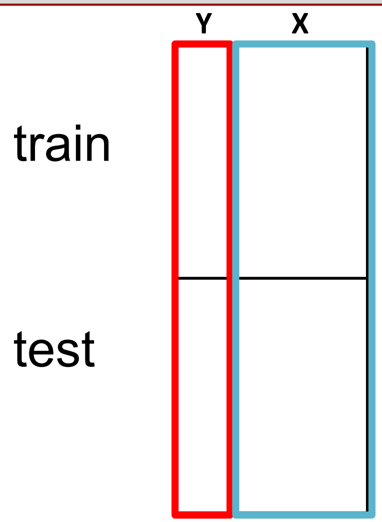

```
from sklearn.model_selection import train_test_split
```

```
hp_train,hp_test,mpg_train,mpg_test = train_test_split(hp1,mpg,
                                                       test_size=0.5,
                                                       random_state=1)
```

HOLDOUT CROSS VALIDATION

Split the data into

- Train set 50%
- Test set 50%




```
from sklearn.model_selection import train_test_split
```

```
hp_train,hp_test,mpg_train,mpg_test = train_test_split(hp1,mpg,
                                                        test_size=0.5,
                                                        random_state=1)
```

POLYNOMIAL REGRESSION - VALIDATION APPROACH

Model (degree 2)

add a column of ones and
a column of squared values
to hp_train, hp_test

```
form = PolynomialFeatures(degree=2)
hp_train_2 = form.fit_transform(hp_train)
hp_test_2 = form.fit_transform(hp_test)
```

POLYNOMIAL REGRESSION - VALIDATION APPROACH

Model (degree 2)

- train set used to fit() the model
- test set used to predict yhat values

```
form = PolynomialFeatures(degree=2)
hp_train_2 = form.fit_transform(hp_train)
hp_test_2 = form.fit_transform(hp_test)
```

```
m2 = LinearRegression().fit(hp_train_2,mpg_train)
yhat2 = m2.predict(hp_test_2)
```

POLYNOMIAL REGRESSION - VALIDATION APPROACH

Model (degree 2)

```
form = PolynomialFeatures(degree=2)
hp_train_2 = form.fit_transform(hp_train)
hp_test_2 = form.fit_transform(hp_test)
```

```
m2 = LinearRegression().fit(hp_train_2,mpg_train)
yhat2 = m2.predict(hp_test_2)
```

```
# mspe
res2 = (yhat2 - mpg_test)**2
mspe2 = np.mean(res2)
mspe2
```

```
18.848292603275382
```

POLYNOMIAL REGRESSION - VALIDATION APPROACH

Model (degree 5)

```
form = PolynomialFeatures(degree=5)
hp_train_5 = form.fit_transform(hp_train)
hp_test_5 = form.fit_transform(hp_test)
```

```
m5 = LinearRegression().fit(hp_train_5,mpg_train)
yhat5 = m5.predict(hp_test_5)
```

```
res5 = (yhat5 - mpg_test)**2
mspe5 = np.mean(res5)
mspe5
```

```
18.324168662607967
```

POLYNOMIAL REGRESSION - VALIDATION APPROACH

Model (all degrees)

into a loop

```
form = PolynomialFeatures(degree=5)
hp_train_5 = form.fit_transform(hp_train)
hp_test_5 = form.fit_transform(hp_test)

m5 = LinearRegression().fit(hp_train_5,mpg_train)
yhat5 = m5.predict(hp_test_5)

res5 = (yhat5 - mpg_test)**2
mspe5 = np.mean(res5)
mspe5
```

```
18.324168662607967
```

POLYNOMIAL REGRESSION - VALIDATION APPROACH

Models (all degrees)

```
mspe = []
for i in range(1,15):
    form = PolynomialFeatures(degree=i)
    hp_train_i = form.fit_transform(hp_train)
    hp_test_i = form.fit_transform(hp_test)
    model = LinearRegression().fit(hp_train_i,mpg_train)
    yhat_i = model.predict(hp_test_i)
    sqres = (yhat_i - mpg_test)**2
    mspe_i = np.mean(sqres)
    mspe.append(mspe_i)
```

POLYNOMIAL REGRESSION - VALIDATION APPROACH

Models (all degrees)

```
mspe = []
for i in range(1,15):                                   in the loop
    form = PolynomialFeatures(degree=i)
    hp_train_i = form.fit_transform(hp_train)
    hp_test_i = form.fit_transform(hp_test)
    model = LinearRegression().fit(hp_train_i,mpg_train)
    yhat_i = model.predict(hp_test_i)
    sqres = (yhat_i - mpg_test)**2
    mspe_i = np.mean(sqres)
    mspe.append(mspe_i)
```

POLYNOMIAL REGRESSION - VALIDATION APPROACH

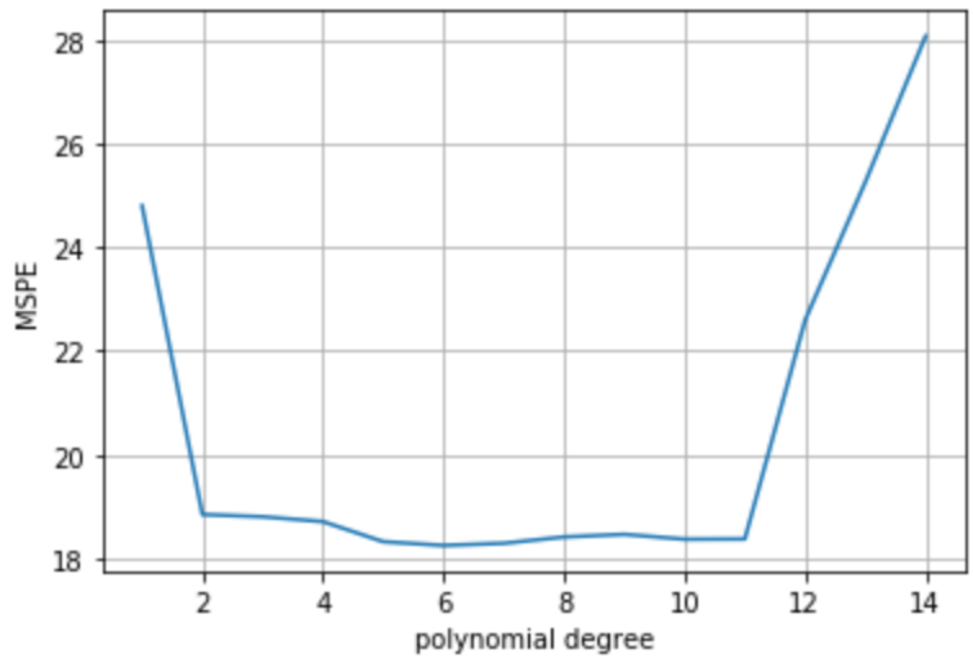
.

```
mspe.insert(0,np.nan)
mspe
```

```
[nan,
 24.80212062059357,
 18.848292603275382,
 18.805111358625545,
 18.711722273821835,
 18.324168662607967,
 18.2513910499586 4,
 18.29312744057875,
 18.415019005606478,
 18.4649796002067 8,
 18.371543424469433,
 18.376202865567404,
 22.5956914252877 4,
 25.260683389280025,
 28.07175102447582]
```

degree 1 →
degree 2 →
.
.
.
degree 14

```
plt.plot(mspe)
plt.xlabel('polynomial degree')
plt.ylabel('MSPE')
plt.grid()
```



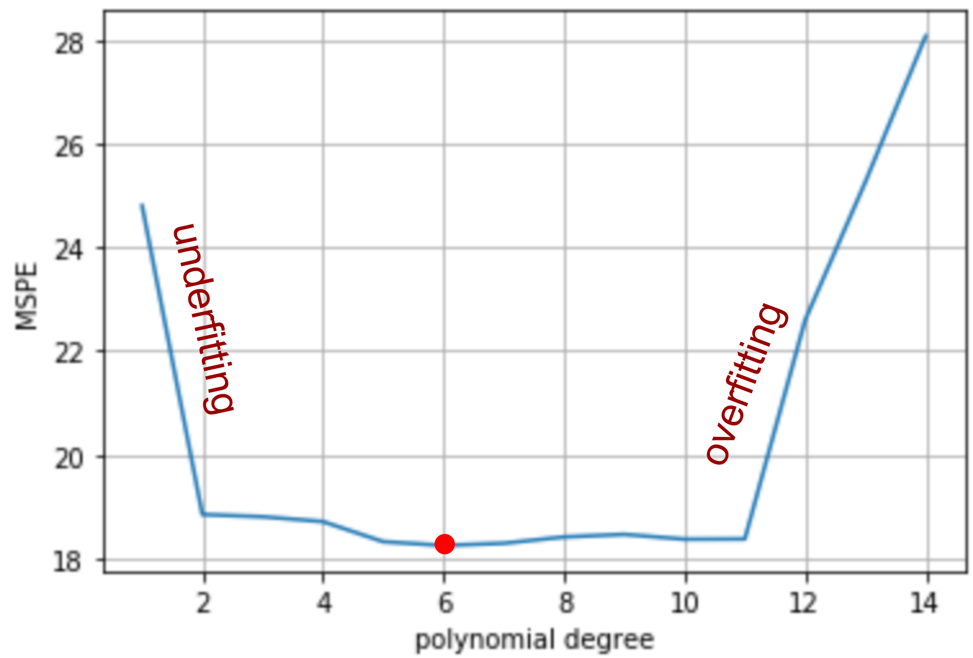
POLYNOMIAL REGRESSION - VALIDATION APPROACH

```
mspe.insert(0,np.nan)
mspe
```

```
[nan,
 24.80212062059357,
 18.848292603275382,
 18.805111358625545,
 18.711722273821835,
 18.324168662607967,
 18.25139104995864,
 18.29312744057875,
 18.415019005606478,
 18.46497960020678,
 18.371543424469433,
 18.376202865567404,
 22.59569142528774,
 25.260683389280025,
 28.07175102447582]
```

degree 6 →

```
plt.plot(mspe)
plt.xlabel('polynomial degree')
plt.ylabel('MSPE')
plt.grid()
```



polynomial degree 6 is best model

POLYNOMIAL REGRESSION

LOOCV
to select the
best polynomial model

POLYNOMIAL REGRESSION - LOOCV

for KFold CV and LOOCV

```
from sklearn.model_selection import cross_val_score
```

no need to split into
train and test sets

```
mspe1 = cross_val_score(LinearRegression(),hp1,mpg,
                        cv = LeaveOneOut(),scoring = measure)
```

POLYNOMIAL REGRESSION - LOOCV

for KFold CV and LOOCV

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
```

no need to split into
train and test sets

```
mspe1 = cross_val_score(LinearRegression(),hp1,mpg,
                        cv = LeaveOneOut(),scoring = measure)
```

POLYNOMIAL REGRESSION - LOOCV

for KFold CV and LOOCV

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
measure = 'neg_mean_squared_error'
```

no need to split into train and test sets

```
mspe1 = cross_val_score(LinearRegression(),hp1,mpg,
                        cv = LeaveOneOut(),scoring = measure)
```

POLYNOMIAL REGRESSION - LOOCV

for KFold CV and LOOCV

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
measure = 'neg_mean_squared_error'
```

no need to split into train and test sets

```
mspe1 = cross_val_score(LinearRegression(),hp1,mpg,
                        cv = LeaveOneOut(),scoring = measure)
cvmspe1 = mspe1.mean()
-cvmspe1
```

mspe

```
24.231513517929226
```

POLYNOMIAL REGRESSION - LOOCV

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
measure = 'neg_mean_squared_error'
```

linear model

```
mspe1 = cross_val_score(LinearRegression(),hp1,mpg,
                        cv = LeaveOneOut(),scoring = measure)
cvmspe1 = mspe1.mean()
-cvmspe1
```

```
24.231513517929226
```

degree 2 model

```
mspe2 = cross_val_score(LinearRegression(),hp2,mpg,
                        cv = LeaveOneOut(),scoring = measure)
cvmspe2 = mspe2.mean()
-cvmspe2
```

```
19.24821312448941
```

POLYNOMIAL REGRESSION - LOOCV

All degrees

```
model = LinearRegression()
```

```
cvmspe = [-cvmspe1]

for i in range(2,14):
    form = PolynomialFeatures(degree = i)
    hp_i = form.fit_transform(hp1)
    mspe = cross_val_score(model,hp_i,mpg,
                           cv = LeaveOneOut(),
                           scoring = measure)
    cvmspe.append(-mspe.mean())
```

```
cvmspe.insert(0,np.nan)
```

mspe values

```
[nan,
 24.23151351792926,
 19.24821312448941,
 19.33498406411397,
 19.42443030941186,
 19.03321184297839,
 18.97301273775870,
 19.12563965510483,
 19.22423029373206,
 19.13385650111735,
 18.94583743686193,
 19.1250385409314,
 24.14841810216805,
 27.76341869209905]
```

POLYNOMIAL REGRESSION - LOOCV

.

```
model = LinearRegression()
```

```
cvmspe = [-cvmspe1]

for i in range(2,14):
    form = PolynomialFeatures(degree = i)
    hp_i = form.fit_transform(hp1)
    mspe = cross_val_score(model,hp_i,mpg,
                           cv = LeaveOneOut(),
                           scoring = measure)
    cvmspe.append(-mspe.mean())
```

```
cvmspe.insert(0,np.nan)
```

best mspe values

```
[nan,
 24.231513517929226,
 19.24821312448941,
 19.33498406411397,
 19.42443030941886,
 19.033211842978396,
 18.973012737758705,      degree 6
 19.125639655104838,
 19.22423029373206,
 19.13385650117357,
 18.945837436861932,      degree 10
 19.1250385409314,
 24.14841810216805,
 27.76341869209905]
```

POLYNOMIAL REGRESSION - LOOCV

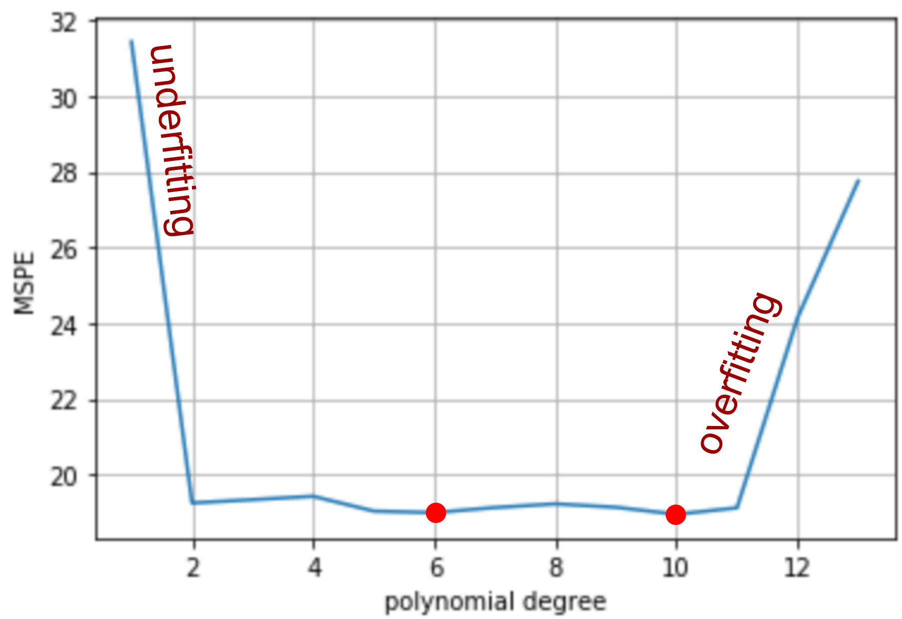
```
model = LinearRegression()
```

```
cvmspe = [-cvmspe1]

for i in range(2,14):
    form = PolynomialFeatures(degree = i)
    hp_i = form.fit_transform(hp1)
    mspe = cross_val_score(model,hp_i,mpg,
                           cv = LeaveOneOut(),
                           scoring = measure)
    cvmspe.append(-mspe.mean())
```

```
cvmspe.insert(0,np.nan)
```

```
plt.plot(cvmspe)
plt.xlabel('polynomial degree')
plt.ylabel('MSPE')
plt.grid()
```



best models are degree 10 and 6

POLYNOMIAL REGRESSION

K-Fold cross-validation to select the best polynomial model

POLYNOMIAL REGRESSION – KFold Cross Validation

.

```python
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

```python
measure = 'neg_mean_squared_error'
```

linear model

```python
mspe1 = cross_val_score(LinearRegression(),hp1,mpg,
                        cv = KFold(n_splits = 5),
                        scoring = measure)
```

```python
cvmspe1 = mspe1.mean()
-cvmspe1
```

```
31.447014088557513
```

POLYNOMIAL REGRESSION – KFold Cross Validation

.

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

```
measure = 'neg_mean_squared_error'
```

linear model

```
mspe1 = cross_val_score(LinearRegression(),hp1,mpg,
                        cv = KFold(n_splits = 5),
                        scoring = measure)
```

```
cvmspe1 = mspe1.mean()
-cvmspe1
```

```
31.447014088557513
```

POLYNOMIAL REGRESSION - KFold Cross-validation

All polynomial models up to degree 13

```
measure = 'neg_mean_squared_error'
```

```
cvmspe = [-cvmspe1]
for i in range(2,14):
    form = PolynomialFeatures(degree = i)
    hp_i = form.fit_transform(hp1)
    mspe = cross_val_score(LinearRegression(),
                           hp_i,mpg,
                           cv = KFold(n_splits = 5),
                           scoring = measure)
    cvmspe.append(-mspe.mean())
```

```
cvmspe.insert(0,np.nan)
```

POLYNOMIAL REGRESSION - KFold Cross-validation

All polynomial models up to degree 13

```
measure = 'neg_mean_squared_error'
```

```
cvmspe = [-cvmspe1]
for i in range(2,14):
    form = PolynomialFeatures(degree = i)
    hp_i = form.fit_transform(hp1)
    mspe = cross_val_score(LinearRegression(),
                           hp_i,mpg,
                           cv = KFold(n_splits = 5),
                           scoring = measure)
    cvmspe.append(-mspe.mean())
```

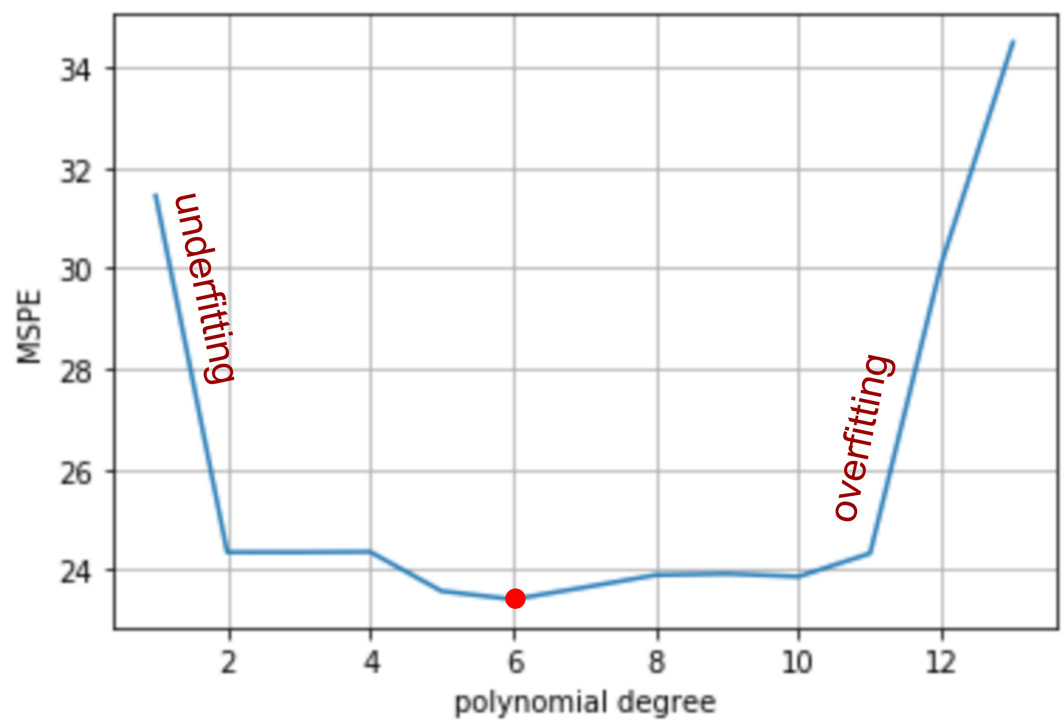
```
cvmspe.insert(0,np.nan)
```

```
[nan,
 31.447014088557513,
 24.34715884367356,
 24.34607807898161,
 24.35538965780616,
 23.57323974697034,
 23.407797059555385,
 23.646448063420337,
 23.892270279390385,
 23.918015840113277,
 23.862407621094167,
 24.323550189304466,
 30.101687309472418,
 34.50104373957487]
```

degree 6

POLYNOMIAL REGRESSION - KFold Cross-validation

```
plt.plot(cvmspe)
plt.xlabel('polynomial degree')
plt.ylabel('MSPE')
plt.grid()
```



best model is degree 6

Sklearn - Cross Validation Summary

.

HOLDOUT

Cross Validation

```
from sklearn.model_selection import train_test_split
```

```
hp_train,hp_test,mpg_train,mpg_test = train_test_split(hp1,mpg,
                                                       test_size=0.5,
                                                       random_state=1)
m2 = LinearRegression().fit(hp_train_2,mpg_train)
yhat2 = m2.predict(hp_test_2)
```

```
# mspe
res2 = (yhat2 - mpg_test)**2
mspe2 = np.mean(res2)
```

k-Fold

Cross Validation

```
measure = 'neg_mean_squared_error'
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
mspe1 = cross_val_score(LinearRegression(),hp1,mpg,
                        cv = KFold(n_splits = 5),
                        scoring = measure)
mspe1.mean()
```