Assignment 2

Regression Models

Verena Blaschke

June 06, 2018

Assignment 2

I: Preprocessing

II: Linear Regression

III: Polynomial Models

IV: Categorical Predictors

V: Regularization

I: Preprocessing

timestamps.train	CEST
1522533600	2018-APR-01 00:00:00
1522533600	2018-APR-01 00:00:00
1522533602	2018-APR-01 00:00:02
...	
1525125597	2018-APR-30 23:59:57
1525125598	2018-APR-30 23:59:58

timestamps.test	CEST
1525647600	2018-MAY-07 01:00:00
...	
1526248799	2018-MAY-13 23:59:59

I: Preprocessing

timestamps.train	CEST
1522533600	2018-APR-01 00:00:00
1522533600	2018-APR-01 00:00:00
1522533602	2018-APR-01 00:00:02
...	
1525125597	2018-APR-30 23:59:57
1525125598	2018-APR-30 23:59:58

hours	[[0]	[1]	[2]	[3]	...
tallies	[[5682]	[3480]	[1782]	[1029]	...
	[22]	[23]	[0]	[1]	...
	[10457]	[7714]	[4714]	[2412]	...
	[20]	[21]	[22]	[23]]	
	[17683]	[13710]	[10774]	[9246]]	

I: Preprocessing

timestamps.train	CEST
1522533600	2018-APR-01 00:00:00
1522533600	2018-APR-01 00:00:00
1522533602	2018-APR-01 00:00:02
...	
1525125597	2018-APR-30 23:59:57
1525125598	2018-APR-30 23:59:58

hours	[[0]	[1]	[2]	[3]	...
tallies	[[5682]	[3480]	[1782]	[1029]	...
	[22]	[23]	[0]	[1]	...
	[10457]	[7714]	[4714]	[2412]	...
	[20]	[21]	[22]	[23]]	
	[17683]	[13710]	[10774]	[9246]]	

(We're only using the hour values as predictors–what else could we use?)

I: Preprocessing

1. read the files

I: Preprocessing

1. read the files

```
import gzip
with gzip.open(filename) as f:
```

I: Preprocessing

1. read the files

```
import gzip
with gzip.open(filename) as f:
```

2. convert timestamps

```
secs = int(line)
date = time.localtime(secs)
```

I: Preprocessing

1. read the files

```
import gzip
with gzip.open(filename) as f:
```

2. convert timestamps

```
secs = int(line)
date = time.localtime(secs)
```

3. differentiate between identical hour values from different days

I: Preprocessing

1. read the files

```
import gzip
with gzip.open(filename) as f:
```

2. convert timestamps

```
secs = int(line)
date = time.localtime(secs)
```

3. differentiate between identical hour values from different days e.g. by mapping hours to keys like `YYYY-MM-DD-HH`

```
key = (date.tm_year, date.tm_mon, date.tm_mday,
       date.tm_hour)
```

I: Preprocessing

3. differentiate between identical hour values from different days e.g. by mapping hours to keys like `YYYY-MM-DD-HH`

```
key = (date.tm_year, date.tm_mon, date.tm_mday,
       date.tm_hour)
```

I: Preprocessing

3. differentiate between identical hour values from different days e.g. by mapping hours to keys like `YYYY-MM-DD-HH`

```
key = (date.tm_year, date.tm_mon, date.tm_mday,
       date.tm_hour)
```

4. count the number of entries per hour

I: Preprocessing

3. differentiate between identical hour values from different days e.g. by mapping hours to keys like YYYY-MM-DD-HH

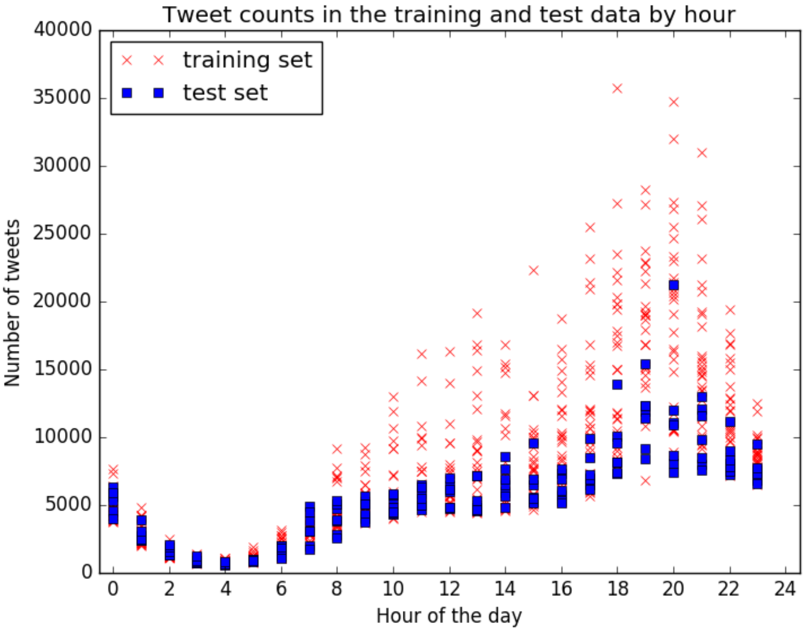
```
key = (date.tm_year, date.tm_mon, date.tm_mday,
       date.tm_hour)
```

4. count the number of entries per hour
5. convert to a numpy array for the hours and one for the tallies and reshape them to (n_samples, 1)

```
my_array.reshape(-1, 1)
```

- training data: (720, 1)
- test data: (167, 1)

I: Preprocessing



II: Linear Regression

```
model = sklearn.linear_model.LinearRegression()
model.fit(x_train, y_train)
r2_train = model.score(x_train, y_train)
r2_test = model.score(x_test, y_test)
```

II: Linear Regression

```
model = sklearn.linear_model.LinearRegression()
model.fit(x_train, y_train)
r2_train = model.score(x_train, y_train)
r2_test = model.score(x_test, y_test)
```

Never fit your statistical model on the test set!

II: Linear Regression

```
model = sklearn.linear_model.LinearRegression()
model.fit(x_train, y_train)
r2_train = model.score(x_train, y_train)
r2_test = model.score(x_test, y_test)
```

Never fit your statistical model on the test set!

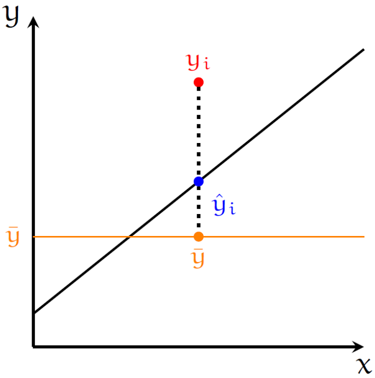
R^2 (training set)	0.5180
R^2 (test set)	0.0540

II: Linear Regression

R^2 (training set)	0.5180
R^2 (test set)	0.0540

Our model explains 51.8% (5.4%) of the training (test) data set's variance.

$$R^2 = \frac{\sum_i^n (\hat{y}_i - \bar{y})^2}{\sum_i^n (y_i - \bar{y})^2} = 1 - \frac{\text{MSE}}{\sigma_y^2}$$



II: Linear Regression

```
x_predict = np.array([0, 8, 12, 18, 23]).reshape(-1, 1)
y_predicted = model.predict(x_predict)
```

0	8	12	18	23
79.61	5208.39	7772.78	11619.37	14824.86

II: Linear Regression

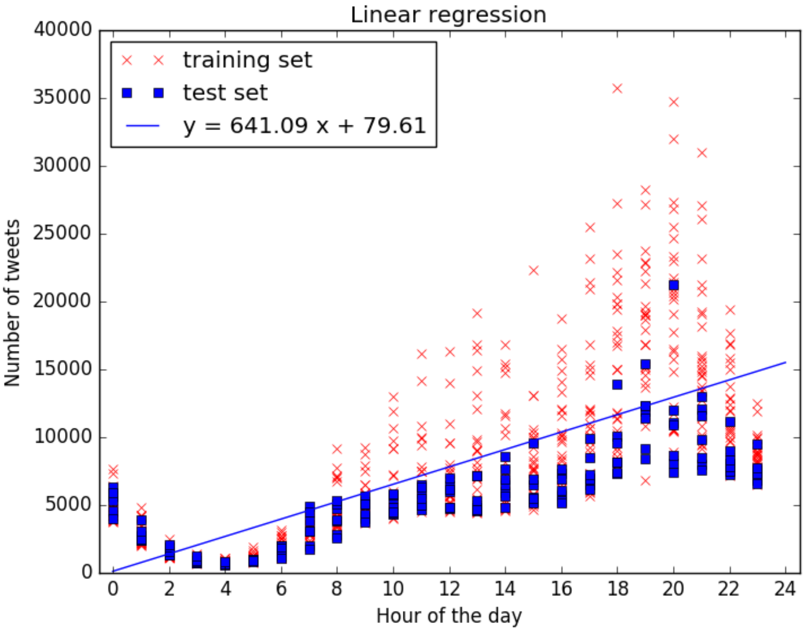
```
x_predict = np.array([0, 8, 12, 18, 23]).reshape(-1, 1)
y_predicted = model.predict(x_predict)
```

0	8	12	18	23
79.61	5208.39	7772.78	11619.37	14824.86

$$y = 641.09x + 79.61$$

(get the model coefficients via `model.coef_` and `model.intercept_`)

II: Linear Regression



III: Polynomial Models

Our linear model didn't do so well...
increase the polynomial degree?

$$y = \sum_{0}^{n} b_i x^i$$

III: Polynomial Models

Our linear model didn't do so well...
increase the polynomial degree?

$$y = \sum_{0}^{n} b_i x^i$$

Prepare suitable input data (training and test):

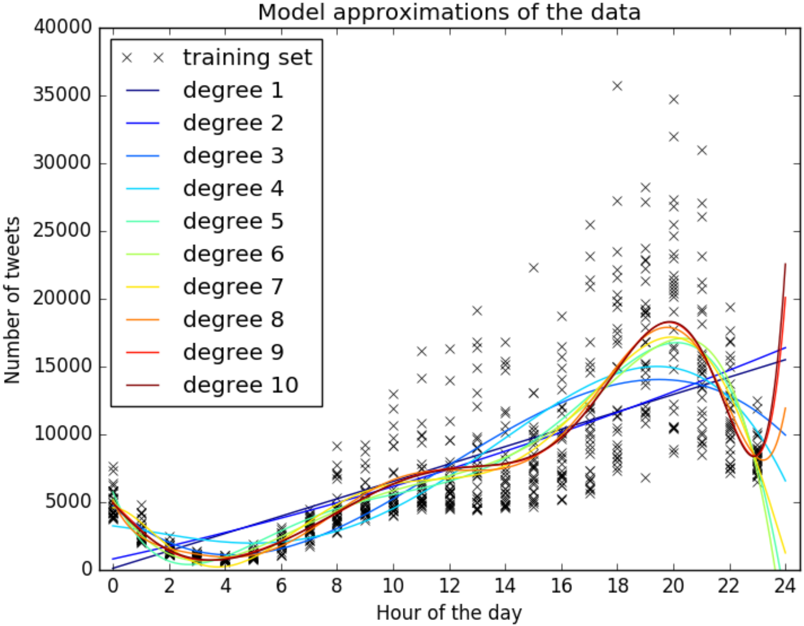
```
x_polynomial = sklearn.preprocessing.PolynomialFeatures(
    degree=n
    ).fit_transform(x)
```

III: Polynomial Models

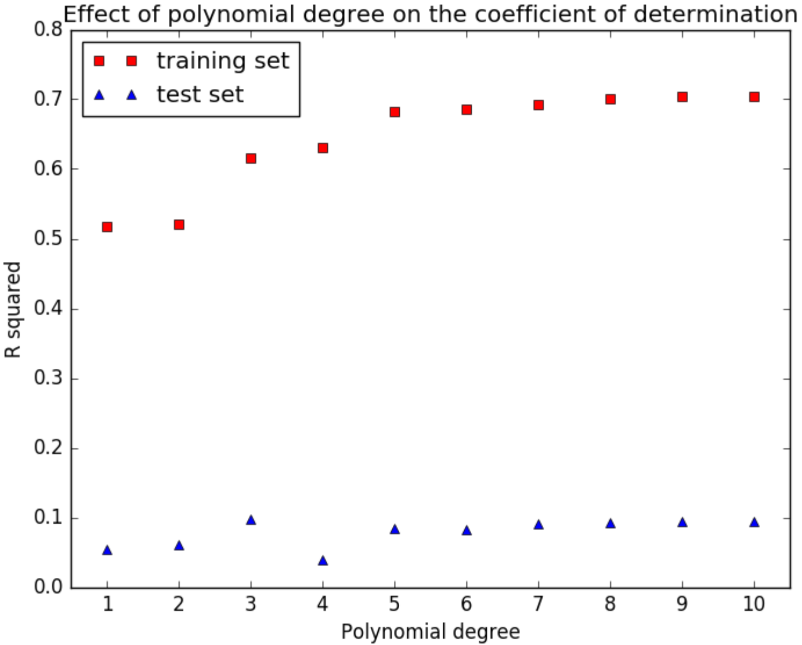
What does this do?

```
>>> p = sklearn.preprocessing.PolynomialFeatures(5)
>>> x = np.arange(1, 5).reshape(-1, 1)
>>> x
array([[1],
       [2],
       [3],
       [4]])
>>> p.fit_transform(x).astype(np.int64)
array([[   1,    1,    1,    1,    1,    1],
       [   1,    2,    4,    8,   16,   32],
       [   1,    3,    9,   27,   81,  243],
       [   1,    4,   16,   64,  256, 1024]], dtype=int64)
```

III: Polynomial Models



III: Polynomial Models



IV: Categorical Predictors

None of these models are great...
use categorical predictors?

```
enc = sklearn.preprocessing.OneHotEncoder()
enc.fit(x_train)
x_train_1hot = enc.transform(x_train)
x_test_1hot = enc.transform(x_test)
```

IV: Categorical Predictors

None of these models are great...
use categorical predictors?

```
enc = sklearn.preprocessing.OneHotEncoder()
enc.fit(x_train)
x_train_1hot = enc.transform(x_train)
x_test_1hot = enc.transform(x_test)
```

With our input data creating separate encoders for training and test set works, but what if the data sets don't contain the same number of distinct values?

IV: Categorical Predictors

None of these models are great...
use categorical predictors?

```
enc = sklearn.preprocessing.OneHotEncoder()
enc.fit(x_train)
x_train_1hot = enc.transform(x_train)
x_test_1hot = enc.transform(x_test)
```

With our input data creating separate encoders for training and test set works, but what if the data sets don't contain the same number of distinct values?

R^2 (training set)	0.7055
R^2 (test set)	0.0902

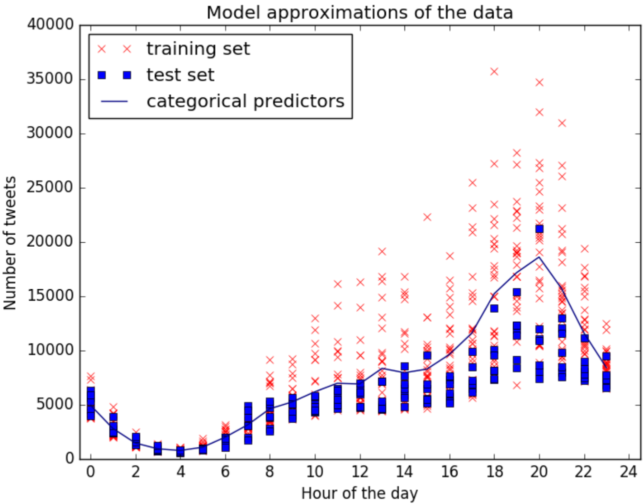
V: Regularization

The model is doing okay on the training data, but it doesn't do well on the test set.

V: Regularization

The model is doing okay on the training data, but it doesn't do well on the test set.

Overfitting!



V: Regularization

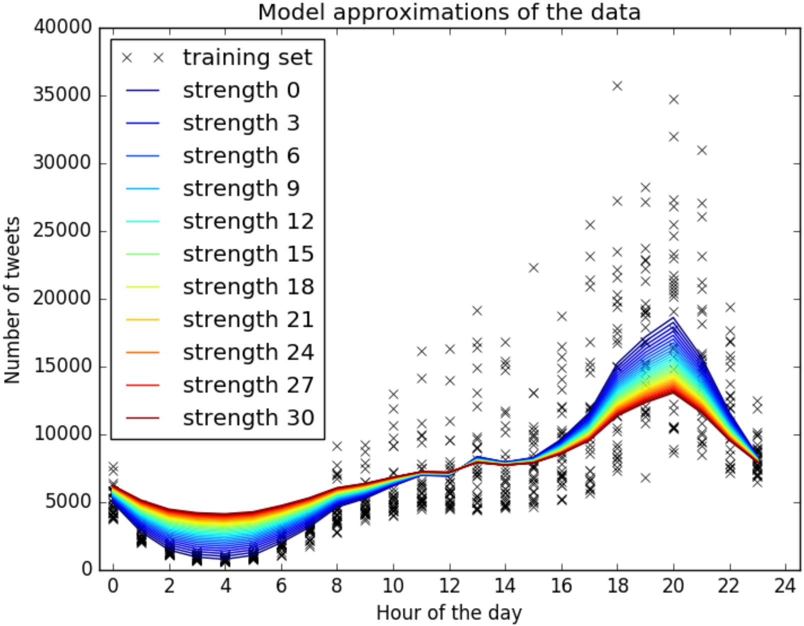
Overfitting!

Adding a regularization term to our loss function:

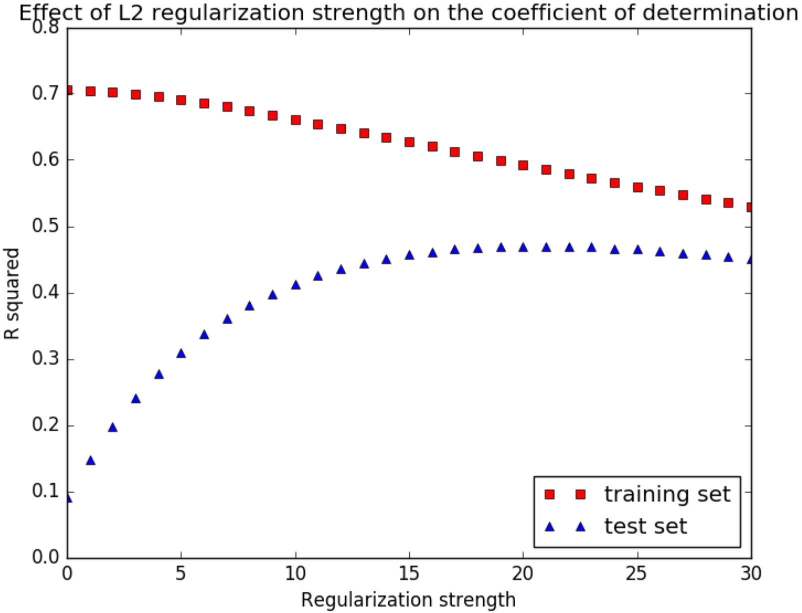
new goal: minimize $J(w) + \lambda \|w\|_2$

```
model = sklearn.linear_model.Ridge(alpha=reg_strength)
model.fit(x_train_1hot, y_train)
r2_train = model.score(x_train_1hot, y_train)
r2_test = model.score(x_test_1hot, y_test)
```

V: Regularization



V: Regularization



V: Regularization

What if we had used L1 regularization?