Lecture 3: Feature engineering

Contents

Imports and LO

Imports

```
import os
import sys

sys.path.append("code/")
import matplotlib.pyplot as plt

import mglearn
import numpy as np
import numpy.random as npr
import pandas as pd
from plotting_functions import *
from sklearn.dummy import DummyClassifier, DummyRegressor
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression, LogisticRegression, Ridge,
from sklearn.metrics import make_scorer, mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score, cross_validate, train_tes
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScale
from sklearn.svm import SVC
```

Learning outcomes

From this lecture, students are expected to be able to:

- Explain the importance of the quality of data.
- Explain the importance of feature engineering when building machine learning models.
- Explain the “column of ones” trick.
- Explain the concept of polynomial feature transformations.
- Apply linear models to non-linear datasets using polynomial features.
- Explain the idea of feature crosses.
- Identify when is it appropriate to apply discretization to numeric features.
- Carry out preliminary feature engineering on text data using `spaCy` and `nltk` (required video).

Lecture slides

Section 1 slides **Section 2 slides**

Data cleaning

Model building is interesting. But in your machine learning projects, you'll be spending more than half of your time on data preparation, feature engineering, and transformations. The *quality* of the data is important. Your model is only as good as your data and we often use the expression garbage in, garbage out to illustrate that no matter how fancy the algorithm is, the data must be of decent quality as well.

Data cleaning often involves fixing mistakes in the data, things that might lead to syntax errors, or just non-sensical information. Example of issues with low quality data that we might want to fix include the following:

- Out of bounds values (e.g. proportions over 1)
- Obscure column names without metadata explanations
- Typos (e.g. spelling differences for the same categorical variable)
- Using multiple values to represent the same value (e.g. NA, NONE, NULL, NaN)

- Duplicate observations
- Incorrect data types
- Untidy format / combined variables in single columns

There are also issues that we can't fix without collecting new data:

- Many missing value (have to impute a lot)
- Few observations overall
- Few observations of some specific groups
- Biased sample that is not representative of the population
- Non-independent observations presented as if they are independent
- Not all observations collected in the same manner (e.g. very far apart in time or by different people)
- Inaccurate measurements
- Proxy measurements
- Fabricated, intentionally modified

❓❓ Questions for you

Select the most accurate option below.

Suppose you are working on a machine learning project. If you have to prioritize one of the following in your project which of the following would it be?

- (A) The quality and size of the data
- (B) Most recent deep neural network model
- (C) Most recent optimization algorithm

▶ Click to see the answers

Feature preprocessing

Usually we distinguish feature pre-processing from features engineering. Preprocessing includes things such as imputations and scaling features, which we have alredy talked about previously. Sometimes these are clumped together, but preprocessing tend to focus on things

you "should" do and without it your model might either run into an error (e.g. not knowing how to handle a missing value) or do something your don't intend it to (e.g. KNN on non-standardized data). Feature engineering can be seen as more optional in the sense that it involves operations that likely will improve your model, but it is not categorically incorrect to not do them. However, this is not a hard distinction and we could refer to cleaning, preprocessing, and engineering alltogether as "feature preprocessing" which you will see e.g. in scikit-learn.

Feature engineering

One of the most important aspects which influences performance of machine learning models is the features used to represent the problem. If your underlying representation is bad whatever fancy model you use is not going to help. With a better feature representation, a simple and a more interpretable model is likely to perform reasonably well.

In 571 we talked about hyperparameter tuning as one way to improve model performance. Another way is by re-engineering the feautres of the input data. Feature engineering is the process of designing new features that we think can be informative to the model, by combining the existing features (or features from alternative data sources) in different ways. Note that this process is often difficult, time-consuming, and requires expert knowledge.

A couple of quote on feature engineering

A quote by Pedro Domingos A Few Useful Things to Know About Machine Learning

> ... At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.

A quote by Andrew Ng, Machine Learning and AI via Brain simulations

> Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

Better features usually help more than a better model.

- Good features would ideally:
  - Capture most important aspects of the problem
  - Allow learning with fewer examples
  - Generalize to new scenarios.
- There is a trade-off between simple and expressive features:
  - With simple features overfitting risk is low, but scores might be low.
  - With complicated features scores can be high, but so is overfitting risk.

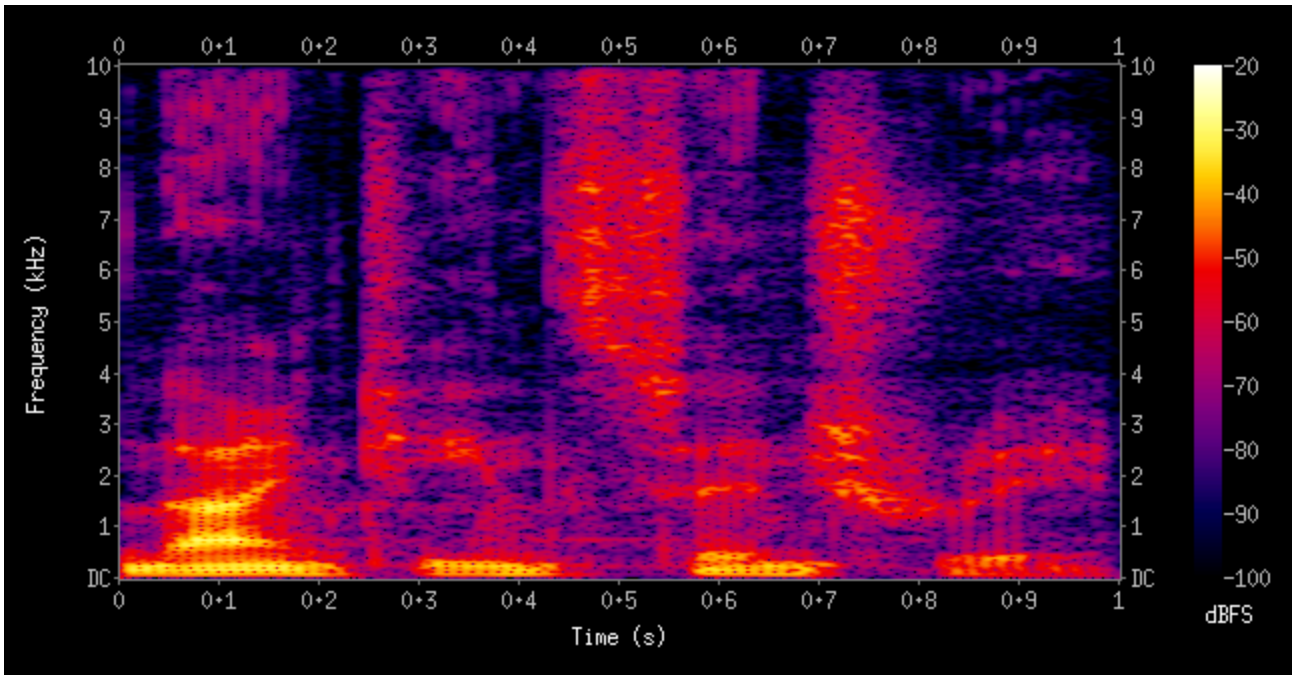
The best features may be dependent on the model you use.

- Examples:
  - For counting-based methods like decision trees separate relevant groups of variable values
    - Discretization makes sense
  - For distance-based methods like KNN, we want different variables to be fairly measured when computing the distances.
    - Standardization
  - For regression-based methods like linear regression, we want targets to have a linear dependency on features.
    - Polynomial feature transformations

Domain-specific transformations

In some domains there are natural transformations to do:

- Spectrograms (sound data)
- Wavelets (image data)
- Convolutions



Source

In this lecture we'll talk about the following:

- Polynomial features (change of basis)
- Feature engineering demos
  - numeric data
  - text data

Polynomial feature transformations to capture non-linear trends

Limitations of linear regression

- Linear models are fast and scalable.
- But they might seem rather limited, especially in low-dimensional spaces because they only learn lines, planes, or hyperplanes.

- What if the true relationship between the target and the features is non-linear?
- Can we still use ordinary least squares to fit non-linear data?
- One way to make linear models more flexible is using feature mappings or transformations!

Polynomial transformations

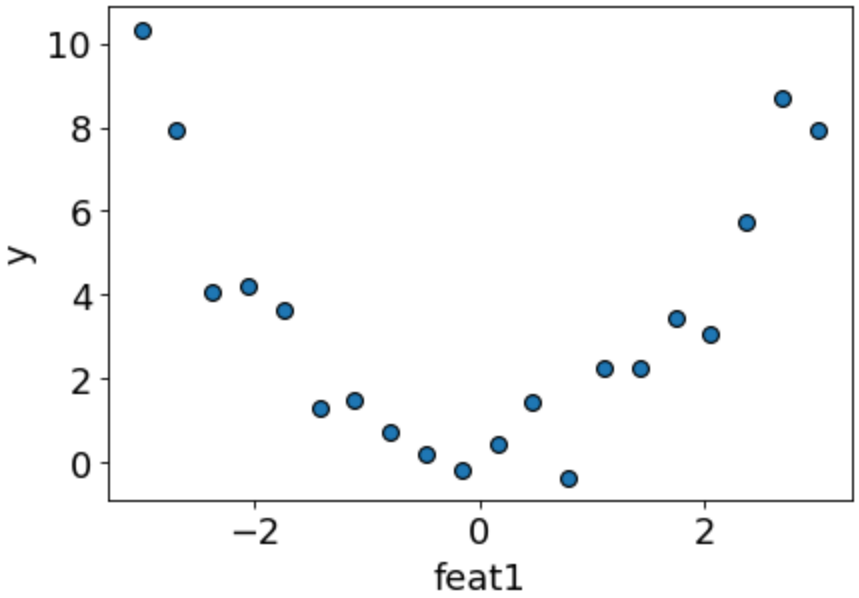
Let's consider this synthetic toy data with only one feature.

```
from matplotlib.pyplot import figure

np.random.seed(10)
n = 20
X = np.linspace(-3, 3, n)
y = X**2 + npr.randn(n)
X_toy = X[:, np.newaxis]
y_toy = y[:, np.newaxis]
pd.DataFrame(np.hstack([X_toy, y_toy]), columns=["feat1", "y"])
```

	feat1	y
0	-3.000000	10.331587
1	-2.684211	7.920265
2	-2.368421	4.064018
3	-2.052632	4.204913
4	-1.736842	3.637956
5	-1.421053	1.299305
6	-1.105263	1.487118
7	-0.789474	0.731817
8	-0.473684	0.228668
9	-0.157895	-0.149669
10	0.157895	0.457957
11	0.473684	1.427414
12	0.789474	-0.341797
13	1.105263	2.249881
14	1.421053	2.248021
15	1.736842	3.461758
16	2.052632	3.076694
17	2.368421	5.744555
18	2.684211	8.689523
19	3.000000	7.920195

Let's plot the data

```
figure(figsize=(6, 4), dpi=80)
plt.scatter(X_toy[:, 0], y_toy, s=50, edgecolors=(0, 0, 0))
plt.xlabel("feat1")
plt.ylabel("y");
```

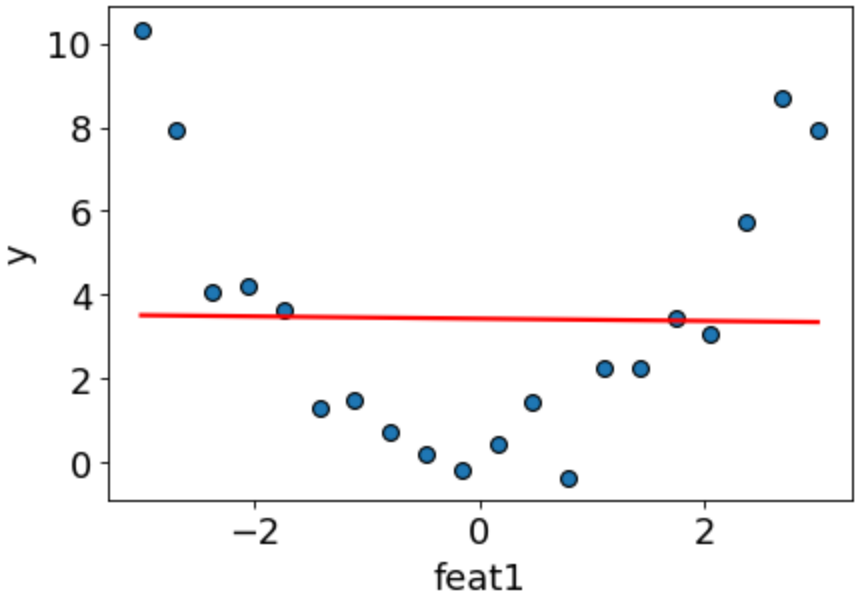


- Can simple linear regression fit this data?
- Let's try it out.
  - Right now we are focussing on creating a better fit on the training data. So we are skipping splitting the data for the demonstration purpose.
  - Also, for simplicity, we are using `LinearRegression`, which doesn't have any hyperparameter controlling the complexity of the model.

```
# Fit a regression line.
lr = LinearRegression()
lr.fit(X_toy, y_toy)
```

```
▾ LinearRegression
LinearRegression()
```

```
figure(figsize=(6, 4), dpi=80)
plt.scatter(X_toy[:, 0], y, s=50, edgecolors=(0, 0, 0))
preds = lr.predict(X_toy)
plt.xlabel("feat1")
plt.ylabel("y")
plt.plot(X_toy, preds, color="red", linewidth=2);
```



- As expected, the regression line is unable to capture the shape of the data.
- The model is underfit.
- The score on the training data is close to a dummy model.

```
lr.score(X_toy, y_toy)
```

```
0.0002572570295679144
```

```
DummyRegressor().fit(X_toy, y_toy).score(X_toy, y_toy)
```

```
0.0
```

Adding quadratic features

- It looks like a quadratic function would be more suitable for this dataset.
- A linear model on its own cannot fit a quadratic function. But what if we **augment the data with a new quadratic feature**?
- Let's try it out.

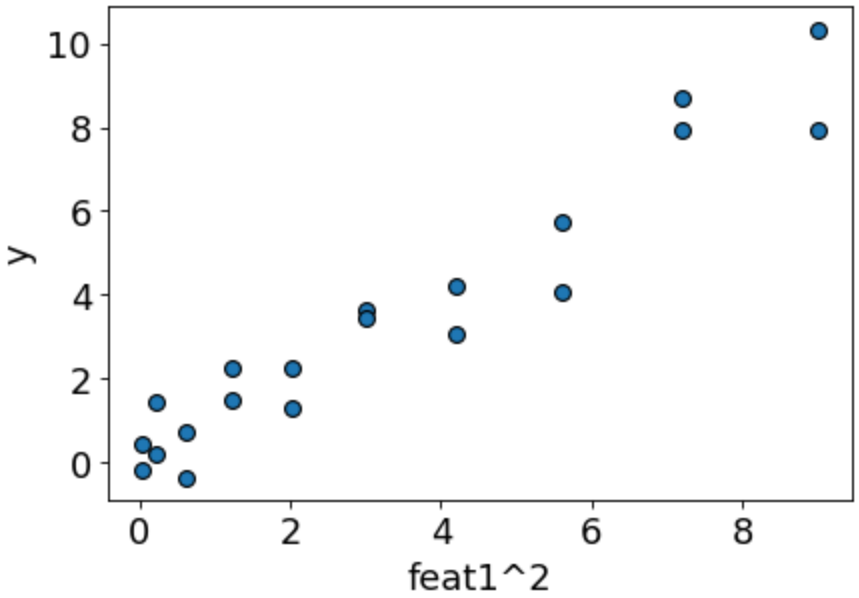
```
# add a squared feature
X_sq = np.hstack([X_toy, X_toy**2])
pd.DataFrame(X_sq, columns=["feat1", "feat1^2"])
```

	feat1	feat1^2
0	-3.000000	9.000000
1	-2.684211	7.204986
2	-2.368421	5.609418
3	-2.052632	4.213296
4	-1.736842	3.016620
5	-1.421053	2.019391
6	-1.105263	1.221607
7	-0.789474	0.623269
8	-0.473684	0.224377
9	-0.157895	0.024931
10	0.157895	0.024931
11	0.473684	0.224377
12	0.789474	0.623269
13	1.105263	1.221607
14	1.421053	2.019391
15	1.736842	3.016620
16	2.052632	4.213296
17	2.368421	5.609418
18	2.684211	7.204986
19	3.000000	9.000000

We can see that there is a linear relationship between this new squared feature and the target.

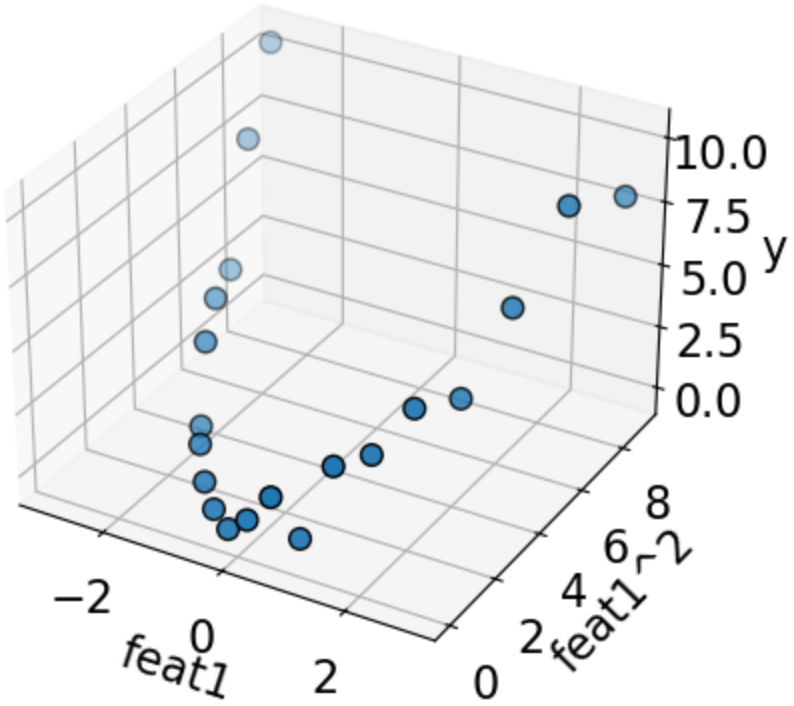
```
figure(figsize=(6, 4), dpi=80)
plt.scatter(X_sq[:, 1], y, s=50, edgecolors=(0, 0, 0))
plt.xlabel("feat1^2")
plt.ylabel("y")
```

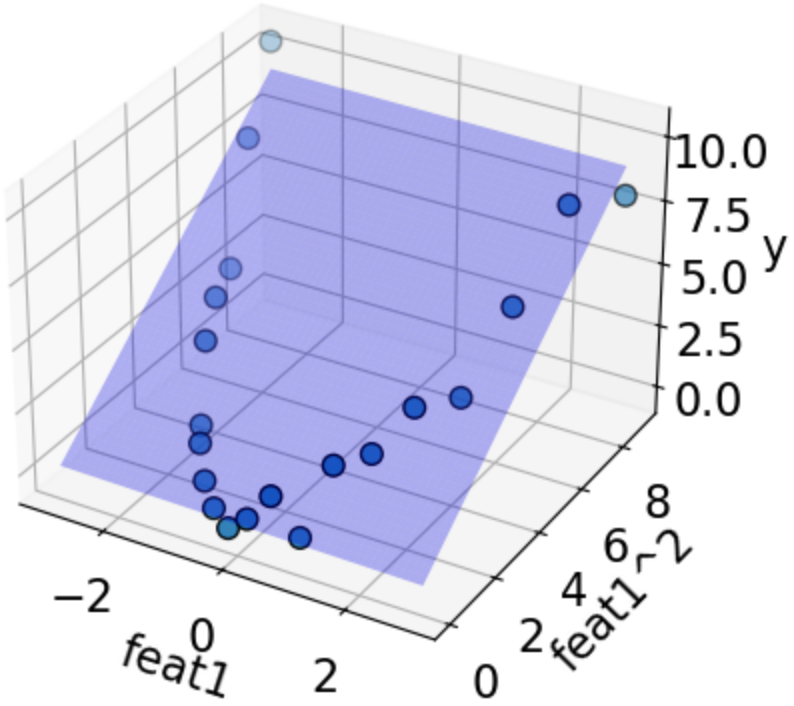
```
Text(0, 0.5, 'y')
```



Now, let's plot both our features along with `y`, and show how a linear model would fit a hyperplane to this data.

```
plot_3d_reg(X_sq, y)  # user-defined function from plotting_functions.py
plot_3d_reg(X_sq, y, surface=True)
```





- A linear model fits well on this augmented data now!!
- This is a common way to make linear models more flexible by adding more degrees of freedom.
- The model is still linear, i.e., it's still learning the coefficients for each feature. But the feature space is augmented now.

```
lr = LinearRegression()
lr.fit(X_sq, y_toy)  # Linear regression with augmented data
lr.score(X_sq, y_toy)  # The scores are much better now
```

```
0.9270602202765702
```

```
pd.DataFrame(
    lr.coef_.transpose(), index=["feat1", "feat1^2"], columns=["Feature coeffi
)
```

	Feature coefficient
feat1	-0.027196
feat1^2	1.006051

- According to the model, our newly created feat1^2 feature is the most important feature for prediction; the coefficient of the squared feature has the biggest magnitude.
- The idea of transforming features and creating new features is referred to as **change of basis**.

Polynomial regression in `sklearn`

- In `sklearn` we can add polynomial features using sklearn's PolynomialFeatures
- Using this we can generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to a specified degree.
- `PolynomialFeatures` is a transformer.
- For one-dimensional feature vector `[a]`, the augmented features with degree 2 polynomial would be: $[1, a, a^2]$.
- For two-dimensional feature vector `[a, b]`, the augmented features with degree 2 polynomial would be: $[1, a, b, a^2, ab, b^2]$.
- The "Column of ones" that is introduced here, is similar to how we can express a linear regression equation in matrix notation. See the section "Column of ones" further down for more detail on this.

Let's try polynomial features with degree 2 and visualize augmented features.

```
from sklearn.preprocessing import PolynomialFeatures

deg = 2
poly_feats = PolynomialFeatures(degree=deg)
X_enc = poly_feats.fit_transform(X_toy)
pd.DataFrame(X_enc, columns=poly_feats.get_feature_names_out()).head()
```

	1	x0	x0^2
0	1.0	-3.000000	9.000000
1	1.0	-2.684211	7.204986
2	1.0	-2.368421	5.609418
3	1.0	-2.052632	4.213296
4	1.0	-1.736842	3.016620

- Let's fit linear regression on the transformed data.

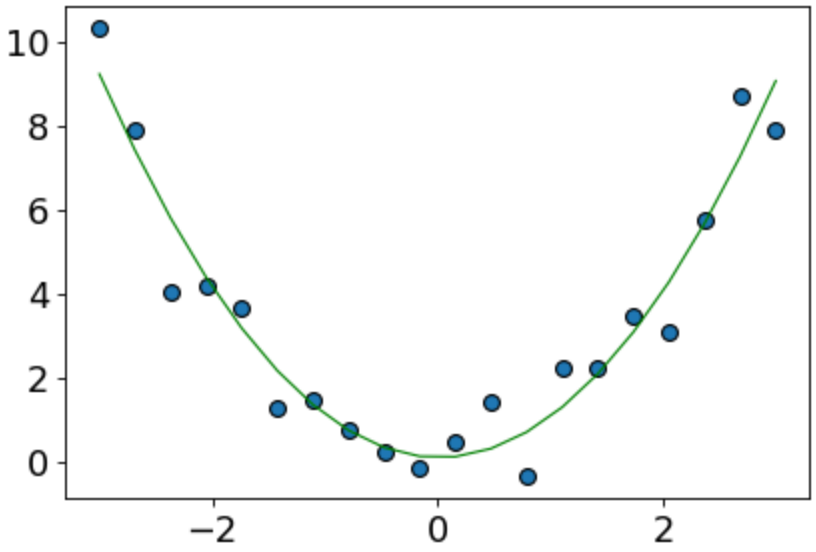
```
lr_poly = LinearRegression()
lr_poly.fit(X_enc, y_toy)
preds = lr_poly.predict(X_enc)
lr_poly.score(X_enc, y_toy)
```

```
0.9270602202765702
```

- The model is not underfit anymore. The training score is pretty good!
- Let's examine the coefficients.

We can also plot the prediction from the augmented 3-D space in the original 2-D space.

```
figure(figsize=(6, 4), dpi=80)
plt.scatter(X_toy[:, 0], y_toy, s=50, edgecolors=(0, 0, 0))
plt.plot(X, preds, color="green", linewidth=1);
```



- Now the fit is much better compared to linear regression on the original data!
- What's happening here?
- The actual linear regression model is fit in the augmented space and making the prediction in that augmented space.

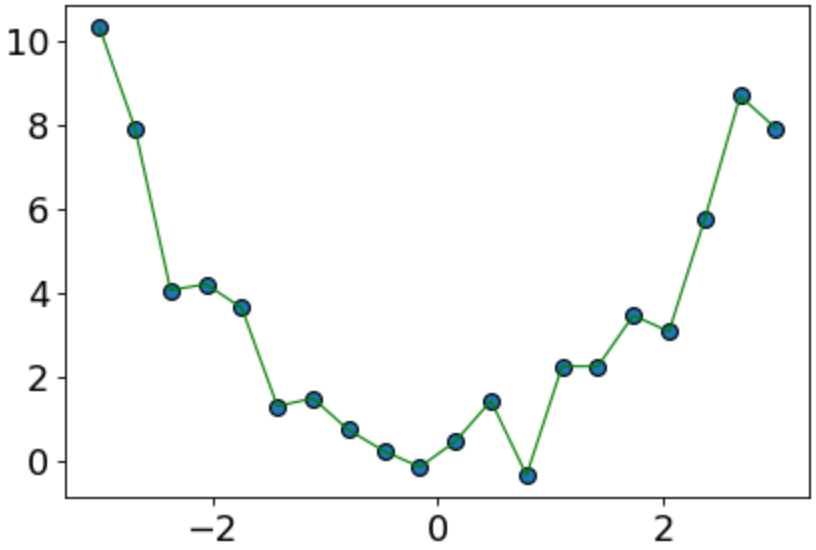
Let's try degree=20 polynomial.

```
deg = 20
poly_feats = PolynomialFeatures(degree=deg)
X_enc = poly_feats.fit_transform(X_toy)
pd.DataFrame(X_enc, columns=poly_feats.get_feature_names_out()).head(10)
```

	1	x0	x0^2	x0^3	x0^4	x0^5	x0^6
0	1.0	-3.000000	9.000000	-27.000000	81.000000	-243.000000	729.000000
1	1.0	-2.684211	7.204986	-19.339700	51.911825	-139.342268	374.023983
2	1.0	-2.368421	5.609418	-13.285464	31.465573	-74.523727	176.503563
3	1.0	-2.052632	4.213296	-8.648345	17.751867	-36.438042	74.793875
4	1.0	-1.736842	3.016620	-5.239393	9.099999	-15.805262	27.451244
5	1.0	-1.421053	2.019391	-2.869660	4.077938	-5.794965	8.234950
6	1.0	-1.105263	1.221607	-1.350197	1.492323	-1.649409	1.823031
7	1.0	-0.789474	0.623269	-0.492054	0.388464	-0.306682	0.242117
8	1.0	-0.473684	0.224377	-0.106284	0.050345	-0.023848	0.011296
9	1.0	-0.157895	0.024931	-0.003936	0.000622	-0.000098	0.000015

10 rows × 21 columns

```
lr_poly = LinearRegression()
lr_poly.fit(X_enc, y)
preds = lr_poly.predict(X_enc)
figure(figsize=(6, 4), dpi=80)
plt.scatter(X_toy[:, 0], y_toy, s=50, edgecolors=(0, 0, 0))
plt.plot(X_toy, preds, color="green", linewidth=1);
```



- It seems like we are overfitting now.
- The model is trying to go through every training point.
- The model is likely to overfit on unseen data.
- You can pick the degree of polynomial using hyperparameter optimization.

Interim summary

- We can make linear models more flexible by augmenting the feature space.
- One way to do it is by applying polynomial transformations.
- Example: Suppose X has only one feature, say $f_1$$

$$X = \begin{bmatrix} 0.86 \\ 0.02 \\ -0.42 \end{bmatrix}$$

- We can add a new feature f_1^2?

- Our Z will have three features f_1^0, f_1^1, f_1^2 with polynomial with degree 2. $
$Z = \begin{bmatrix} 1 & 0.86 & 0.74 \\ 1 & 0.02 & 0.0004 \\ 1 & -0.42 & 0.18 \end{bmatrix}$ $

- $Z \rightarrow$ augmented dataset with quadratic features
- `fit`: We fit using Z and learn weights v.
- `predict`: When we predict, we need to apply the same transformations on the test example and add these features in the test example and predict using learned weights v.

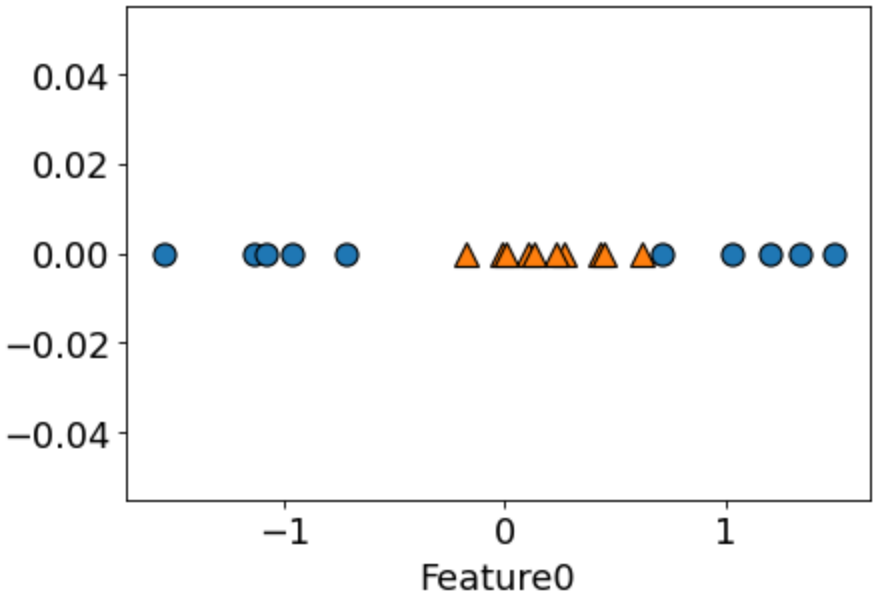
$\hat{y}$ is still a linear function of v and Z.

Classification setting: Non-linearly separable data

Polynomial feature transformations are not only suitable for regression problems, they can also help in classification. Let's consider this non-linearly separable 1-D data.

```
# Consider this one-dimensional classification dataset
n = 20
d = 1
np.random.seed(10)
X = np.random.randn(n, d)
y = np.sum(X**2, axis=1) < 0.4
figure(figsize=(6, 4), dpi=80)
# plt.scatter(X[:, 0], np.zeros_like(X), c=y, s=50, edgecolors=(0, 0, 0));
mglearn.discrete_scatter(X[:, 0], np.zeros_like(X), y)
plt.xlabel("Feature0")
# plt.legend();
```

```
Text(0.5, 0, 'Feature0')
```



Can we use a linear classifier on this dataset?

```
linear_svm = SVC(kernel="linear", C=100)
linear_svm.fit(X, y)
print("Training accuracy", linear_svm.score(X, y))
```

```
Training accuracy 0.75
```

What if we augmented this data with polynomial with degree=2 feature?

```
X[:5]
```

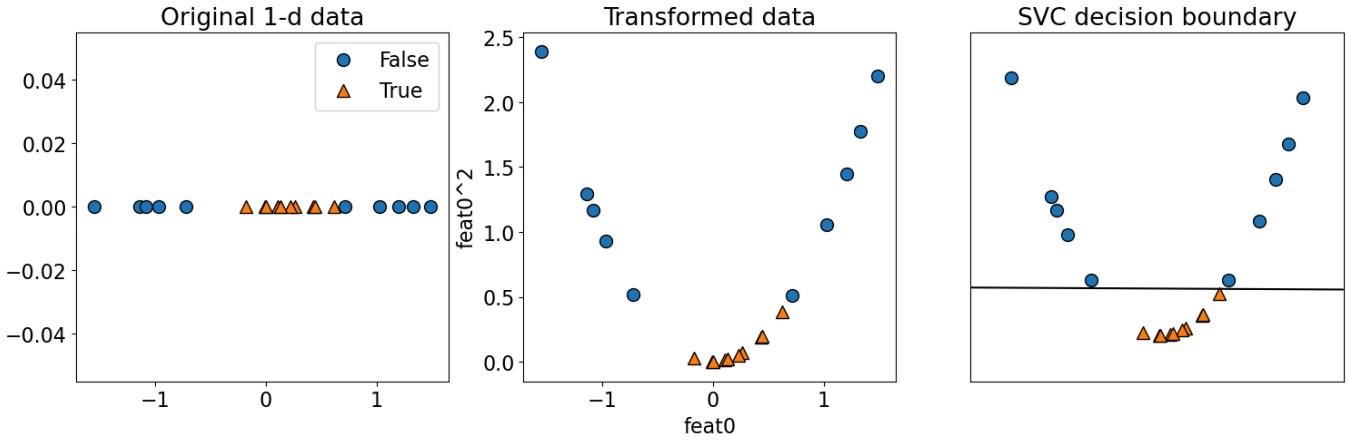
```
array([[ 1.3315865 ],
       [ 0.71527897],
       [-1.54540029],
       [-0.00838385],
       [ 0.62133597]])
```

```
poly = PolynomialFeatures(
    2, include_bias=False
)  # Excluding the bias term for simplicity
X_transformed = poly.fit_transform(X)
X_transformed[0:5]
```

```
array([[ 1.33158650e+00,  1.77312262e+00],
       [ 7.15278974e-01,  5.11624011e-01],
       [-1.54540029e+00,  2.38826206e+00],
       [-8.38384993e-03,  7.02889396e-05],
       [ 6.21335974e-01,  3.86058392e-01]])
```

```
linear_svm = SVC(kernel="linear", C=100)
linear_svm.fit(X_transformed, y)
print("Training accuracy", linear_svm.score(X_transformed, y))
plot_orig_transformed_svc(linear_svm, X, X_transformed, y)
```

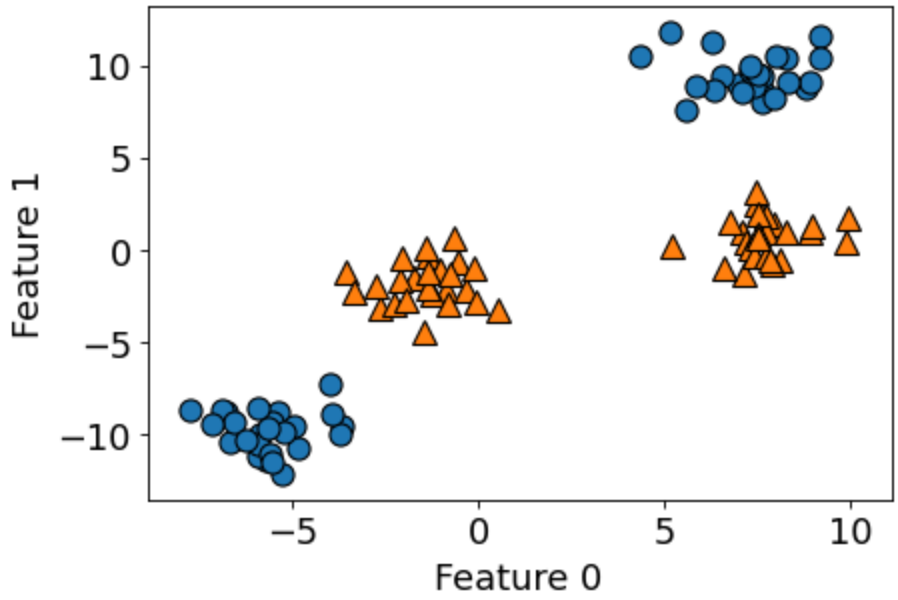
```
Training accuracy 1.0
```



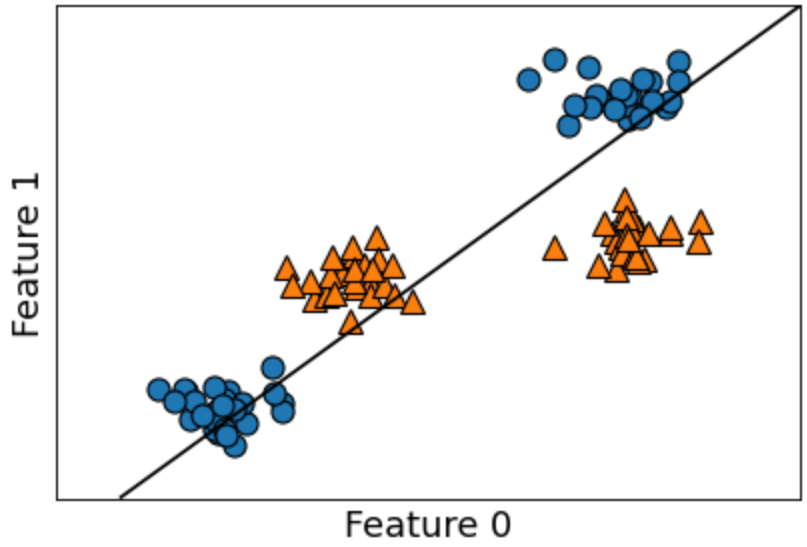
The data is linearly separable in this new feature space!!

(Optional) Another example with two features

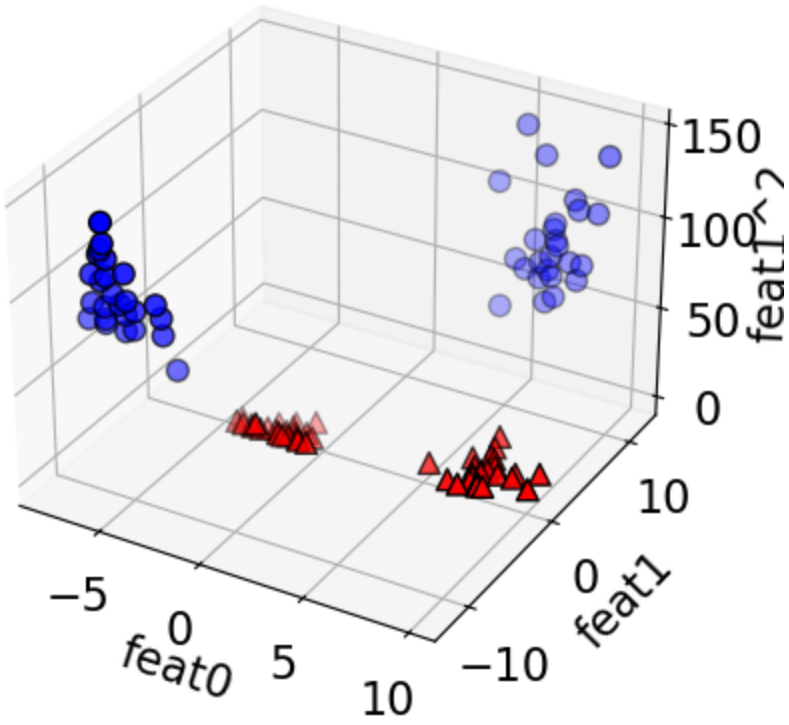
```
import mglearn
from sklearn.datasets import make_blobs

X, y = make_blobs(centers=4, random_state=8)
y = y % 2

figure(figsize=(6, 4), dpi=80)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1");
```



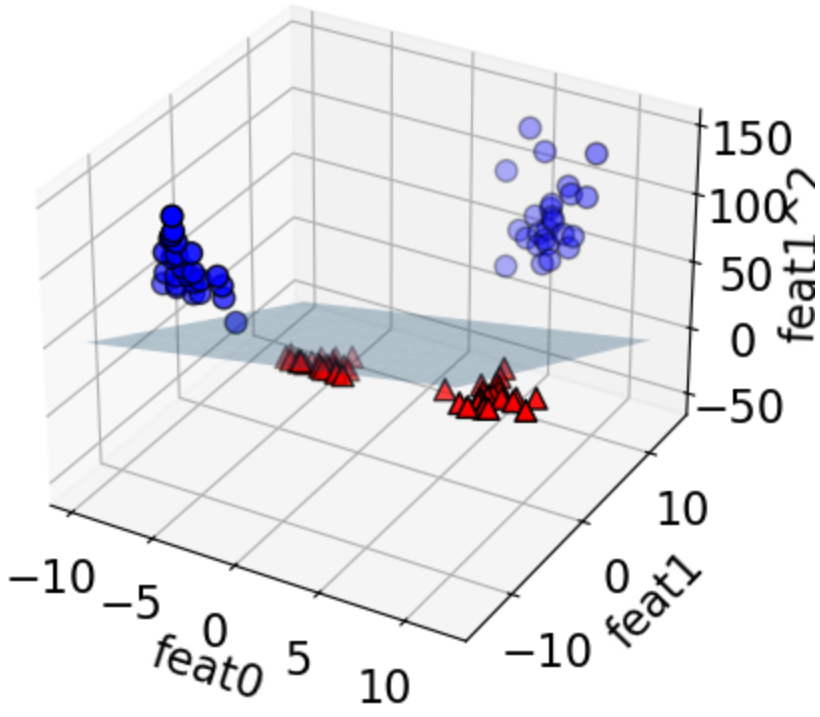

```
from sklearn.svm import LinearSVC

linear_svm = LinearSVC().fit(X, y)
figure(figsize=(6, 4), dpi=80)
mglearn.plots.plot_2d_separator(linear_svm, X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1");
```




```
# add square of the second feature
X_new = np.hstack([X, X[:, 1:] ** 2])
plot_mglearn_3d(X_new, y);
```

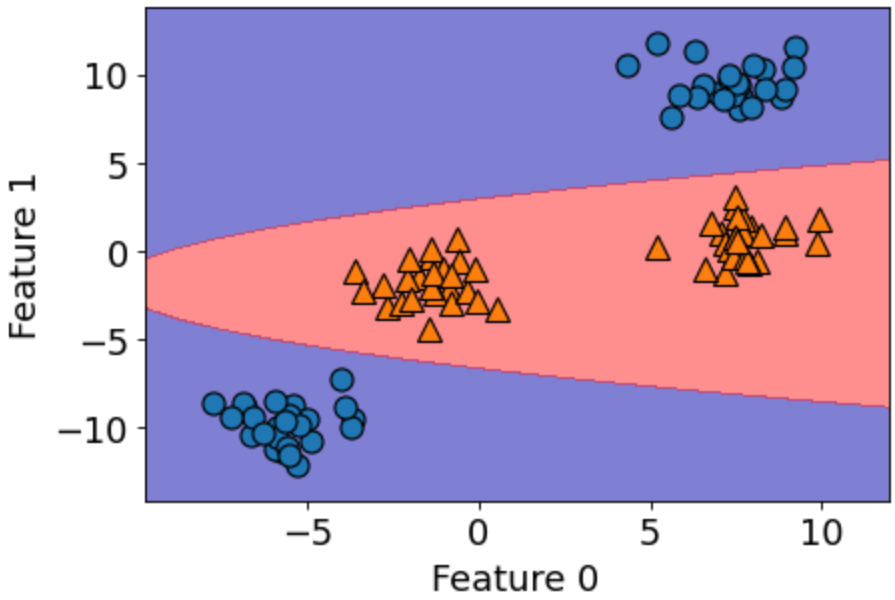



```
linear_svm_3d = LinearSVC().fit(X_new, y)
XX, YY = plot_svc_3d_decision_boundary(X_new, y, linear_svm_3d)
```



What does this linear boundary in Z-space correspond to *in the original (X) space*?

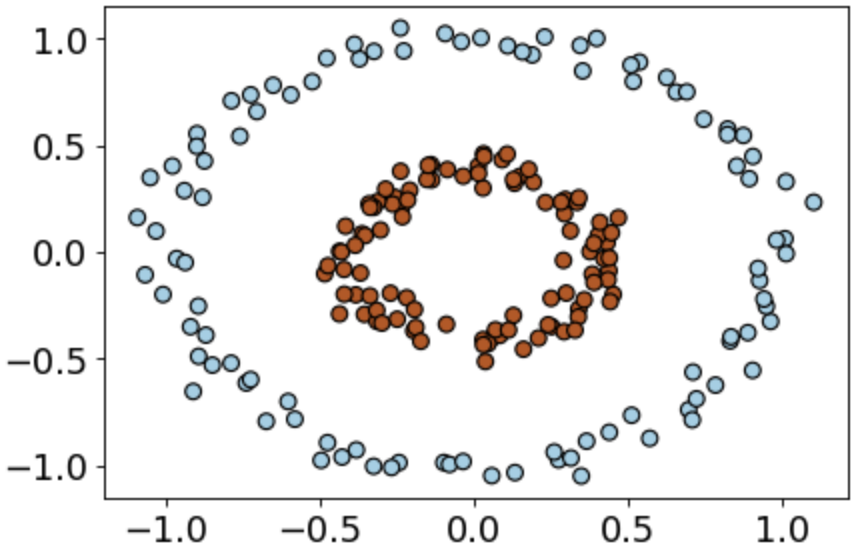
```
figure(figsize=(6, 4), dpi=80)
plot_Z_space_boundary_in_X_space(linear_svm_3d, X, y, XX, YY)
```



It's a parabola!

Another example with non-linearly separable data

```
from sklearn import datasets

figure(figsize=(6, 4), dpi=80)
X, y = datasets.make_circles(n_samples=200, noise=0.06, factor=0.4)
plt.scatter(X[:, 0], X[:, 1], s=50, c=y, cmap=plt.cm.Paired, edgecolors=(0, 0,
```



```
lr_circ = LinearRegression()
lr_circ.fit(X, y).score(X, y)
```

```
0.00015933776258525434
```

```
lr_circ_pipe = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
lr_circ_pipe.fit(X, y).score(X, y)
```

```
0.9669328201796121
```

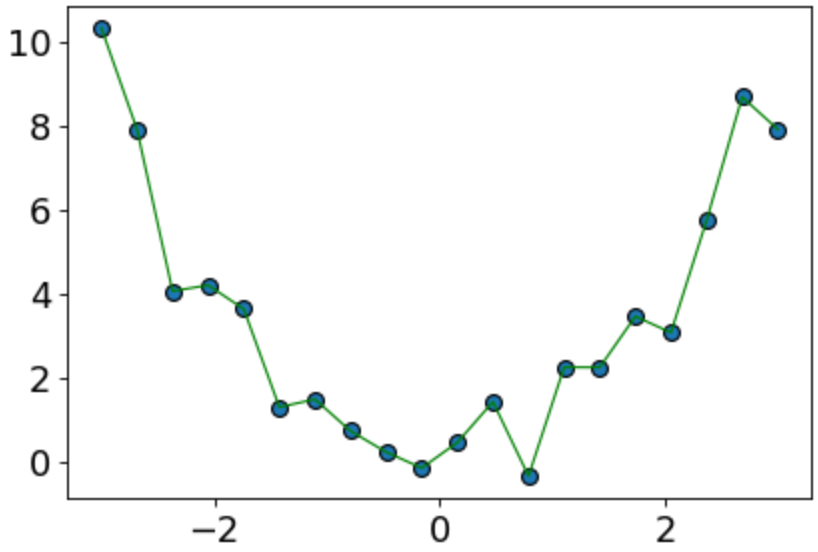
`PolynomialFeatures` with sklearn pipelines

Polynomial feature transformations can be used just as other preprocessing tools in the scikit learn library, so we can include them in pipelines to keep our code tidy.

```
from matplotlib.pyplot import figure

np.random.seed(10)
n = 20
X = np.linspace(-3, 3, n)
y = X**2 + npr.randn(n)
X_toy = X[:, np.newaxis]
y_toy = y[:, np.newaxis]
pd.DataFrame(np.hstack([X_toy, y_toy]), columns=["feat1", "y"])
```

	feat1	y
0	-3.000000	10.331587
1	-2.684211	7.920265
2	-2.368421	4.064018
3	-2.052632	4.204913
4	-1.736842	3.637956
5	-1.421053	1.299305
6	-1.105263	1.487118
7	-0.789474	0.731817
8	-0.473684	0.228668
9	-0.157895	-0.149669
10	0.157895	0.457957
11	0.473684	1.427414
12	0.789474	-0.341797
13	1.105263	2.249881
14	1.421053	2.248021
15	1.736842	3.461758
16	2.052632	3.076694
17	2.368421	5.744555
18	2.684211	8.689523
19	3.000000	7.920195

```
degree = 20
pipe_poly = make_pipeline(PolynomialFeatures(degree=degree), LinearRegression(

pipe_poly.fit(X_toy, y_toy)
preds = pipe_poly.predict(X_toy)
figure(figsize=(6, 4), dpi=80)
plt.scatter(X_toy[:, 0], y_toy, s=50, edgecolors=(0, 0, 0))
plt.plot(X, preds, color="green", linewidth=1);
```



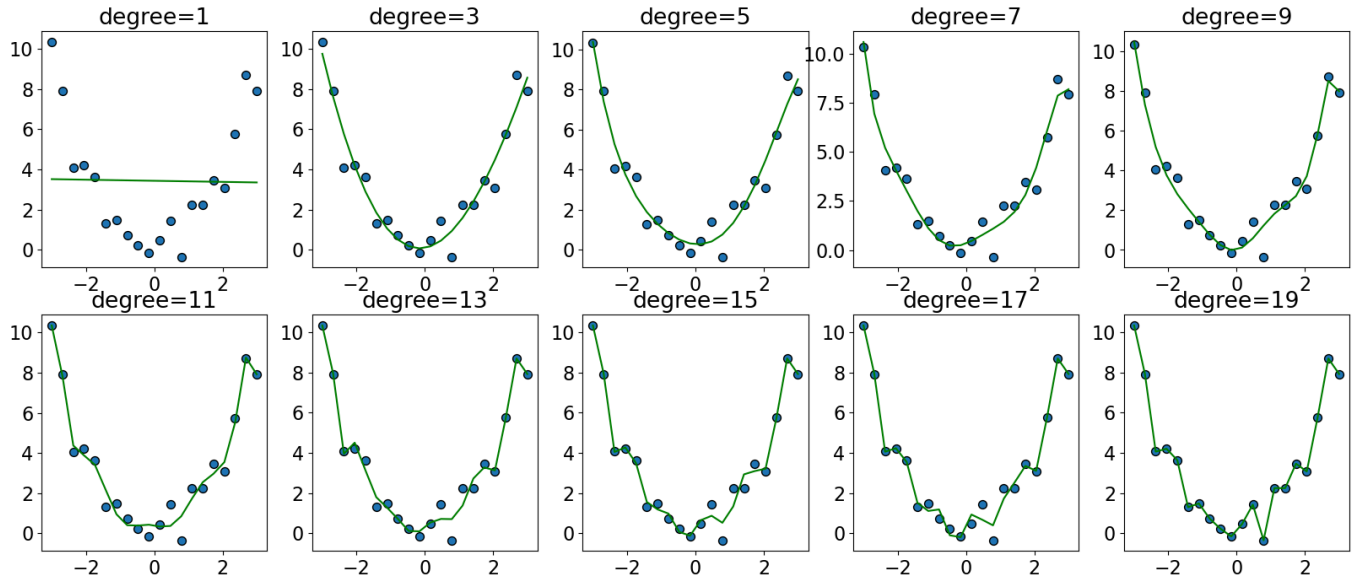
The model has learned coefficients for the transformed features.

```
pd.DataFrame(
    pipe_poly.named_steps["linearregression"].coef_.transpose(),
    index=pipe_poly.named_steps["polynomialfeatures"].get_feature_names_out(),
    columns=["Feature coefficients"],
).sort_values("Feature coefficients", ascending=False)
```

	Feature coefficients
x0^7	84.271533
x0^11	31.398818
x0^6	29.056499
x0^12	16.435514
x0^2	9.626527
x0^8	8.711761
x0^3	3.763840
x0	1.856515
x0^15	1.138977
x0^16	0.755986
x0^19	0.002640
x0^20	0.001922
1	-0.000220
x0^18	-0.060286
x0^17	-0.085945
x0^14	-4.832276
x0^13	-7.941176
x0^10	-26.869921
x0^4	-31.828364
x0^5	-44.211552
x0^9	-70.499855

Let’s explore multiple different degrees and see which model fits our data the best:

```
fig, axes = plt.subplots(2, 5, figsize=(20, 8))

degrees = np.arange(1, 20, 2)

for deg, ax in zip(degrees, axes.ravel()):
    pipe_poly = make_pipeline(PolynomialFeatures(degree=deg), LinearRegression
    pipe_poly.fit(X_toy, y_toy)
    preds = pipe_poly.predict(X_toy)
    ax.scatter(X_toy[:, 0], y_toy, s=50, edgecolors=(0, 0, 0))
    ax.plot(X_toy, preds, color="green", linewidth=1.5)
    title = "degree={}".format(deg)
    ax.set_title(title)
```



How do we know which is the correct degree of polynomial for our data? Remember the eventual goal of feature engineering: we want to introduce features that are informative to the model and makes it generalize better to unseen data. So just as when we decide which values we should use for our hyperparameters we can use cross-validation during a random/grid search to evaluate which degree of polynomial helps our model generalize.

```
from sklearn.model_selection import GridSearchCV

pipe = make_pipeline(
    PolynomialFeatures(),
    LinearRegression()
)

param_grid = {'polynomialfeatures__degree': [1, 2, 3, 4, 5, 19]}
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5, n_jobs=1)
grid.fit(X_toy, y_toy)
pd.DataFrame(grid.cv_results_)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_polyn
0	0.001114	0.000193	0.000579	0.000077	
1	0.001266	0.000231	0.000778	0.000222	
2	0.000977	0.000067	0.000604	0.000057	
3	0.000989	0.000209	0.000555	0.000076	
4	0.000836	0.000059	0.000505	0.000034	
5	0.000889	0.000033	0.000575	0.000101	

Note that finding the right degree of polynomial, doesn't mean that all the features introduced are important. For example, in degree=3, maybe exponentiating the feature on their own didn't add any useful info not already in the interaction terms (or the other way around). At this point, we can't tell, but in the next lecture we will see how to select only the features that are important to the model.

If we are using standardization in our pipeline, we generally want to put this after the polynomial feature engineering to avoid that the polynomial features end up being of much larger absolute magnitude than the original features, which would be an issue for any distance based calculations and for interpretation of coefficients. However, if we create high degree polynomial features from values that are already very large, we can run into issues with numerical representation because the values get so large, so in these cases we might put the standardization first.

How can we find the correct degree more efficiently?

Exploring many high degree polynomials in an optimization search is very computationally expensive as there are a lot of data transformations involved. If d is the original number of features and p be the degree of polynomial, we have roughly $\mathcal{O}(d^p)$ feature combinations. For example, for $d = 1000$, and $p = 3$, we would have around a billion new feature combinations!

A more efficient way to do this is by using a "kernel" with the so called "Kernel trick":

- Computationally efficient approach to map features
- Calculate the relationships between features in higher dimensional space without actually carrying out the transformation for each feature.
- Overall, saying something is a "kernel method" correspond to this idea of implicitly calculating relationships in data in higher dimensional space.
- Then the different kernels/transformations have different names like "polynomial kernel" or "RBF kernel" (you can read more about the RBF kernel in the optional section at the end).
- Two optional videos:
  - A short video on the intuition behind the kernel trick
  - A longer video with details on the math from CPSC 340.

Feature interactions and feature crosses

- A **feature cross** is another word for a feautre interaction: a synthetic feature formed by combining (often multiplying/"crossing") two or more features. (Although sometimes the term "feature cross" is reseverved for combining categorical features).
- We already saw these in polynomial feature transformations and here is a more explicit example:

Is the following dataset (XOR function) linearly separable?

x_1	x_2	target
1	1	0
-1	1	1
1	-1	1
-1	-1	0

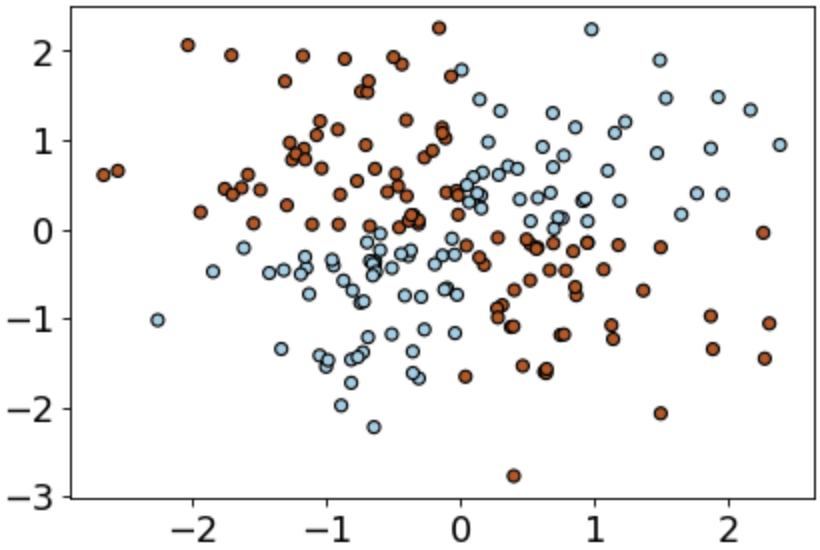
- For XOR like problems, if we create a feature cross $x1x2$, the data becomes linearly separable.

x_1	x_2	x_1x_2	target
1	1	1	0
-1	1	-1	1
1	-1	-1	1
-1	-1	1	0

Let's look at an example with more data points.

```
xx, yy = np.meshgrid(np.linspace(-3, 3, 50), np.linspace(-3, 3, 50))
rng = np.random.RandomState(0)
X_xor = rng.randn(200, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0, X_xor[:, 1] > 0)
```

```
figure(figsize=(6, 4), dpi=80)
plt.scatter(
    X_xor[:, 0], X_xor[:, 1], s=30, c=y_xor, cmap=plt.cm.Paired, edgecolors=(0
);
```




```
LogisticRegression().fit(X_xor, y_xor).score(X_xor, y_xor)
```

```
0.535
```

```
pipe_xor = make_pipeline(
    PolynomialFeatures(interaction_only=True), LogisticRegression()
)
pipe_xor.fit(X_xor, y_xor)
pipe_xor.score(X_xor, y_xor)
```

```
0.995
```

```
feature_names = (
    pipe_xor.named_steps["polynomialfeatures"].get_feature_names_out().tolist(
)
```

```
pd.DataFrame(
    pipe_xor.named_steps["logisticregression"].coef_.transpose(),
    index=feature_names,
    columns=["Feature coefficient"],
)
```

	Feature coefficient
1	-0.001853
x0	-0.028367
x1	0.130500
x0 x1	-5.085984

The interaction feature has the biggest coefficient!

Feature crosses for one-hot encoded features

- You can think of feature crosses of one-hot-features as logical conjunctions
- Suppose you want to predict whether you will find parking or not based on two features:
  - area (possible categories: UBC campus and Rogers Arena)
  - time of the day (possible categories: 9am and 7pm)
- A feature cross in this case would create four new features:

    - UBC campus and 9am
    - UBC campus and 7pm
    - Rogers Arena and 9am
    - Rogers Arena and 7pm.
- The features UBC campus and 9am on their own are not that informative but the newly created feature UBC campus and 9am or Rogers Arena and 7pm would be quite informative.
- Coming up with the right combination of features requires some domain knowledge or careful examination of the data.
- There is no easy way to support feature crosses in sklearn.

Demo of feature engineering with numeric features

- Remember the California housing dataset we used in DSCI 571?
- The prediction task is predicting `median_house_value` for a given property.

```
housing_df = pd.read_csv("data/housing.csv")
housing_df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	popu
0	-122.23	37.88	41.0	880.0	129.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2
2	-122.24	37.85	52.0	1467.0	190.0	
3	-122.25	37.85	52.0	1274.0	235.0	
4	-122.25	37.85	52.0	1627.0	280.0	

```
housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Suppose we decide to train `ridge` model on this dataset.

- What would happen if you train a model without applying any transformation on the categorical features ocean_proximity?
  - Error!! A linear model requires all features in a numeric form.
- What would happen if we apply OHE on `ocean_proximity` but we do not scale the features?
  - No syntax error. But the model results are likely to be poor.
- Do we need to apply any other transformations on this data?

In this section, we will look into some common ways to do feature engineering for numeric or categorical features.

```
train_df, test_df = train_test_split(housing_df, test_size=0.2, random_state=1
```

We have total rooms and the number of households in the neighbourhood. How about creating rooms_per_household feature using this information?

```
train_df = train_df.assign(
    rooms_per_household=train_df["total_rooms"] / train_df["households"]
)
test_df = test_df.assign(
    rooms_per_household=test_df["total_rooms"] / test_df["households"]
)
```

```
train_df
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
9950	-122.33	38.38	28.0	1020.0	169.0
3547	-118.60	34.26	18.0	6154.0	1070.0
4448	-118.21	34.07	47.0	1346.0	383.0
6984	-118.02	33.96	36.0	2071.0	398.0
4432	-118.20	34.08	49.0	1320.0	309.0
...	...	...	...	...	...
7763	-118.10	33.91	36.0	726.0	NaN
15377	-117.24	33.37	14.0	4687.0	793.0
17730	-121.76	37.33	5.0	4153.0	719.0
15725	-122.44	37.78	44.0	1545.0	334.0
19966	-119.08	36.21	20.0	1911.0	389.0

16512 rows × 11 columns

Let's start simple. Imagine that we only three features: `longitude`, `latitude`, and our newly created `rooms_per_household` feature.

```
X_train_housing = train_df[["latitude", "longitude", "rooms_per_household"]]
y_train_housing = train_df["median_house_value"]
```

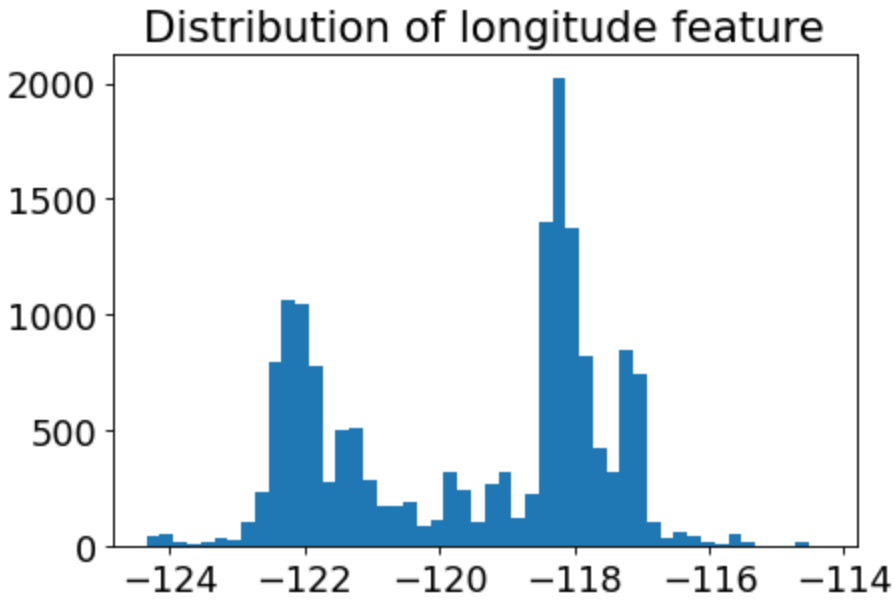
```
from sklearn.compose import make_column_transformer

numeric_feats = ["latitude", "longitude", "rooms_per_household"]

preprocessor1 = make_column_transformer(
    (make_pipeline(SimpleImputer(), StandardScaler()), numeric_feats)
)
```

```
lr_1 = make_pipeline(preprocessor1, Ridge())
pd.DataFrame(
    cross_validate(lr_1, X_train_housing, y_train_housing, return_train_score=
)
```

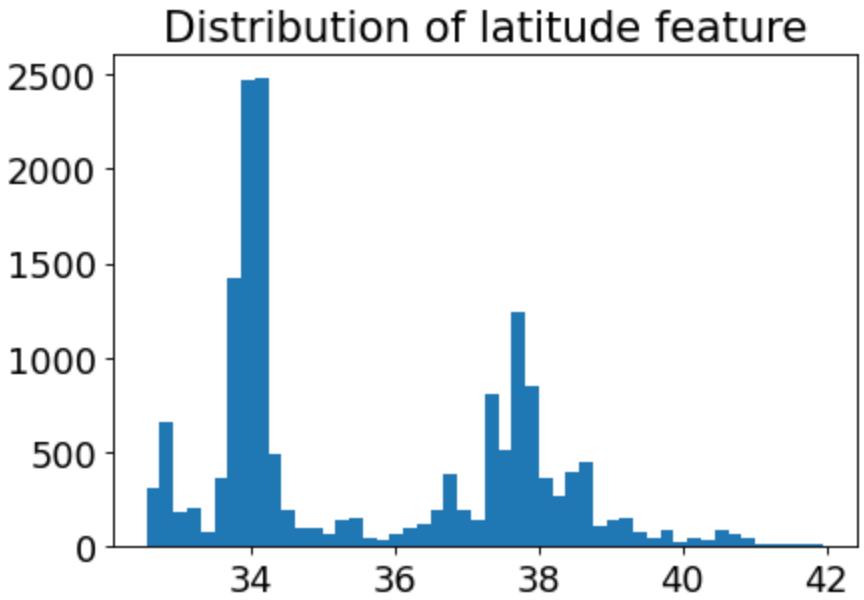
	fit_time	score_time	test_score	train_score
0	0.009066	0.003013	0.280028	0.311769
1	0.006257	0.002579	0.325319	0.300464
2	0.005699	0.002414	0.317277	0.301952
3	0.005697	0.002337	0.316798	0.303004
4	0.005678	0.002334	0.260258	0.314840

- The scores are not great.
- Let's look at the distribution of the longitude and latitude features.

```
figure(figsize=(6, 4), dpi=80)
plt.hist(train_df["longitude"], bins=50)
plt.title("Distribution of longitude feature");
```




```
figure(figsize=(6, 4), dpi=80)
plt.hist(train_df["latitude"], bins=50)
plt.title("Distribution of latitude feature");
```



- Suppose you are planning to build a linear model for housing price prediction.
- If we think longitude is a good feature for prediction, does it makes sense to use the floating point representation of this feature that's given to us?
- Remember that linear models can capture only linear relationships.

- How about discretizing latitude and longitude features and putting them into buckets?
- This process of transforming numeric features into categorical features is called bucketing or binning.
- In `sklearn` you can do this using `KBinsDiscretizer` transformer.
- Let's examine whether we get better results with binning.

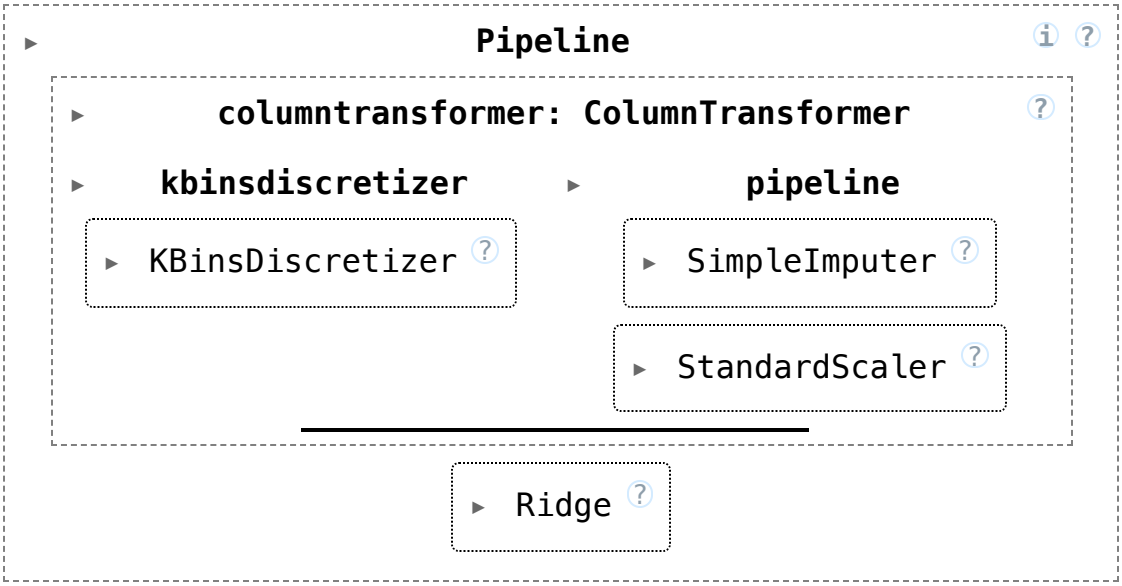
```
from sklearn.preprocessing import KBinsDiscretizer

discretization_feats = ["latitude", "longitude"]
numeric_feats = ["rooms_per_household"]

preprocessor2 = make_column_transformer(
    (KBinsDiscretizer(n_bins=20, encode="onehot"), discretization_feats),
    (make_pipeline(SimpleImputer(), StandardScaler()), numeric_feats),
)
```

```
lr_2 = make_pipeline(preprocessor2, Ridge())
pd.DataFrame(
    cross_validate(lr_2, X_train_housing, y_train_housing, return_train_score=
)
```

	fit_time	score_time	test_score	train_score
0	0.030428	0.009134	0.441445	0.456419
1	0.026731	0.006474	0.469571	0.446216
2	0.029451	0.005945	0.479132	0.446869
3	0.019656	0.005778	0.450822	0.453367
4	0.021519	0.012547	0.388169	0.467628

The results are better with binned features. Let's examine how do these binned features look like.

```
lr_2.fit(X_train_housing, y_train_housing)
```




```
pd.DataFrame(
    preprocessor2.fit_transform(X_train_housing).todense(),
    columns=preprocessor2.get_feature_names_out(),
)
```

	kbinsdiscretizer__latitude_0.0	kbinsdiscretizer__latitude_1.0	kbinsdiscreti
0	0.0	0.0	
1	0.0	0.0	
2	0.0	0.0	
3	0.0	0.0	
4	0.0	0.0	
...	...	...	
16507	0.0	0.0	
16508	0.0	1.0	
16509	0.0	0.0	
16510	0.0	0.0	
16511	0.0	0.0	

16512 rows × 41 columns

How about discretizing all three features?

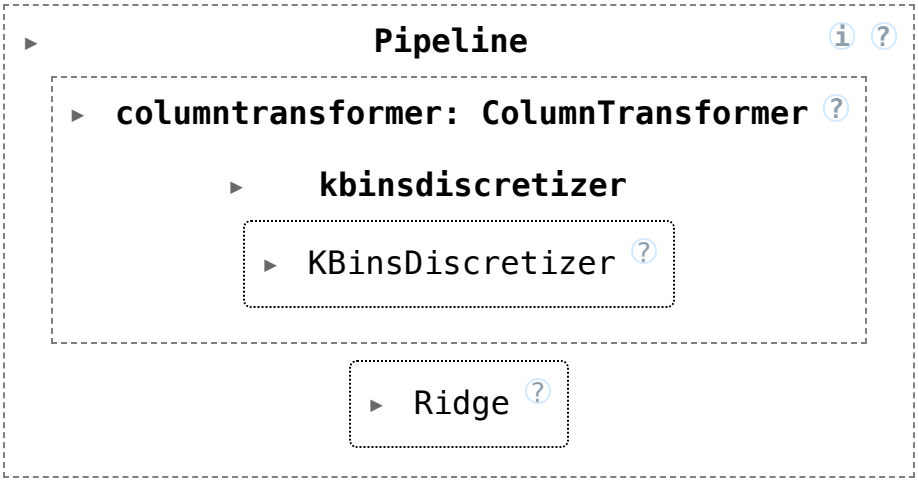
```
from sklearn.preprocessing import KBinsDiscretizer

discretization_feats = ["latitude", "longitude", "rooms_per_household"]

preprocessor3 = make_column_transformer(
    (KBinsDiscretizer(n_bins=20, encode="onehot"), discretization_feats),
)
```

```
lr_3 = make_pipeline(preprocessor3, Ridge())
pd.DataFrame(
    cross_validate(lr_3, X_train_housing, y_train_housing, return_train_score=
)
```

	fit_time	score_time	test_score	train_score
0	0.015876	0.004724	0.590618	0.571969
1	0.015245	0.003892	0.575907	0.570473
2	0.020790	0.004973	0.579091	0.573542
3	0.016910	0.004020	0.571500	0.574260
4	0.016168	0.003963	0.541488	0.581687

- The results have improved further!!
- Let's examine the coefficients

```
lr_3.fit(X_train_housing, y_train_housing)
```




```
feature_names = (
    lr_3.named_steps["columntransformer"]
    .named_transformers_["kbinsdiscretizer"]
    .get_feature_names_out()
)
```

```
lr_3.named_steps["ridge"].coef_.shape
```

```
(60,)
```

```
coefs_df = pd.DataFrame(
    lr_3.named_steps["ridge"].coef_.transpose(),
    index=feature_names,
    columns=["coefficient"],
).sort_values("coefficient", ascending=False)
coefs_df.head
```

```
<bound method NDFrame.head of                              coefficient
longitude_1.0              211343.036136
latitude_1.0               205059.296601
latitude_0.0               201862.534342
longitude_0.0              190319.721818
longitude_2.0              160282.191204
longitude_3.0              157234.920305
latitude_2.0               154105.963689
rooms_per_household_19.0   138503.477291
latitude_8.0               135299.516394
longitude_4.0              132292.924485
latitude_7.0               124982.236174
latitude_3.0               118563.786115
longitude_5.0              116145.526596
rooms_per_household_18.0   102044.252042
longitude_6.0               96554.525554
latitude_4.0                92809.389349
latitude_6.0                90982.951669
latitude_9.0                71096.652487
rooms_per_household_17.0    70472.564483
latitude_5.0                69411.023366
longitude_10.0              52398.892961
rooms_per_household_16.0    44311.362553
rooms_per_household_15.0    31454.877046
longitude_7.0               25658.862997
latitude_10.0               20311.784573
rooms_per_household_14.0    16460.273962
rooms_per_household_13.0     9351.210272
longitude_8.0                6322.652986
rooms_per_household_12.0     1858.970683
rooms_per_household_11.0   -12178.614567
longitude_9.0              -14579.657675
rooms_per_household_10.0   -16630.535622
rooms_per_household_9.0    -19591.810098
longitude_11.0             -22741.635200
rooms_per_household_8.0    -26919.381190
rooms_per_household_7.0    -30573.540359
rooms_per_household_6.0    -32734.570739
rooms_per_household_4.0    -40689.197649
rooms_per_household_3.0    -42060.071975
rooms_per_household_5.0    -43445.134061
rooms_per_household_2.0    -47606.596151
rooms_per_household_0.0    -50884.444297
rooms_per_household_1.0    -51143.091625
latitude_13.0              -57510.779271
longitude_14.0             -70978.802502
longitude_13.0             -89270.957075
longitude_12.0             -90669.093228
latitude_11.0             -100275.316426
longitude_15.0            -105080.071654
latitude_12.0             -111438.823543
latitude_14.0             -114836.305674
latitude_15.0             -116443.256437
longitude_16.0            -119570.316230
latitude_16.0             -140185.299164
```

```
longitude_17.0          -174766.515848
latitude_18.0           -185868.754874
latitude_17.0           -195564.951574
longitude_18.0          -205144.956966
longitude_19.0          -255751.248664
latitude_19.0           -262361.647796>
```

- Does it make sense to take feature crosses in this context?
- What information would they encode?

Feature engineering for text data (video)

- Feature engineering is very relevant in the context of "structured" data such as text data or image data.
- We can extract important information using human knowledge and incorporate it into our models.
- So it's hard to talk about general methods for feature engineering.
- In the lab you'll be carrying out feature engineering on text data.
- Let's look at an example of feature engineering for text data.

We will be using Covid tweets dataset for this.

```
df = pd.read_csv("data/Corona_NLP_test.csv")
df["Sentiment"].value_counts()
```

```
Sentiment
Negative              1041
Positive               947
Neutral                619
Extremely Positive     599
Extremely Negative     592
Name: count, dtype: int64
```

```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=123)
```

```
train_df
```

	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentim
1927	1928	46880	Seattle, WA	13-03-2020	While I don't like all of Amazon's choices, to...	Pos
1068	1069	46021	NaN	13-03-2020	Me: shit buckets, it s time to do the weekly s...	Nega
803	804	45756	The Outer Limits	12-03-2020	@SecPompeo @realDonaldTrump You mean the plan ...	Neu
2846	2847	47799	Flagstaff, AZ	15-03-2020	@lauvagrande People who are sick aren t panic ...	Extrer Nega
3768	3769	48721	Montreal, Canada	16-03-2020	Coronavirus Panic: Toilet Paper Is the People...	Nega
...	...	...	...	...	...	
1122	1123	46075	NaN	13-03-2020	Photos of our local grocery store shelves wher...	Extrer Pos
1346	1347	46299	Toronto	13-03-2020	Just went to the the grocery store (Highland F...	Pos
3454	3455	48407	Houston, TX	16-03-2020	Real talk though. Am I the only one spending h...	Neu
3437	3438	48390	Washington, DC	16-03-2020	The supermarket business is booming! #COVID2019	Neu
3582	3583	48535	St James' Park, Newcastle	16-03-2020	Evening All Here s the story on the and the im...	Pos

3038 rows × 6 columns

```
train_df.columns
```

```
Index(['UserName', 'ScreenName', 'Location', 'TweetAt', 'OriginalTweet',
       'Sentiment'],
      dtype='object')
```

```
train_df["Location"].value_counts()
```

```
Location
United States                     63
London, England                   37
Los Angeles, CA                   30
New York, NY                      29
Washington, DC                    29
                                  ..
Suburb of Chicago                  1
philippines                        1
Dont ask for freedom, take it.     1
Windsor Heights, IA                1
St James' Park, Newcastle          1
Name: count, Length: 1441, dtype: int64
```

```
X_train, y_train = train_df[["OriginalTweet"]], train_df["Sentiment"]
X_test, y_test = test_df[["OriginalTweet"]], test_df["Sentiment"]
```

```
y_train.value_counts()
```

```
Sentiment
Negative              852
Positive              743
Neutral               501
Extremely Negative    472
Extremely Positive    470
Name: count, dtype: int64
```

```
scoring_metrics = "accuracy"
```

```
results = {}
```

```
def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation

    Parameters
    ----------
    model :
        scikit-learn model
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data

    Returns
    ----------
        pandas Series with mean scores from cross_validation
    """

    scores = cross_validate(model, X_train, y_train, **kwargs)

    mean_scores = pd.DataFrame(scores).mean()
    std_scores = pd.DataFrame(scores).std()
    out_col = []

    for i in range(len(mean_scores)):
        out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])

    return pd.Series(data=out_col, index=mean_scores.index)
```

Dummy classifier

```
from sklearn.dummy import DummyClassifier

dummy = DummyClassifier()
results["dummy"] = mean_std_cross_val_scores(
    dummy, X_train, y_train, return_train_score=True, scoring=scoring_metrics
)
pd.DataFrame(results).T
```

```
/tmp/ipykernel_546162/4158382658.py:26: FutureWarning: Series.__getitem__ trea
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

	fit_time	score_time	test_score	train_score
dummy	0.002 (+/- 0.001)	0.001 (+/- 0.001)	0.280 (+/- 0.001)	0.280 (+/- 0.000)

Bag-of-words model

```
from sklearn.feature_extraction.text import CountVectorizer

pipe = make_pipeline(
    CountVectorizer(stop_words="english"), LogisticRegression(max_iter=1000)
)
results["logistic regression"] = mean_std_cross_val_scores(
    pipe,
    X_train["OriginalTweet"],
    y_train,
    return_train_score=True,
    scoring=scoring_metrics,
)
pd.DataFrame(results).T
```

```
/tmp/ipykernel_546162/4158382658.py:26: FutureWarning: Series.__getitem__ trea
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

	fit_time	score_time	test_score	train_score
dummy	0.002 (+/- 0.001)	0.001 (+/- 0.001)	0.280 (+/- 0.001)	0.280 (+/- 0.000)
logistic regression	0.612 (+/- 0.161)	0.022 (+/- 0.002)	0.414 (+/- 0.012)	0.999 (+/- 0.000)

Is it possible to further improve the scores?

- How about adding new features based on our intuitions? Let's extract our own features that might be useful for this prediction task. In other words, let's carry out **feature engineering**.
- The code below adds some very basic length-related and sentiment features. We will be using a popular library called `nltk` for this exercise. If you have successfully created the course `conda` environment on your machine, you should already have this package in the environment.
- How do we extract interesting information from text?

- We use **pre-trained models**!

```
import nltk

nltk.download("vader_lexicon")
nltk.download("punkt")
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /home/joel/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package punkt to /home/joel/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
True
```

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()
```

```
s = "MDS students are smart, sweet, and funny."
print(sid.polarity_scores(s))
```

```
{'neg': 0.0, 'neu': 0.317, 'pos': 0.683, 'compound': 0.8225}
```

```
s = "MDS students are tired because of all the hard work they have been doing.
print(sid.polarity_scores(s))
```

```
{'neg': 0.264, 'neu': 0.736, 'pos': 0.0, 'compound': -0.5106}
```

spaCy

A useful package for text processing and feature extraction

- Active development: https://github.com/explosion/spaCy
- Interactive lessons by Ines Montani: https://course.spacy.io/en/

- Good documentation, easy to use, and customizable.

```
# Download the language model which contains all the pre-trained models
import spacy
spacy.cli.download('en_core_web_md')

import en_core_web_md  # pre-trained model

nlp = en_core_web_md.load()
# if you got an error, try `spacy.cli.download('en_core_web_md')`
```

```
Collecting en-core-web-md==3.7.1
  Downloading https://github.com/explosion/spacy-models/releases/download/en_c
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.8/42.8 MB 3.9 MB/s eta 0:00:0
?25hRequirement already satisfied: spacy<3.8.0,>=3.7.2 in /home/joel/miniconda
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /home/joel/minic
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /home/joel/minic
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /home/joel/minicon
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /home/joel/miniconda3/en
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /home/joel/miniconda3/
Requirement already satisfied: thinc<8.3.0,>=8.1.8 in /home/joel/miniconda3/en
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /home/joel/miniconda3/e
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /home/joel/miniconda3/en
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /home/joel/miniconda
Requirement already satisfied: weasel<0.4.0,>=0.1.0 in /home/joel/miniconda3/e
Requirement already satisfied: typer<0.10.0,>=0.3.0 in /home/joel/miniconda3/e
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /home/joel/miniconda
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /home/joel/miniconda3/en
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /home/joel/miniconda
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /home/j
Requirement already satisfied: jinja2 in /home/joel/miniconda3/envs/573/lib/py
Requirement already satisfied: setuptools in /home/joel/miniconda3/envs/573/li
Requirement already satisfied: packaging>=20.0 in /home/joel/miniconda3/envs/5
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /home/joel/miniconda
Requirement already satisfied: numpy>=1.19.0 in /home/joel/miniconda3/envs/573,
Requirement already satisfied: language-data>=1.2 in /home/joel/miniconda3/env
Requirement already satisfied: annotated-types>=0.6.0 in /home/joel/miniconda3,
Requirement already satisfied: pydantic-core==2.23.4 in /home/joel/miniconda3/
Requirement already satisfied: typing-extensions>=4.6.1 in /home/joel/miniconda
Requirement already satisfied: charset-normalizer<4,>=2 in /home/joel/miniconda
Requirement already satisfied: idna<4,>=2.5 in /home/joel/miniconda3/envs/573/
Requirement already satisfied: urllib3<3,>=1.21.1 in /home/joel/miniconda3/env
Requirement already satisfied: certifi>=2017.4.17 in /home/joel/miniconda3/env
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /home/joel/miniconda3/env
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /home/joel/miniconda
Requirement already satisfied: click<9.0.0,>=7.1.1 in /home/joel/miniconda3/en
Requirement already satisfied: cloudpathlib<0.17.0,>=0.7.0 in /home/joel/minic
Requirement already satisfied: MarkupSafe>=2.0 in /home/joel/miniconda3/envs/5
Requirement already satisfied: marisa-trie>=0.7.7 in /home/joel/miniconda3/env
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_md')
```

```
sample_text = """Dolly Parton is a gift to us all.
From writing all-time great songs like “Jolene” and “I Will Always Love You”,
to great performances in films like 9 to 5, to helping fund a COVID-19 vaccine
she’s given us so much. Now, Netflix bring us Dolly Parton’s Christmas on the
an original musical that stars Christine Baranski as a Scrooge-like landowner
who threatens to evict an entire town on Christmas Eve to make room for a new
Directed and choreographed by the legendary Debbie Allen and counting Jennifer
and Parton herself amongst its cast, Christmas on the Square seems like the pe
to save Christmas 2020. 😻 👍🏿"""

# [Adapted from here.](https://thepopbreak.com/2020/11/22/dolly-partons-christ
```

Spacy extracts all interesting information from text with this call.

```
doc = nlp(sample_text)
```

Let’s look at part-of-speech tags.

```
print([(token, token.pos_) for token in doc][:20])
```

```
[(Dolly, 'PROPN'), (Parton, 'PROPN'), (is, 'AUX'), (a, 'DET'), (gift, 'NOUN'),
, 'SPACE'), (From, 'ADP'), (writing, 'VERB'), (all, 'DET'), (-, 'PUNCT'), (time
```

- Often we want to know who did what to whom.
- **Named entities** give you this information.
- What are named entities in the text?

```
print("Named entities:\n", [(ent.text, ent.label_) for ent in doc.ents])
print("\nORG means: ", spacy.explain("ORG"))
print("\nPERSON means: ", spacy.explain("PERSON"))
print("\nDATE means: ", spacy.explain("DATE"))
```

```
Named entities:
 [('Dolly Parton', 'PERSON'), ('Jolene', 'PERSON'), ('9 to 5', 'DATE'), ('Netf

ORG means:  Companies, agencies, institutions, etc.

PERSON means:  People, including fictional

DATE means:  Absolute or relative dates or periods
```

```
from spacy import displacy

displacy.render(doc, style="ent")
```

Dolly Parton PERSON is a gift to us all.

From writing all-time great songs like “ Jolene PERSON ” and “I Will Always Love You”,

to great performances in films like 9 to 5 DATE , to helping fund a COVID-19 vaccine,

she’s given us so much. Now, Netflix ORG bring us Dolly Parton PERSON ’s

Christmas DATE on the Square FAC ,

an original musical that stars Christine Baranski PERSON as a Scrooge-like landowner

who threatens to evict an entire town on Christmas Eve DATE to make room for a new

mall.

Directed and choreographed by the legendary Debbie Allen PERSON and counting

Jennifer Lewis PERSON

and Parton PERSON herself amongst its cast, Christmas DATE on the Square FAC

seems like the perfect movie

to save Christmas 2020 DATE . 😻 👍🏿

An example from a project

Goal: Extract and visualize inter-corporate relationships from disclosed annual 10-K reports of public companies.

Source for the text below.

```
text = (
    "Heavy hitters, including Microsoft and Google, "
    "are competing for customers in cloud services with the likes of IBM and S
)
```

```
doc = nlp(text)
displacy.render(doc, style="ent")
print("Named entities:\n", [(ent.text, ent.label_) for ent in doc.ents])
```

Heavy hitters, including Microsoft **ORG** and Google **ORG** , are competing for customers in cloud services with the likes of IBM **ORG** and Salesforce **PRODUCT** .

```
Named entities:
 [('Microsoft', 'ORG'), ('Google', 'ORG'), ('IBM', 'ORG'), ('Salesforce', 'PROI
```

If you want emoji identification support install `spacymoji` in the course environment.

```
pip install spacymoji
```

After installing `spacymoji`, if it's still complaining about module not found, my guess is that you do not have `pip` installed in your `conda` environment. Go to your course `conda` environment install `pip` and install the `spacymoji` package in the environment using the `pip` you just installed in the current environment.

```
conda install pip
YOUR_MINICONDA_PATH/miniconda3/envs/cpsc330/bin/pip install spacymoji
```

```
from spacymoji import Emoji

nlp.add_pipe("emoji", first=True);
```

Does the text have any emojis? If yes, extract the description.

```
doc = nlp(sample_text)
doc._.emoji
```

```
[('😻', 138, 'smiling cat with heart-eyes'),
 ('👍🏿', 139, 'thumbs up dark skin tone')]
```

Simple feature engineering for our problem.

```
def get_relative_length(text, TWITTER_ALLOWED_CHARS=280.0):
    """
    Returns the relative length of text.

    Parameters:
    ------
    text: (str)
    the input text

    Keyword arguments:
    ------
    TWITTER_ALLOWED_CHARS: (float)
    the denominator for finding relative length

    Returns:
    -------
    relative length of text: (float)

    """
    return len(text) / TWITTER_ALLOWED_CHARS


def get_length_in_words(text):
    """
    Returns the length of the text in words.

    Parameters:
    ------
    text: (str)
    the input text

    Returns:
    -------
    length of tokenized text: (int)

    """
    return len(nltk.word_tokenize(text))


def get_sentiment(text):
    """
    Returns the compound score representing the sentiment: -1 (most extreme ne
    The compound score is a normalized score calculated by summing the valence

    Parameters:
    ------
    text: (str)
    the input text

    Returns:
    -------
    sentiment of the text: (str)
    """
```

```
    scores = sid.polarity_scores(text)
    return scores["compound"]
```

```
train_df = train_df.assign(n_words=train_df["OriginalTweet"].apply(get_length_
train_df = train_df.assign(
    vader_sentiment=train_df["OriginalTweet"].apply(get_sentiment)
)
train_df = train_df.assign(
    rel_char_len=train_df["OriginalTweet"].apply(get_relative_length)
)

test_df = test_df.assign(n_words=test_df["OriginalTweet"].apply(get_length_in_
test_df = test_df.assign(vader_sentiment=test_df["OriginalTweet"].apply(get_se
test_df = test_df.assign(
    rel_char_len=test_df["OriginalTweet"].apply(get_relative_length)
)
```

```
train_df.shape
```

```
(3038, 9)
```

```
X_train
```

	OriginalTweet
1927	While I don't like all of Amazon's choices, to...
1068	Me: shit buckets, it s time to do the weekly s...
803	@SecPompeo @realDonaldTrump You mean the plan ...
2846	@lauvagrande People who are sick aren t panic ...
3768	Coronavirus Panic: Toilet Paper Is the People...
...	...
1122	Photos of our local grocery store shelves wher...
1346	Just went to the the grocery store (Highland F...
3454	Real talk though. Am I the only one spending h...
3437	The supermarket business is booming! #COVID2019
3582	Evening All Here s the story on the and the im...

3038 rows × 1 columns

```
X_train = train_df.drop(columns=["Sentiment"])
```

```
numeric_features = ["vader_sentiment", "rel_char_len", "n_words"]
text_feature = "OriginalTweet"
drop_features = ["UserName", "ScreenName", "Location", "TweetAt"]
```

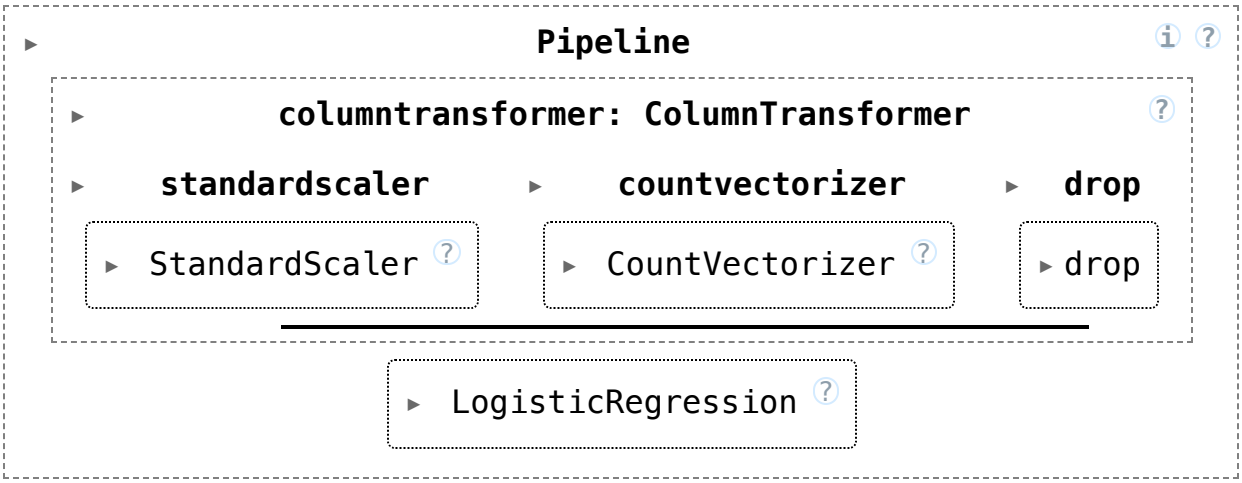
```
preprocessor = make_column_transformer(
    (StandardScaler(), numeric_features),
    (CountVectorizer(stop_words="english"), text_feature),
    ("drop", drop_features),
)
```

```
pipe = make_pipeline(preprocessor, LogisticRegression(max_iter=1000))
results["LR (more feats)"] = mean_std_cross_val_scores(
    pipe, X_train, y_train, return_train_score=True, scoring=scoring_metrics
)
pd.DataFrame(results).T
```

```
/tmp/ipykernel_546162/4158382658.py:26: FutureWarning: Series.__getitem__ trea
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

	fit_time	score_time	test_score	train_score
dummy	0.002 (+/- 0.001)	0.001 (+/- 0.001)	0.280 (+/- 0.001)	0.280 (+/- 0.000)
logistic regression	0.612 (+/- 0.161)	0.022 (+/- 0.002)	0.414 (+/- 0.012)	0.999 (+/- 0.000)
LR (more feats)	0.703 (+/- 0.274)	0.025 (+/- 0.002)	0.690 (+/- 0.006)	0.998 (+/- 0.001)

```
pipe.fit(X_train, y_train)
```




```
cv_feats = (
    pipe.named_steps["columntransformer"]
    .named_transformers_["countvectorizer"]
    .get_feature_names_out().tolist()
)
```

```
feat_names = numeric_features + cv_feats
```

```
coefs = pipe.named_steps["logisticregression"].coef_[0]
```

```
df = pd.DataFrame(
    data={
        "features": feat_names,
        "coefficients": coefs,
    }
)
df.sort_values("coefficients")
```

	features	coefficients
0	vader_sentiment	-6.177901
11329	won	-1.396151
2549	coronapocalypse	-0.823250
2212	closed	-0.758922
8659	retail	-0.732348
...	...	...
3297	don	1.147993
9860	stupid	1.161380
4877	hell	1.296487
3127	die	1.363616
7502	panic	1.512459

11662 rows × 2 columns

We get some improvements with our engineered features!

Common features used in text classification

Bag of words

- So far for text data we have been using bag of word features.
- They are good enough for many tasks. But ...
- This encoding throws out a lot of things we know about language
- It assumes that word order is not that important.
- So if you want to improve the scores further on text classification tasks you carry out **feature engineering**.

Let's look at some examples from research papers.

Example: Label "Personalized" Important E-mails:

- The Learning Behind Gmail Priority Inbox
- Features: bag of words, trigrams, regular expressions, and so on.
- There might be some "globally" important messages:
  - "This is your mother, something terrible happened, give me a call ASAP."
- But your "important" message may be unimportant to others.
  - Similar for spam: "spam" for one user could be "not spam" for another.
- Social features (e.g., percentage of sender emails that is read by the recipient)
- Content features (e.g., recent terms the user has been using in emails)
- Thread features (e.g., whether the user has started the thread)
- ...

The Learning Behind Gmail Priority Inbox

2.1 Features

There are many hundred features falling into a few categories. *Social features* are based on the degree of interaction between sender and recipient, e.g. the percentage of a sender's mail that is read by the recipient. *Content features* attempt to identify headers and recent terms that are highly correlated with the recipient acting (or not) on the mail, e.g. the presence of a recent term in the subject. Recent user terms are discovered as a pre-processing step prior to learning. *Thread features* note the user's interaction with the thread so far, e.g. if a user began a thread. *Label features* examine the labels that the user applies to mail using filters. We calculate feature values during ranking and we temporarily store those values for later learning. Continuous features are automatically partitioned into binary features using a simple ID3 style algorithm on the histogram of the feature values.

Feature engineering examples: Automatically Identifying Good

Conversations Online

BOW $(21k)$	Counts of tokens.
Embeddings (300)	Averaged word embedding values from Google News vectors (Mikolov et al. 2013).
Entity (12)	Counts of named entity types.
Length (2)	Mean # sentences/comment, # tokens/sentence.
Lexicon (6)	# pronouns; agreement and certainty phrases; discourse connectives; and abusive language.
POS $(23k)$	Counts of 1–3-gram POS tags.
Popularity (4)	# thumbs up (TU), # thumbs down (TD), TU + TD, and $\frac{\text{TU}}{\text{TU+TD}}$.
Similarity (8)	Overlap between comment and headline, first comment, previous comment, and all previous comments (if applicable).
User (7)	# comments posted, # threads participated in, # threads initiated, TU and TD received, and commenting rate.

Table 4: Features used in the linear model. The number of features from each group is indicated in parentheses.

N-grams

- Incorporating more context
- A contiguous sequence of *n* items (characters, tokens) in text.

> MDS students are hard-working .

- 2-grams (bigrams): a contiguous sequence of two words
  - MDS students, students are, are hard-working, hard-working ._
- 3-grams (trigrams): a contiguous sequence of three words
  - *MDS students are, students are hard-working, are hard-working .*

You can extract ngram features using `CountVectorizer` by passing `ngram_range`.

```
from sklearn.feature_extraction.text import CountVectorizer

X = [
    "URGENT!! As a valued network customer you have been selected to receive a
    "Lol you are always so convincing.",
    "URGENT!! Call right away!!",
]
vec = CountVectorizer(ngram_range=(1, 3))
X_counts = vec.fit_transform(X)
bow_df = pd.DataFrame(X_counts.toarray(), columns=vec.get_feature_names_out(),
```

```
bow_df
```

	900	900 prize	900 prize reward	always	always so	always so convincing	are	are always	alwa
URGENT!! As a valued network customer you have been selected to receive a $900 prize reward!	1	1	1	0	0	0	0	0	
Lol you are always so convincing.	0	0	0	1	1	1	1	1	
URGENT!! Call right away!!	0	0	0	0	0	0	0	0	

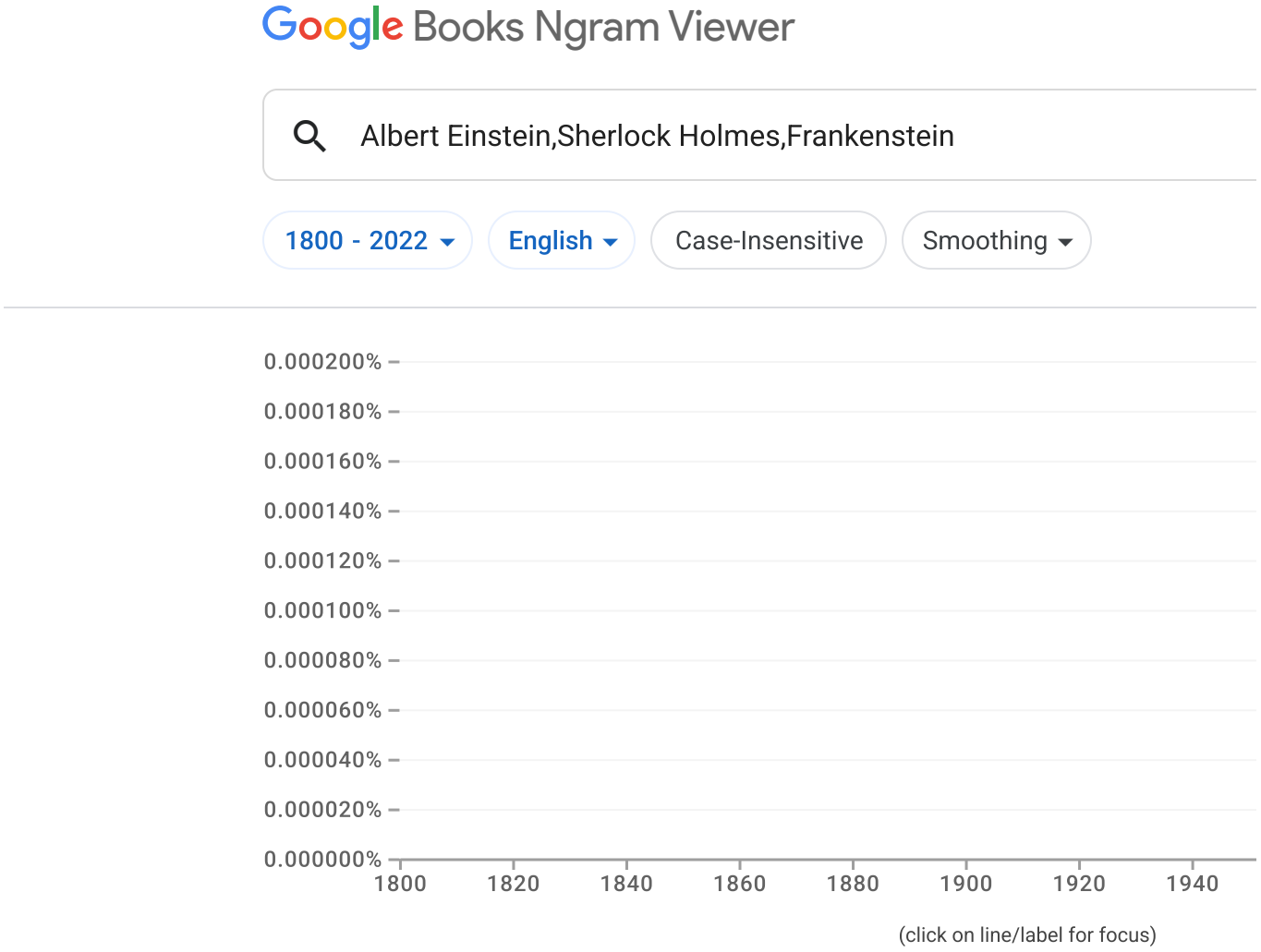
3 rows × 61 columns

(Optional) ASIDE: Google n-gram viewer

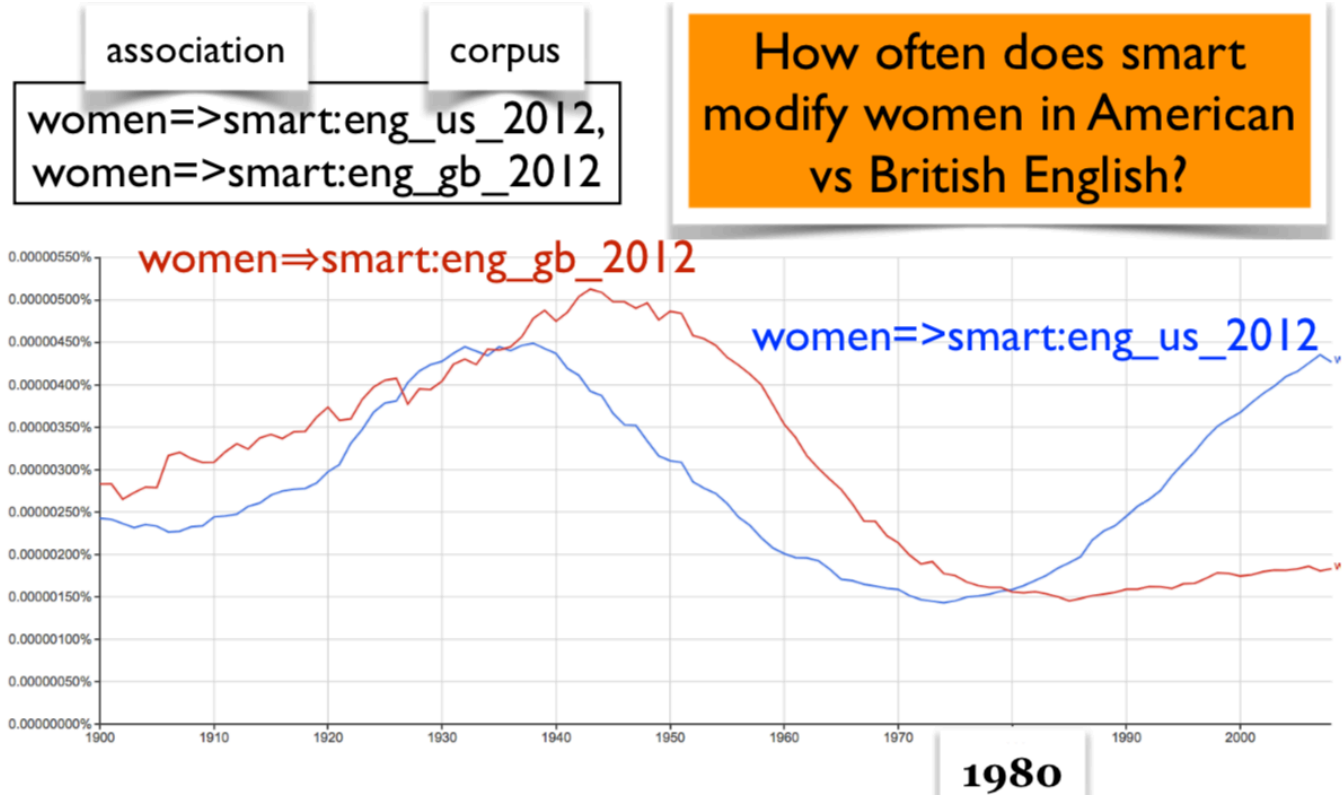
- All Our N-gram are Belong to You
  - https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-toyou.html

> Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times."

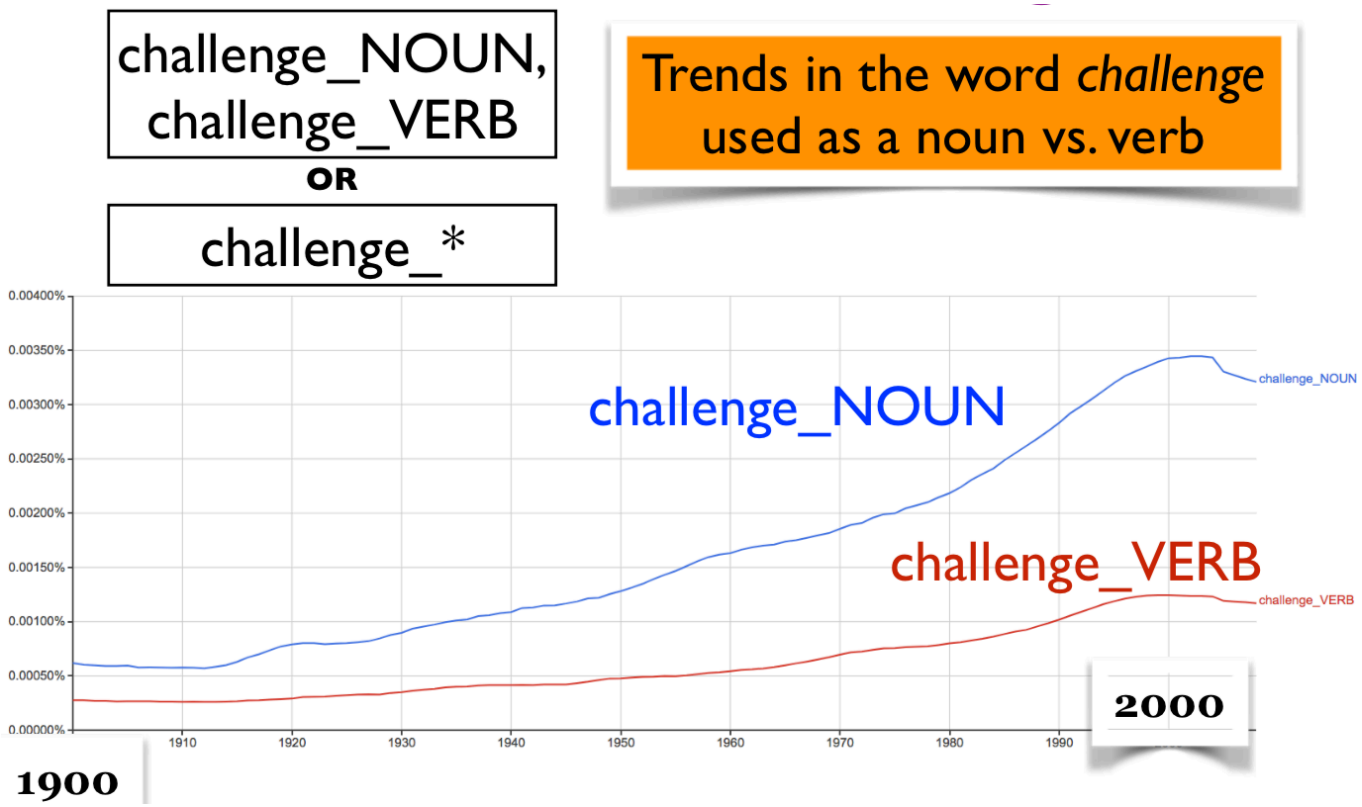
```
from IPython.display import IFrame

url = "https://books.google.com/ngrams/"
IFrame(url, width=1000, height=800)
```



- Count the occurrences of the bigram *smart women* in the corpus from 1800 to 2000



- Trends in the word *challenge* used as a noun vs. verb



Part-of-speech features

Part-of-speech (POS) in English

- Part-of-speech: A kind of syntactic category that tells you some of the grammatical properties of a word.
  - Noun → water, sun, cat
  - Verb → run, eat, teach

> The ____ was running.

- Only a noun fits here.

Part-of-speech (POS) features

- POS features use POS information for the words in text.

> CPSC330/PROPER_NOUN students/NOUN are/VERB hard-working/ADJECTIVE

An example from a project

- Data: a bunch of documents
- Task: identify texts with *permissions* and identify who is giving permission to whom.

> **You** may **disclose** Google confidential information when compelled to do so by law if **you** provide **us** reasonable prior notice, unless a court orders that **we** not receive notice.

- A very simple solution
  - Look for pronouns and verbs.
  - Add POS tags as features in your model.
  - Maybe look up words similar to **disclose**.

Penn Treebank part-of-speech tags (bonus)

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	+,%, &	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

- You also need to download the language model which contains all the pre-trained models. For that run the following in your course `conda` environment.

(Optional) Term weighing (TF-IDF)

- A measure of relatedness between words and documents
- Intuition: Meaningful words may occur repeatedly in related documents, but functional words (e.g., *make*, *the*) may be distributed evenly over all documents

$$tf.idf(w_i, d_j) = (1 + log(tf_{ij}))log\frac{D}{df_i}$$

where,

- tf_{ij} → number of occurrences of the term w_i in document d_j
- D → number of documents
- df_i → number of documents in which w_i occurs

Check `TfidfVectorizer` from `sklearn`.

Interim summary

- In the context of text data, if we want to go beyond bag-of-words and incorporate human knowledge in models, we carry out feature engineering.
- Some common features include:
  - ngram features
  - part-of-speech features
  - named entity features
  - emoticons in text
- These are usually extracted from pre-trained models using libraries such as `spaCy`.
- Now a lot of this has moved to deep learning.
- But many industries still rely on manual feature engineering.

(Optional) Recall RBF Kernel

- Hard to visualize but you can think of this as a weighted nearest-neighbour.
- During prediction, the closest examples have a lot of influence on how we classify the new example compared to the ones further away.
- In general, for both regression/classification, you can think of RBF kernel as "smooth KNN".
- During test time, each training example gets to "vote" on the label of the test point and the amount of vote the n^{th} training example gets is proportional to the the distance between the test point and itself.

RBFs

- What is a radial basis function (RBF)?
  - A set of non-parametric bases that depend on distances to training points.
  - Non-parametric because size of basis (number of features) grows with n.
- Model gets more complicated as you get more data.

Example: RBFs

- Similar to polynomial basis, we transform X to Z.
- Consider X_{train} with three examples: x_1, x_2, and x_3 and 2 features sand X_{test} with two examples: $\tilde{x_1}$ and $\tilde{x_2}$

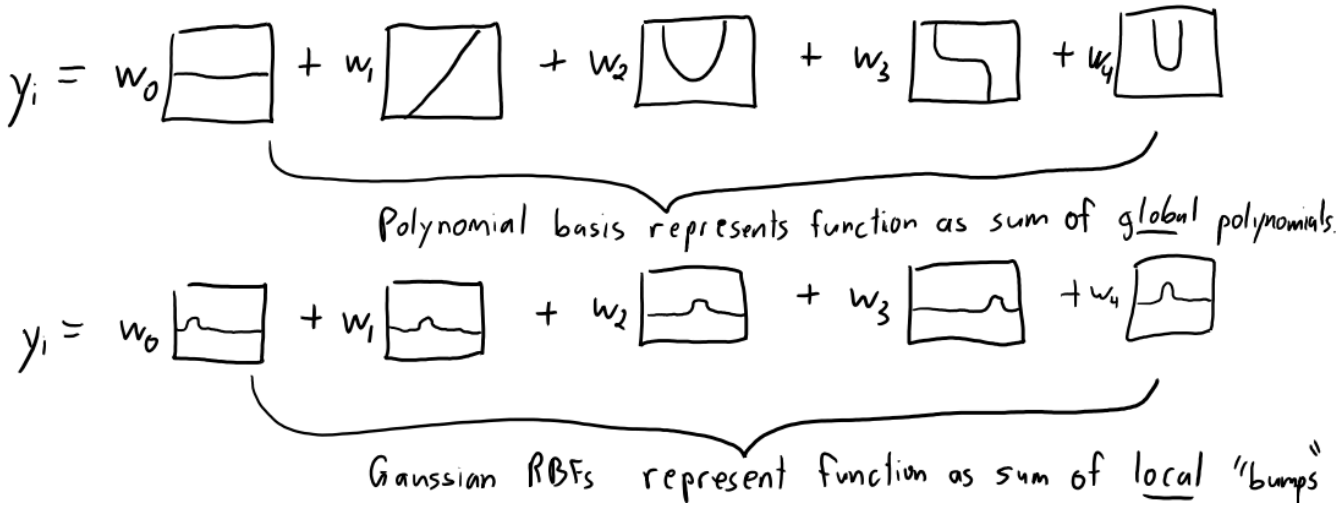
$$\text{Transform } X_{train} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} \text{ to } Z_{train} = \begin{bmatrix} g\|x_1 - x_1\| & g\|x_1 - x_2\| & g\|x_1 - x_3\| \\ g\|x_2 - x_1\| & g\|x_2 - x_2\| & g\|x_2 - x_3\| \\ g\|x_3 - x_1\| & g\|x_3 - x_2\| & g\|x_3 - x_3\| \end{bmatrix}$$

$$\text{Transform } X_{test} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \text{ to } Z_{test} = \begin{bmatrix} g\|\tilde{x_1} - x_1\| & g\|\tilde{x_1} - x_2\| & g\|\tilde{x_1} - x_3\| \\ g\|\tilde{x_2} - x_1\| & g\|\tilde{x_2} - x_2\| & g\|\tilde{x_2} - x_3\| \end{bmatrix}$$

- We create n features.

Gaussian Radial Basis Functions (Gaussian RBFs)

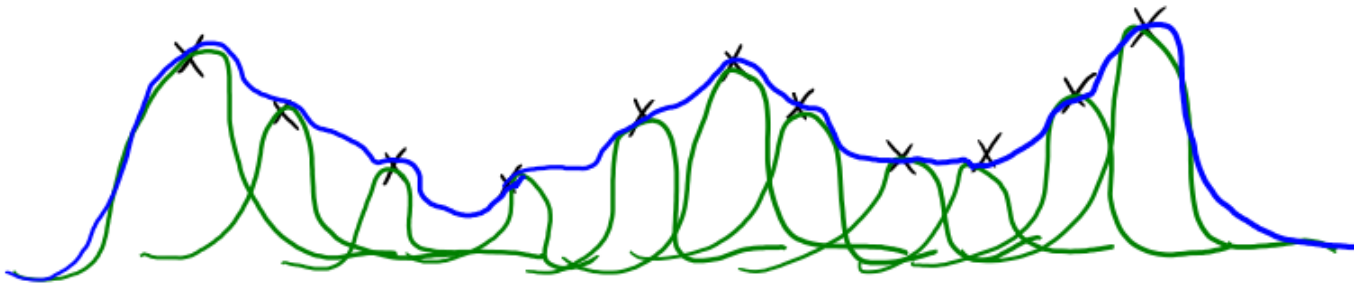
- Most common g is Gaussian RBF: $$g(\varepsilon) = \exp\left(-\frac{\varepsilon^2}{2\sigma^2}\right)$$



Source

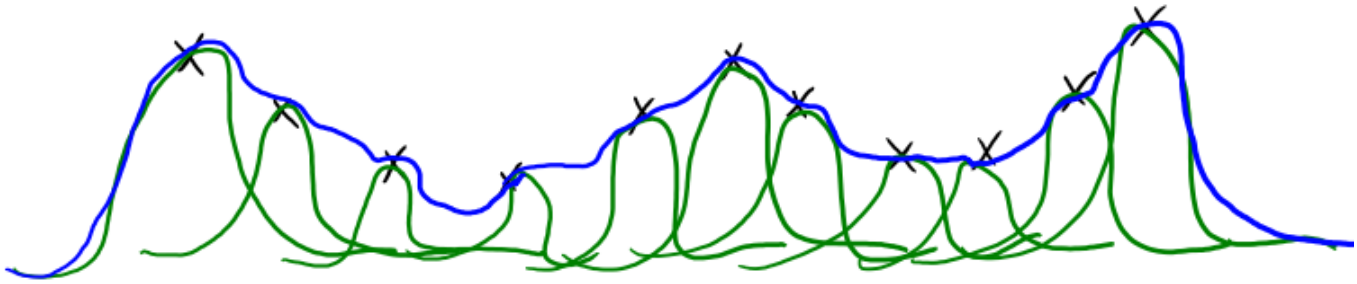
- So in our case: $$g(x_i - x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$
- σ is a hyperparameter that controls the width of the bumps.
- We can fit least squares with different σ values

Gaussian RBFs (non-parametric basis)



source

- How many bumps should we use?
  - We use n bumps (non-parametric basis)
- Where should the bumps be centered?
  - Each bump is centered on one training example x_i.
- How high should the bumps go?
  - Fitting regression weights w gives us the heights (and signs).
- How wide should the bumps be?
  - The width is a hyper-parameter (narrow bumps = complicated model)



source

- Enough bumps can approximate any continuous function to arbitrary precision.
- But with n data points RBFs have n features
  - How do we avoid overfitting with this huge number of features?
  - We regularize w (coming up in two weeks) and use validation error to choose σ and λ.

Interpretation of `gamma` in SVM RBF

- `gamma` controls the complexity (fundamental trade-off).

  - larger `gamma` $\rightarrow$ more complex
  - smaller `gamma` $\rightarrow$ less complex
- The `gamma` hyperparameter in SVC is related to σ in Gaussian RBF.
- You can think of `gamma` as the inverse width of the "bumps"
  - Larger `gamma` means narrower peaks.
  - Narrower peaks means more complex model.

$$g(x_i - x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

Constructing Gaussian RBF with X and σ

```
Z = zeros(n,n)
for i1 in 1:n
    for i2 in 1:n
        Z(i1,i2) = exp(-(norm(X[i1:] - X[i2:])**2)/(2 * sigma**2)
```

Gaussian RBFs: Prediction

- Given a test example $\tilde{x_i}$:

$$\hat{y_i} = w_1 \exp\left(\frac{-\|\tilde{x_i} - x_1\|^2}{2\sigma^2}\right) + w_2 \exp\left(\frac{-\|\tilde{x_i} - x_2\|^2}{2\sigma^2}\right) + \cdots + w_n \exp\left(\frac{-\|\tilde{x_i}}{2}\right.$$

- Expensive at test time: needs distance to all training examples.

RBF with regulariation and optimized $sigma$ and λ

- A model that is hard to beat:
  - RBF basis with L2-regularization and cross-validation to choose σ and λ.
  - Flexible non-parametric basis, magic of regularization, and tuning for test error
- For each value of λ and $sigma$
  - Compute Z on training data

- Compute best weights V using least squares
- Compute $\tilde{Z}$ on validation set (using train set distances)
- Make predictions $\hat{y} = \tilde{Z}v$
- Compute validation error

Using RBF with least squares: KernelRidge (optional)

> sklearn.kernel_ridge.KernelRidge(alpha=1, kernel='linear', gamma=None, degree=3, coef0=1, kernel_params=None)

> Kernel ridge regression. Kernel ridge regression (KRR) combines ridge regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function in the original space.

> The form of the model learned by KRR is identical to support vector regression (SVR). However, different loss functions are used: KRR uses squared error loss while support vector regression uses epsilon-insensitive loss, both combined with l2 regularization. In contrast to SVR, fitting a KRR model can be done in closed-form and is typically faster for medium-sized datasets. On the other hand, the learned model is non-sparse and thus slower than SVR, which learns a sparse model for epsilon > 0, at prediction-time.

❓❓ Questions for you

Select all of the following statements which are TRUE.

- (A) Suppose we add quadratic features to dataset X and the augmented dataset is Z. Fitting linear regression on Z would learn a linear function of Z.

- (B) The least squares loss function shown below is independent of the bias term (the y intercept).

$$J(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

- (C) If you get the same validation error with polynomials of degrees d and $d+4$, it is a better to pick the polynomial of degree d.
- (D) If you are given a large dataset with 1000 features, it's a good idea to start simple and work with one or two features in order to verify your intuitions.
- (E) Suppose you apply polynomial transformations with degree 3 polynomial during training. During prediction time on the test set, you must calculate degree three polynomial features of the given feature vector in order to get predictions.

▶ Click to view the answers

Summary

What did we learn today?

- Feature engineering is finding the useful representation of the data that can help us effectively solve our problem.
- Non-linear regression (change of basis)
- Radial basis functions
- Importance of feature engineering in text data and audio data.

Feature engineering

- The best features are application-dependent.

- It's hard to give general advice. But here are some guidelines.
  - Ask the domain experts.
  - Go through academic papers in the discipline.
  - Often have idea of right discretization/standardization/transformation.
  - If no domain expert, cross-validation will help.
- If you have lots of data, use deep learning methods.

> The algorithms we used are very standard for Kagglers ... We spent most of our efforts in feature engineering...
> - Xavier Conort, on winning the Flight Quest challenge on Kaggle