Feature Selection and Overfitting

Presenter: Katyaini

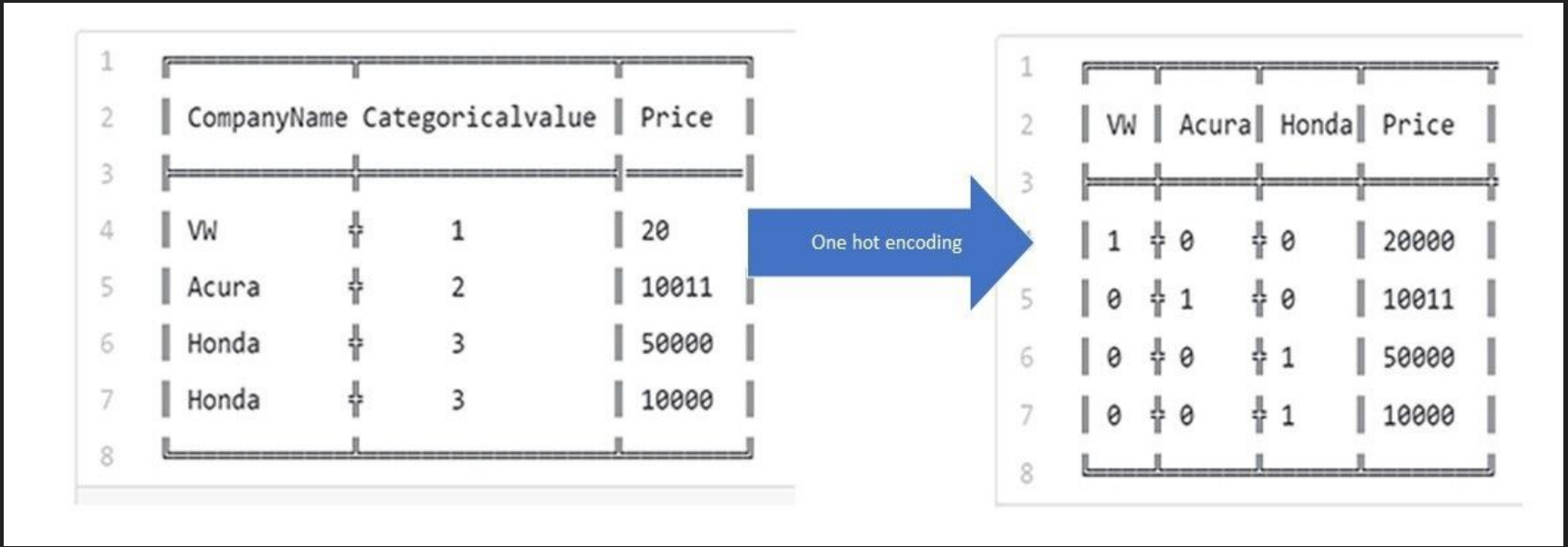
Feature Engineering

- Feature engineering is the process of using domain knowledge to extract features from raw data via data mining techniques. These features can be used to improve the performance of machine learning algorithms.

- For instance interpolating missing data; converting it to a form that the machine learning algorithm takes as input(for text data this could be converting it to a one-hot encoding), outlier detection, log transform

One-Hot Encoding

A one hot encoding is a representation of categorical variables as binary vectors.

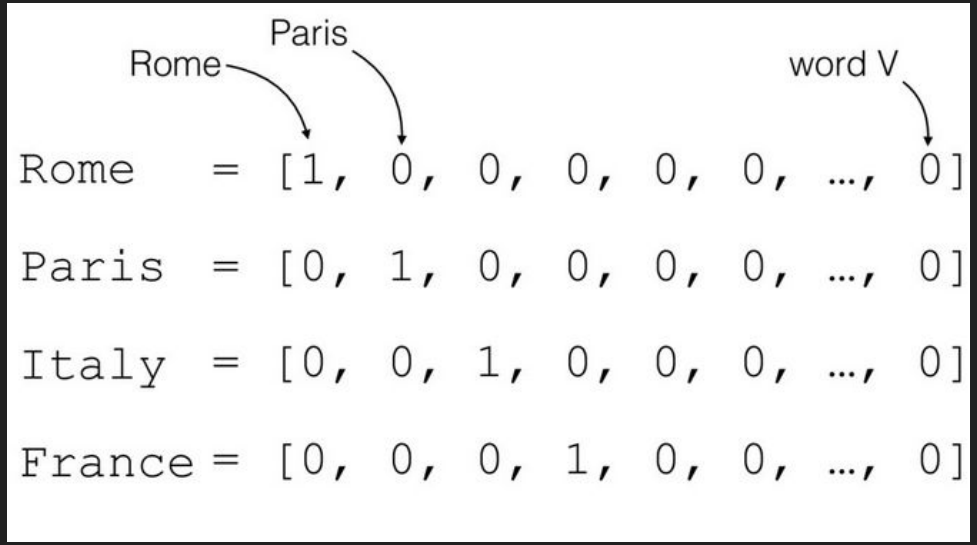


How it works with text data

So what are some issues that might pop up using this approach?

- Un-ordered, therefore the context of words are lost.
- The vector representation is in binary form, therefore no frequency information is taken into account.

Source :(Marco Bonzanini, 2017)



TF-IDF Vectorization - potential improvement

A weighting system to evaluate how important a word is to a document in a collection of documents.

Determined using frequency!

TF-IDF is the product of the TF and IDF scores of the term.

$$TF\text{-}IDF=TF/IDF$$

Tf

Term Frequency : This summarizes how often a given word appears within a document.

TF =

$$\frac{\text{Number of times a word appears in a document}}{\text{Total num of words}}$$

A term has a high TF score if it appears frequently in a document!

Idf

Inverse Document Frequency: This downscales words that appear a lot across documents.

IDF=

$$\text{Inv}\left(\frac{\text{Number of documents}}{\text{Number docs the term appears in}}\right)$$

A term has a high IDF score if it appears in a few documents. Conversely, if the term is very common among documents, it would have a low IDF score.

Tf-idf demo

In [ ]:

In [1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [4]:
```
tfidf = TfidfVectorizer(sublinear_tf=True,
                        ngram_range=(1, 2),
                        stop_words='english')
test_arr = ['this is a sentence','lets talk about doughnuts','doughnuts are great','data science is great' ]
features = tfidf.fit_transform(test_arr).toarray()
```

```
C:\Users\Katyaini\Anaconda3\envs\python3\lib\site-packages\sklearn\feature_extraction\text.py:1059: FutureWarning: Conversion o
f the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 =
= np.dtype(float).type`.
  if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.float):
```

In [5]:
```
features
```

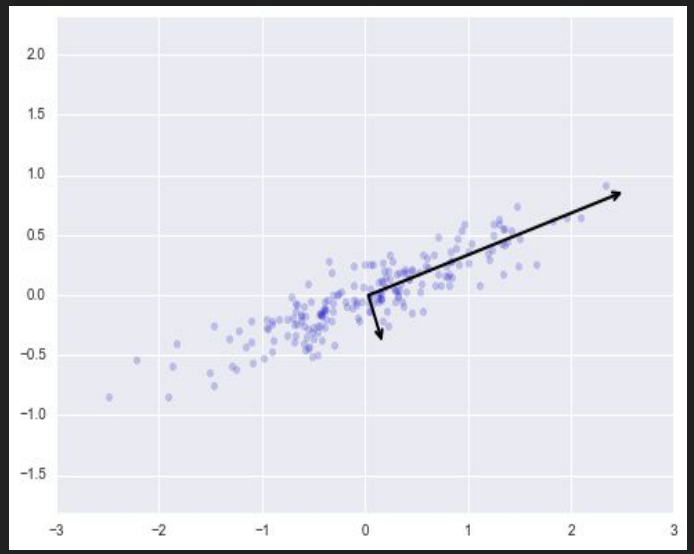
Out[5]:
```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 1.        ,
        0.        , 0.        ],
       [0.        , 0.        , 0.36673901, 0.        , 0.        ,
        0.46516193, 0.46516193, 0.        , 0.        , 0.        ,
        0.46516193, 0.46516193],
       [0.        , 0.        , 0.52640543, 0.66767854, 0.52640543,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
       [0.46516193, 0.46516193, 0.        , 0.        , 0.36673901,
        0.        , 0.        , 0.46516193, 0.46516193, 0.        ,
        0.        , 0.        ]])
```

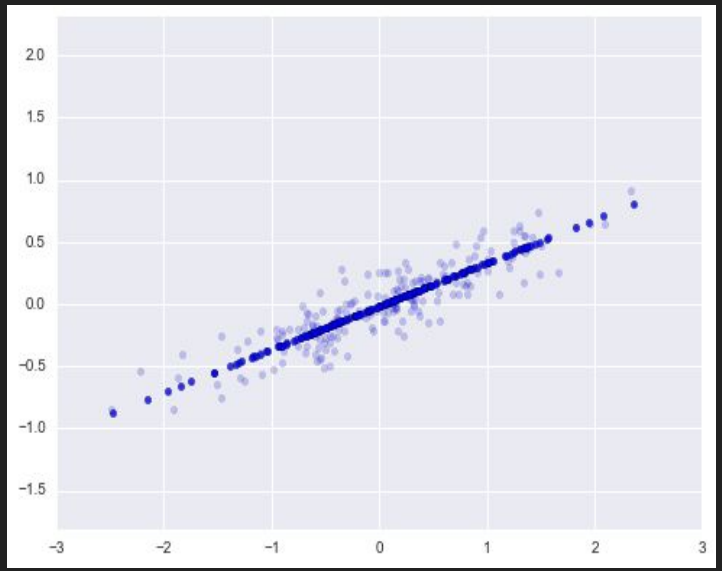
In [ ]:

Optimization and Interpretation

- Do we really need all variables/features we have? Or why might it be useful to drop certain features?
  - Curse of dimensionality — Overfitting
  - Non informative features - Poor-quality input will produce Poor-Quality output.
  - large number of features make a model bulky, time-taking, and harder to implement in production.

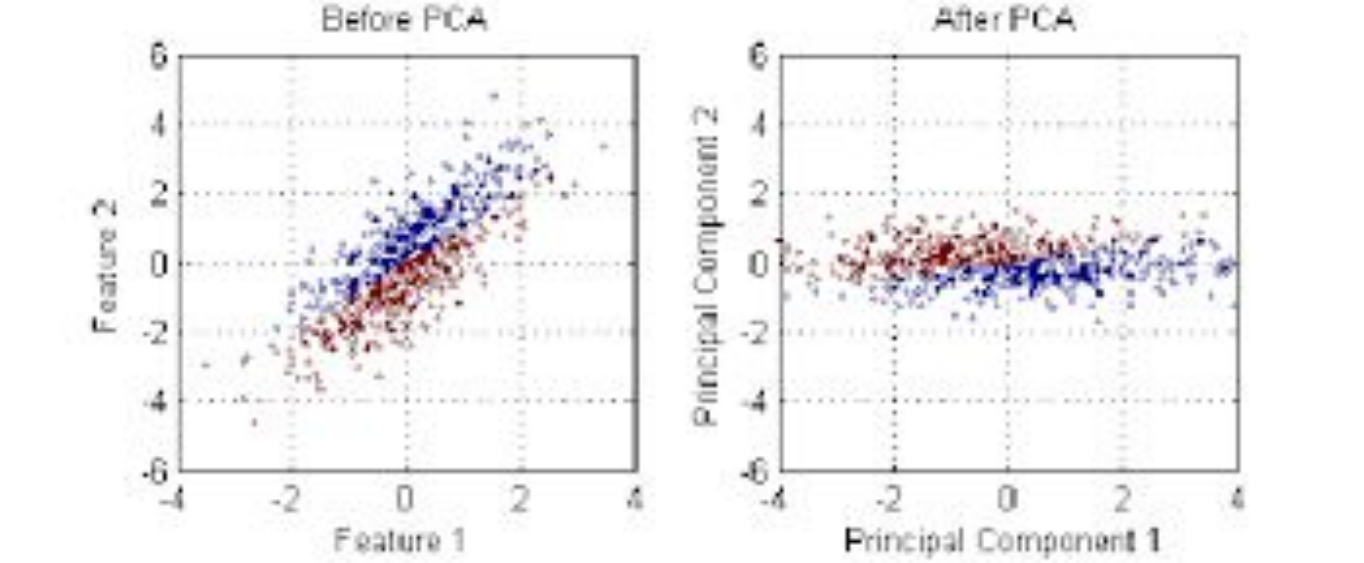
PCA: Drop dependent/insignificant variables





Why can we do this?

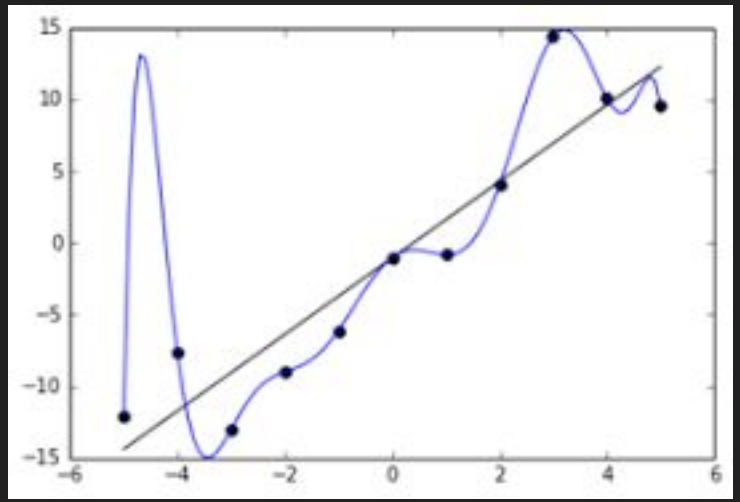
Notice how X and Y are linearly dependent- basically means knowing X you can predict y. No extra information is being provided by Y, so we can just combine info into 1 feature.

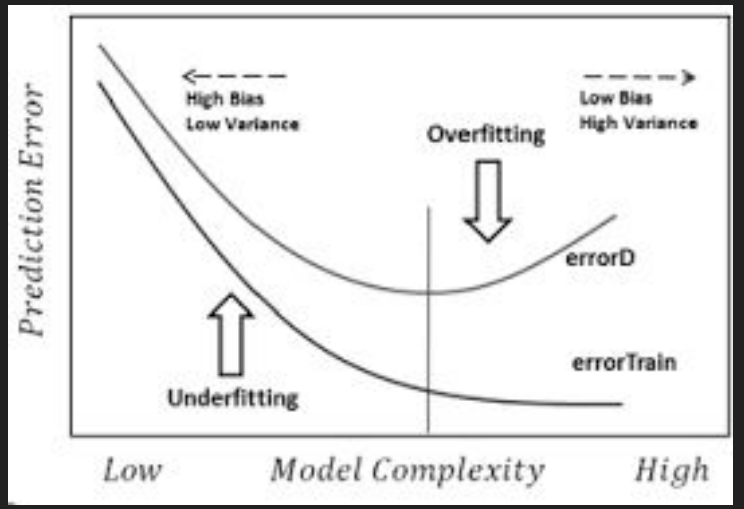


Overfitting

If we have more columns in the data than the number of rows, we will be able to fit our training data perfectly, but that won't generalize to the new samples. And thus we learn absolutely nothing.

In this case your model will have low training error but high testing error





Solution : Regularization

Can you not only minimize the cost function but also restrict the parameters not to become too large?

$$J = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(X^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Where λ is a constant to control the value of regularization term and n is the number of the features.

The regularization term penalizes large parameters. Obviously, minimizing the cost function consists of reducing both terms in the right: the MSE term and the regularization term. So each time some parameter is updated to become significantly large, it will increase the value of the cost function by the regularization term, and as a result, it will be penalized and updated to a small value.

L1(Lasso Regression) and L2(Ridge Regression)

Ridge Regression

“squared magnitude” of coefficient as penalty term

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Lasso Regression

“absolute value of magnitude” of coefficient as penalty term

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$