CS-GY 6923: Lecture 2
Multiple Linear Regression + Feature Transformations + Model Selection

NYU Tandon School of Engineering, Akbar Rafiey

Slides by Prof. Christopher Musco

Course news

- First lab assignment `lab1.ipynb` due **Monday, by midnight.**
- Lab 02 will be posted this weekend.
- First written assignment will be posted next week.

Recap: supervised learning

Training Dataset:

- Given input pairs $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_i$ is an input data vector (the predictor).
- Each y_i is a (continuous) output variable (the target).

Objective:

- Have the computer automatically find some function $f(\mathbf{x})$ such that $f(\mathbf{x}_i)$ is close to y_i for the input data.

Standard approach: Convert the supervised learning problem to a multi-variable optimization problem.

Supervised learning definitions

What are the three components needed to setup a supervised learning problem?

- **Model** $f_{\boldsymbol{\theta}}(x)$**:** Class of equations or programs which map input x to predicted output. We want $f_{\boldsymbol{\theta}}(x_i) \approx y_i$ for training inputs.
- **Model Parameters** $\boldsymbol{\theta}$**:** Vector of numbers. These are numerical nobs which parameterize our class of models.
- **Loss Function** $L(\boldsymbol{\theta})$**:** Measure of how well a model fits our data. Typically some function of $f_{\boldsymbol{\theta}}(x_1) - y_1, \ldots, f_{\boldsymbol{\theta}}(x_n) - y_n$

Empirical Risk Minimization: Choose parameters $\boldsymbol{\theta}^*$ which minimize the Loss Function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

Simple linear regression

Simple Linear Regression

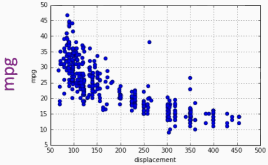
- Model: $f_{\beta_0,\beta_1}(x) = \beta_0 + \beta_1 \cdot x$
- Model Parameters: β_0, β_1
- Loss Function: $L(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - f_{\beta_0,\beta_1}(x_i))^2$

Goal: Choose β_0, β_1 to minimize
$L(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$.

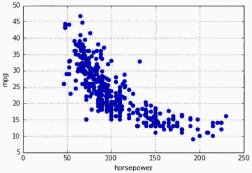
Simple closed form solution: $\beta_1 = \sigma_{xy}/\sigma_x^2$,$\beta_0 = \bar{y} - \beta_1\bar{x}$. **How did we solve for this solution?**

Example from last class

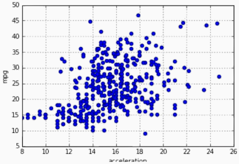
Predict miles per gallon of a vehicle given information about its engine/make/age/etc.



Displacement



Horsepower



Acceleration

Multiple linear regression

More common goal

Predict target y using multiple features, simultaneously.

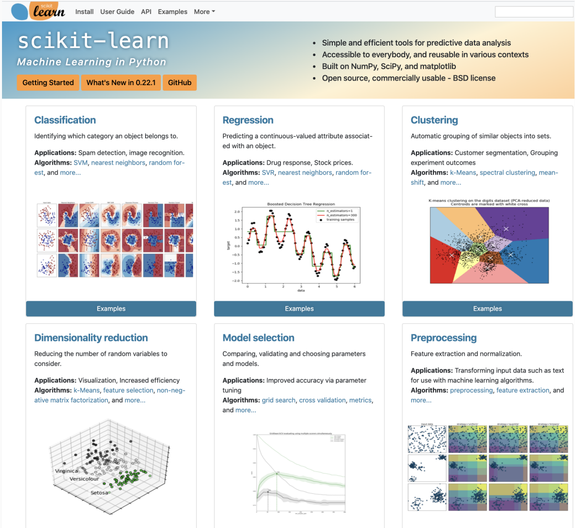
Motivating example: Predict diabetes progression in patients after 1 year based on health metrics. (Measured via numerical score.)

Features: Age, sex, average blood pressure, six blood serum measurements (e.g. cholesterol, lipid levels, iron, etc.)

Demo in `demo_diabetes.ipynb (Demo 3)`.

Libraries for this demo

Introducing **Scikit Learn**.



scikit-learn

Machine Learning in Python

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross
Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...

Scikit learn

Pros:

- One of the most popular “traditional” ML libraries.
- Many built in models for regression, classification, dimensionality reduction, etc.
- Easy to use, works with ‘numpy‘, ‘scipy‘, other libraries we use.
- Great for rapid prototyping, testing models.

Cons:

- Everything is very “black-box”: difficult to debug, understand why models aren’t working, speed up code, etc.

Scikit learn

Modules used:

- `datasets` module contains a number of pre-loaded datasets. Saves time over downloading and importing with `pandas`.
- `linear_model` can be used to solve Multiple Linear Regression. A bit overkill for this simple model, but gives you an idea of `sklearn`'s general structure.

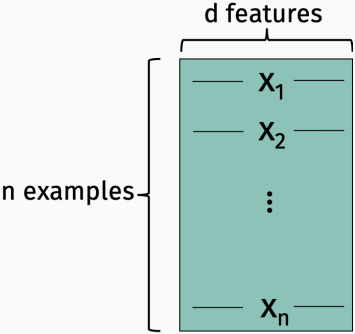
The data matrix

Target variable:

- Scalars $y_1, \ldots, y_n$ for n data examples (a.k.a. samples).

Predictor variables:

- d dimensional vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ for n data examples and d features



Linear algebra review

Now it the time to review your linear algebra!

Notation:

- Let $\mathbf{X}$ be an $n \times d$ matrix. Written $\mathbf{X} \in \mathbb{R}^{n \times d}$.
- $\mathbf{x}_i$ is the i^{th} row of the matrix.
- $\mathbf{x}^{(j)}$ is the j^{th} column.
- x_{ij} is the i, j entry.
- For a vector $\mathbf{y}$, y_i is the i^{th} entry.
- $\mathbf{X}^T$ is the matrix transpose.
- $\mathbf{y}^T$ is a vector transpose.

Linear algebra review

Things to remember:

- Matrix multiplication. If we multiply $\mathbf{X} \in \mathbb{R}^{n \times d}$ by $\mathbf{Y} \in \mathbb{R}^{d \times k}$ we get $\mathbf{X}\mathbf{Y} = \mathbf{Z} \in \mathbb{R}^{n \times k}$.
- Inner product/dot product. $\langle \mathbf{y}, \mathbf{z} \rangle = \sum_{i=1}^{n} y_i z_i$.
- $\langle \mathbf{y}, \mathbf{z} \rangle = \mathbf{y}^T \mathbf{z} = \mathbf{z}^T \mathbf{y}$.
- Euclidean norm: $\|\mathbf{y}\|_2 = \sqrt{\mathbf{y}^T \mathbf{y}}$.
- $(\mathbf{X}\mathbf{Y})^T = \mathbf{Y}^T \mathbf{X}^T$.

Linear algebra review

Things to remember:

- Identity matrix is denoted as $\mathbf{I}$.
- "Most" square matrices have an inverse: i.e. if $\mathbf{Z} \in \mathbb{R}^{n \times n}$, there is a matrix $\mathbf{Z}^{-1}$ such that $\mathbf{Z}^{-1}\mathbf{Z} = \mathbf{Z}\mathbf{Z}^{-1} = \mathbf{I}$.
- Let $\mathbf{D} = \text{diag}(\mathbf{d})$ be a diagonal matrix containing the entries in $\mathbf{d}$.
- $\mathbf{XD}$ scales the columns of $\mathbf{X}$. $\mathbf{DX}$ scales the rows.

Linear algebra review

You also need to be comfortable working with matrices in `numpy` .
Go through the `demo_numpy_matrices.ipynb (Demo 1)` slowly.

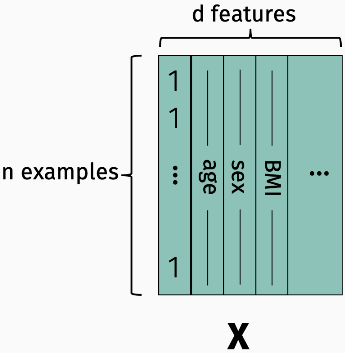
The data matrix

Target variable:

- Scalars $y_1, \ldots, y_n$ for n data examples (a.k.a. samples).

Predictor variables:

- d dimensional vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ for n data examples and d features



Multiple linear regression

Data matrix indexing:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ x_{31} & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

Multiple Linear Regression Model:

$$\text{Predict} \qquad y_i \approx \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_d x_{id}$$

The rate at which diabetes progresses depends on many factors, with each factor having a different magnitude effect.

Multiple linear regression

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ x_{31} & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} = \begin{bmatrix} 1 & x_{12} & \dots & x_{1d} \\ 1 & x_{22} & \dots & x_{2d} \\ 1 & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

Multiple Linear Regression Model:

$$\text{Predict} \qquad y_i \approx \boldsymbol{\beta_1} + \beta_2 x_{i2} + \ldots + \beta_d x_{id}$$

In this case, β_1 serves as the "intercept" parameter.

Multiple linear regression

Multiple Linear Regression Model:

$$\text{Predict} \qquad y_i \approx \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_d x_{id}$$

Data matrix:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ x_{21} & x_{22} & \ldots & x_{2d} \\ x_{31} & x_{32} & \ldots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nd} \end{bmatrix} = \begin{bmatrix} 1 & x_{12} & \ldots & x_{1d} \\ 1 & x_{22} & \ldots & x_{2d} \\ 1 & x_{32} & \ldots & x_{3d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n2} & \ldots & x_{nd} \end{bmatrix}$$

Linear algebraic form:

$$y_i \sim \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle$$

$$\mathbf{y} \sim \mathbf{X}\boldsymbol{\beta}$$

Multiple linear regression

Linear Least-Squares Regression.

- Model Parameters:

$$\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_d]$$

- Model:

$$f_{\boldsymbol{\beta}}(\mathbf{x}) = \langle \mathbf{x}, \boldsymbol{\beta} \rangle$$

- Loss Function:

$$\begin{aligned} L(\boldsymbol{\beta}) &= \sum_{i=1}^{n} |y_i - \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle|^2 \\ &= \|\mathbf{y} - \mathbf{X}\beta\|_2^2 \end{aligned}$$

Linear algebraic form of loss function

Loss minimization

Machine learning goal: minimize the loss function $L(\boldsymbol{\beta}) : \mathbb{R}^d \rightarrow \mathbb{R}$.

Find possible optima by determining for which $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_d]$ all the **gradient** equals $\mathbf{0}$. I.e. when do we have:

$$\nabla L(\boldsymbol{\beta}) = \begin{bmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_d} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \ldots \\ 0 \end{bmatrix}$$

Gradient

Loss function:

$$L(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$$

Gradient:

$$-2 \cdot \mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

Can check that this is equal to 0 if $\boldsymbol{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$. There are no other options, so this must be the minimum.

Single variable warmup

What is the derivative of: $f(x) = x^2$?

Gradient

Loss function: $L(\beta) = \|\mathbf{y} - \mathbf{X}\beta\|_2^2$

Multiple linear regression solution

Take away: simple form for the gradient means that multiple linear regression models are easy and efficient to optimize.

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

- Often the “go to” first regression method. Throw your data in a matrix and see what happens.
- Serve as the basis for much richer classes of models.

Multiple linear regression solution

Need to compute $\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$.

- Main cost is computing $(\mathbf{X}^T\mathbf{X})^{-1}$ which takes $O(nd^2)$ time.
- Can solve slightly faster using the method `numpy.linalg.lstsq`, which is running an algorithm based on QR decomposition.
- For larger problems, can solve much faster using an *iterative methods* like `scipy.sparse.linalg.lsqr`.

Will learn more about iterative methods when we study Gradient Descent.

Encoding data as a numerical matrix

It is not always immediately clear how to do this! One of the first issue we run into is categorical data:

$$
\begin{aligned}
\mathbf{x}_1 &= [42, 4, 104, \texttt{hybrid}, \texttt{ford}] \\
\mathbf{x}_2 &= [18, 8, 307, \texttt{gas}, \texttt{bmw}] \\
\mathbf{x}_2 &= [31, 4, 150, \texttt{gas}, \texttt{honda}] \\
&\vdots
\end{aligned}
$$

Encoding data as a numerical matrix

Binary data is easy to deal with – pick one category to be 0, one to be 1. The choice doesn't matter – it will not impact the overall loss of the model

$$\begin{aligned}
\mathbf{x}_1 &= [42, 4, 104, \texttt{hybrid}, \texttt{ford}] \\
\mathbf{x}_2 &= [18, 8, 307, \texttt{gas}, \texttt{bmw}] \\
\mathbf{x}_2 &= [31, 4, 150, \texttt{gas}, \texttt{honda}] \\
&\vdots
\end{aligned}$$

$$\begin{aligned}
\mathbf{x}_1 &= [42, 4, 104, \texttt{1}, \texttt{ford}] \\
\mathbf{x}_2 &= [18, 8, 307, \texttt{0}, \texttt{bmw}] \\
\mathbf{x}_2 &= [31, 4, 150, \texttt{0}, \texttt{honda}] \\
&\vdots
\end{aligned}$$

Dealing with categorical variables

What about a categorical predictor variable for car make with more than 2 options: e.g. Ford, BMW, Honda. **How would you encode as a numerical column?**

$$\begin{bmatrix} \texttt{ford} \\ \texttt{ford} \\ \texttt{honda} \\ \texttt{bmw} \\ \texttt{honda} \\ \texttt{ford} \end{bmatrix} \rightarrow \begin{bmatrix} \qquad \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

One hot encoding

Better approach: One Hot Encoding.

$$\begin{bmatrix} \texttt{ford} \\ \texttt{ford} \\ \texttt{honda} \\ \texttt{bmw} \\ \texttt{honda} \\ \texttt{ford} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

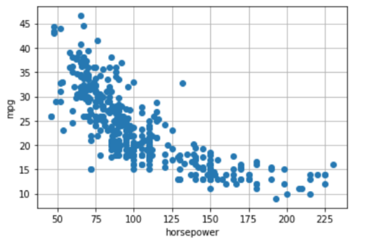
- Create a separate feature for every category, which is 1 when the variable is in that category, zero otherwise.
- Not too hard to do by hand, but you can also use library functions like `sklearn.preprocessing.OneHotEncoder`.

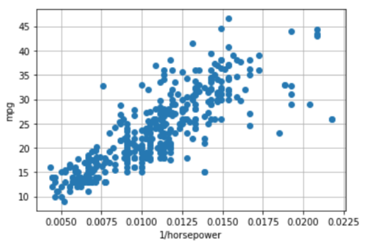
Avoids adding inadvertent linear relationships.

Transformed linear models

Example from last time

Instead of fitting the model mpg $\approx \beta_0 + \beta_1 \cdot$ horsepower, fit the model mpg $\approx \beta_0 + \beta_1 \cdot 1/$horsepower.





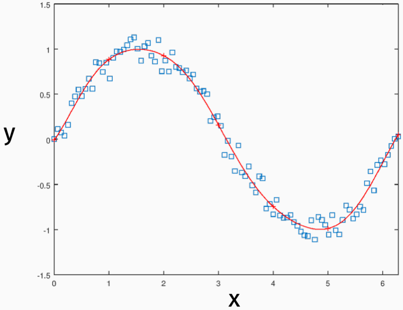
How would you know to make such a transformation?

Better approach: Choose a more flexible non-linear model class.

Transformed linear models

Suppose we have singular variate data examples (x, y). We could fit the non-linear polynomial model:

$$y \approx \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3.$$



Claim: This can be done using an algorithm for multivariate regression!

Transformed linear models

Transform into a multiple linear regression problem:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^1 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}$$

What is the output of $\mathbf{X}\beta$?

Transformed linear models

More generally, have each column j is generated by a different basis function $\phi_j(x)$. Could have:

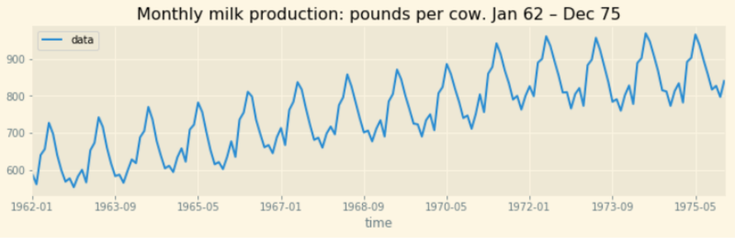
- $\phi_j(x) = x^q$
- $\phi_j(x) = sin(x)$
- $\phi_j(x) = cos(10x)$
- $\phi_j(x) = 1/x$

When might you want to include sins and cosines?

Transformed linear models

When might you want to include sins and cosines?

Time series data:



There is usually not much harm in including irrelevant variable transformation.

Multinomial model

Transformations can also be for multivariate data.

Example: Multinomial model.

- Given a dataset with target y and predictors x, z.
- For inputs $(x_1, z_1), \ldots, (x_n, z_n)$ construct the data matrix:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & z_1 & z_1^2 & x_1 z_1 \\ 1 & x_2 & x_2^2 & z_2 & z_2^2 & x_2 z_2 \\ \vdots & \vdots & & \vdots & & \\ 1 & x_n & x_n^2 & z_n & z_n^2 & x_n z_n \end{bmatrix}$$

- Captures non-linear interaction between x and z.

Model selection

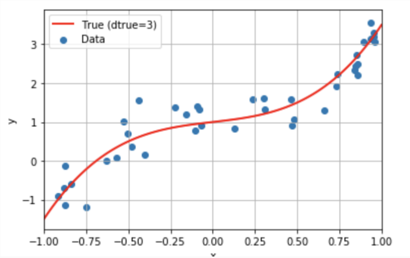
Remainder of lecture: Learn about model selection, test/train paradigm, and cross-validation through a simple example.

I have a Python demo working through this example.

Fitting a polynomial

Simple experiment:

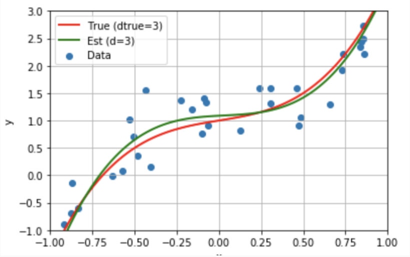
- Randomly select data points $x_1, \ldots, x_n \in [-1, 1]$.
- Choose a degree 3 polynomial $p(x)$.
- Create some fake data: $y_i = p(x_i) + \eta$ where η is a random number (e.g random Gaussian).



Fitting a polynomial

Simple experiment:

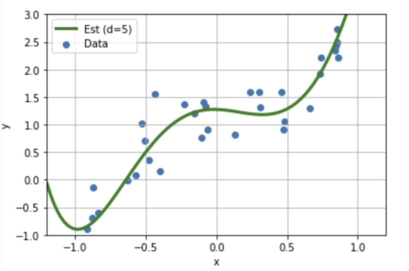
- Use multiple linear regression to fit a degree 3 polynomial.

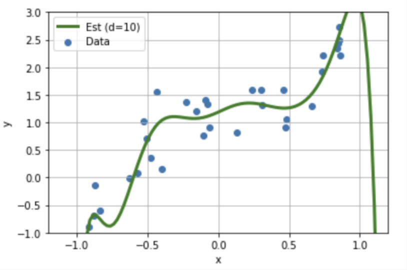


fitting a polynomial

What if we fit a higher degree polynomial?

- Fit degree 5 polynomial under squared loss.
- Fit degree 10 polynomial under squared loss.

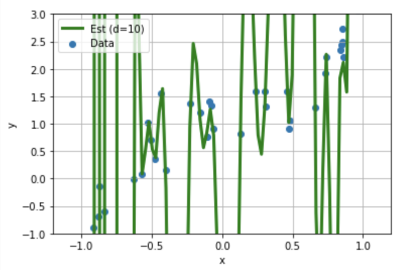




Fitting a polynomial

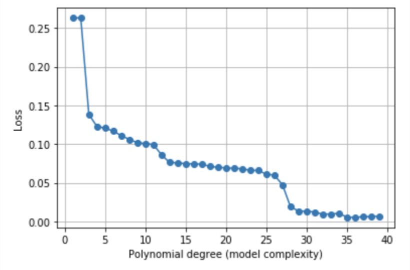
Even higher?

- Fit degree 40 polynomial under squared loss.



Model selection

The more **complex** our model class (i.e. the higher degree we allow) the better our loss:



Model selection

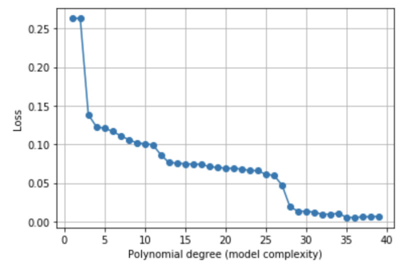
Consider $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\bar{\mathbf{X}} = [\mathbf{X}, \mathbf{z}] \in \mathbb{R}^{n \times d+1}$ with one additional column appended on.

Claim:

$$\min_{\bar{\boldsymbol{\beta}} \in \mathbb{R}^{d+1}} \|\bar{\mathbf{X}}\bar{\boldsymbol{\beta}} - \mathbf{y}\|_2^2 \leq \min_{\boldsymbol{\beta} \in \mathbb{R}^{d}} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2.$$

Model selection

The more **complex** our model class the better our loss:



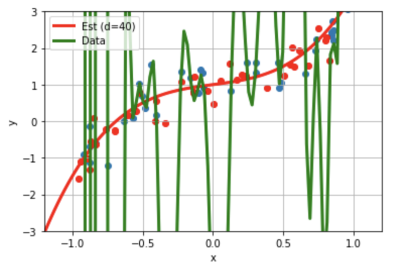
So training loss alone is not usually a good metric for model selection. Small loss does not imply generalization.

Generalization: How well do we do on new data.

Model selection

Problem: Loss alone is not informative for choosing model.

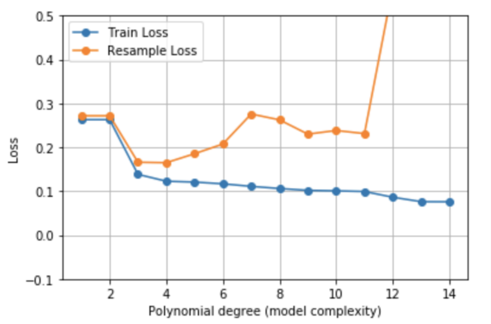
For more complex models, we get smaller loss on the training data, but don't expect to perform well on "new" data:



In other words, the model does not **generalize.**

Model selection

Solution: Directly test model on “new data”.



- Loss continues to decrease as model complexity grows.
- Performance on new data “turns around” once our model gets too complex. Minimized around degree 4.

Train-test paradigm

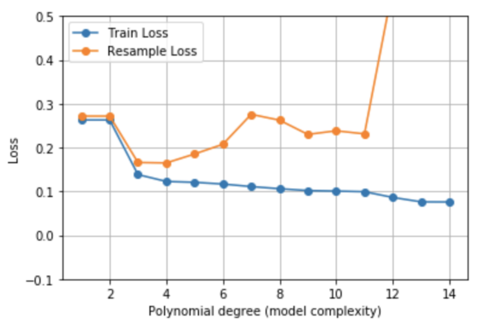
More reasonable approach: Evaluate model on fresh test data which was not used during training.

Test/train split:

- Given data set $(\mathbf{X}, \mathbf{y})$, split into two sets $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$ and $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$.
- Train q models $f^{(1)}, \ldots, f^{(q)}$ by finding parameters which minimize the loss on $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$.
- Evaluate loss of each trained model on $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$.

Sometimes you will see the term **validation set** instead of test set. Sometimes there will be both: use validation set for choosing the model, and test set for getting a final performance measure.

Train-test paradigm



- **Train loss** continues to decrease as model complexity grows.
- **Test loss** “turns around” once our model gets too complex. Minimized around degree $3 - 4$.

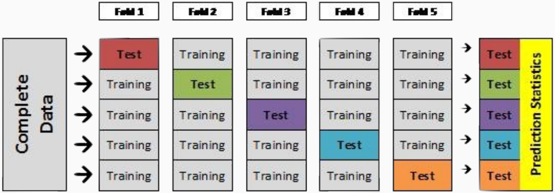
Generalization error

If the test loss remains low, we say that the model **generalizes**.
Test lost is often called **generalization error.**

Train-test paradigm

Typical train-test split: 90-70% / 10-30%. Trade-off between optimization of model parameters and better estimate of model performance.

K-fold cross validation



Complete Data

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	
Test	Training	Training	Training	Training	Test
Training	Test	Training	Training	Training	Test
Training	Training	Test	Training	Training	Test
Training	Training	Training	Test	Training	Test
Training	Training	Training	Training	Test	Test

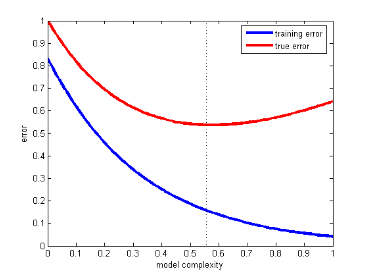
Prediction Statistics

- Randomly divide data in K parts.
    - Typical choice: $K = 5$ or $K = 10$.
- Use $K - 1$ parts for training, 1 for test.
- For each model, compute test loss L_{ts} for each “fold”.
- Choose model with best average loss.
- Retrain best model on entire dataset.

Is there any disadvantage to choosing K larger?

The fundamental curve of ML

The above trend is fairly representative of what we tend to see across the board:



Train-test intuition

Is “test error” the end goal though? Don’t we care about “future” error?

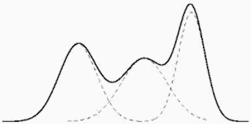
Intuition: Models which perform better on the test set will **generalize** better to future data.

Goal: Introduce a little bit of formalism to better understand what this means. What is “future” data?

Statistical learning model

Statistical Learning Model:

- Assume each data example is randomly drawn from some distribution $(\mathbf{x}, y) \sim \mathcal{D}$.



✓

✓

✓

E.g. $x_1, \ldots, x_d$ are Gaussian random variables with parameters $\mu_1, \sigma_1, \ldots, \mu_d, \sigma_d$.

This is not (really) a simplifying assumption! The distribution could be arbitrarily complicated.

Risk

Statistical Learning Model:

- Assume each data example is randomly drawn from some distribution $(\mathbf{x}, y) \sim \mathcal{D}$.
- Define the **Risk** of a model/parameters:

$$R(f, \boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\ell\left(f(\mathbf{x}, \boldsymbol{\theta}), y\right)\right]$$

here ℓ is our loss function (e.g. $\ell(z, y) = |z - y|$ or $\ell(z, y) = (z - y)^2$).

Goal: Find model $f \in \{f^{(1)}, \ldots, f^{(q)}\}$ and parameter vector $\boldsymbol{\theta}$ to minimize the $R(f, \boldsymbol{\theta})$.

Risk

- **(Population) Risk**:

$$R(f, \boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} \left[\ell \left(f(\mathbf{x}, \boldsymbol{\theta}), y \right) \right]$$

- **Empirical Risk**: Draw $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \sim \mathcal{D}$

$$R_E(f, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell \left(f(\mathbf{x}_i, \boldsymbol{\theta}), y_i \right)$$

Empirical risk

For any fixed model f and parameters $\boldsymbol{\theta}$,

$$\mathbb{E}\left[R_E(f, \boldsymbol{\theta})\right] = R(f, \boldsymbol{\theta}).$$

Only true if f and $\boldsymbol{\theta}$ are chosen *without looking at the data used to compute the empirical risk.*

Model selection

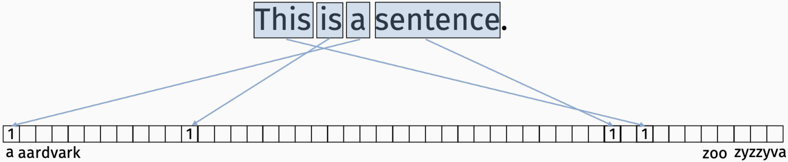
- Train q models $(f^{(1)}, \boldsymbol{\theta}_1^*), \ldots, (f^{(q)}, \boldsymbol{\theta}_q^*)$.
- For each model, compute empirical risk $R_E(f^{(i)}, \boldsymbol{\theta}_i^*)$ using test data.
- Since we assume our original dataset was drawn independently from $\mathcal{D}$, so is the random test subset.

No matter how our models were trained or how complex they are, $R_E(f^{(i)}, \boldsymbol{\theta}_i^*)$ is an unbiased estimate of the true risk $R(f^{(i)}, \boldsymbol{\theta}_i^*)$ for every i. Can use it to distinguish between models.

Model selection example

bag-of-words models and **n-grams**

Common way to represent documents (emails, webpages, books) as numerical data. The ultimate example of 1-hot encoding.

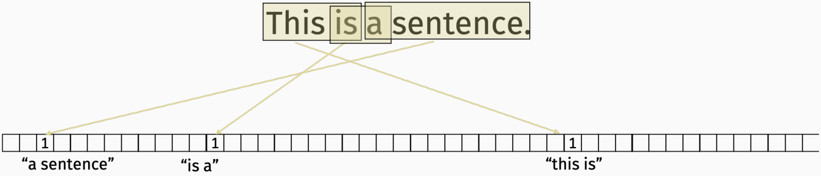


bag-of-words

Model selection example

bag-of-words models and **n-grams**

Common way to represent documents (emails, webpages, books) as numerical data. The ultimate example of 1-hot encoding.

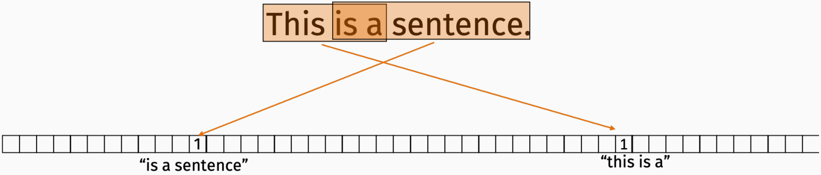


bi-grams

Model selection example

bag-of-words models and **n-grams**

Common way to represent documents (emails, webpages, books) as numerical data. The ultimate example of 1-hot encoding.



tri-grams

Model selection example

Models of increasing order:

- Model $f_{\boldsymbol{\theta}_1}^{(1)}$: spam filter that looks at **single words**.
- Model $f_{\boldsymbol{\theta}_2}^{(2)}$: spam filter that looks at **bi-grams**.
- Model $f_{\boldsymbol{\theta}_3}^{(3)}$: spam filter that looks at **tri-grams**.
- . . .

“interest” “low interest” “low interest loan”

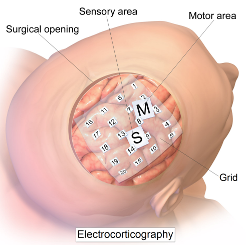
Increased length of **n-gram** means more expressive power.

Will also be relevant in our first generative ML lab!

Model selection example

Electrocorticography ECoG (upcoming lab):

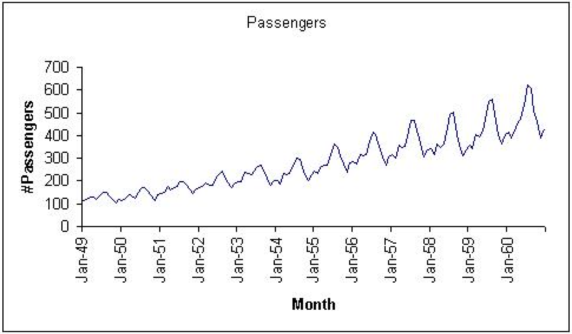
- Implant grid of electrodes on surface of the brain to measure electrical activity in different regions.



- Predict hand motion based on ECoG measurements.
- **Model order:** predict movement at time t using brain signals at time $t, t-1, \dots, t-q$ for varying values of q.

Autoregressive model

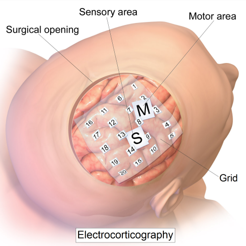
Predicting time t based on a linear function of the signals at time $t, t-1, \ldots, t-q$ is not the same as fitting a line to the time series. It's much more expressive.



Predecessor of modern "recurrent neural networks".

Model selection lab tip

Electrocorticography ECoG lab:



First lab where computation actually matters (solving regression problems with $\sim 40k$ examples, ~ 1500 features)

Makes sense to test and debug code using a subset of the data.

Adaptive data analysis

Slight caveat: This is typically not how machine learning or scientific discovery works in practice!

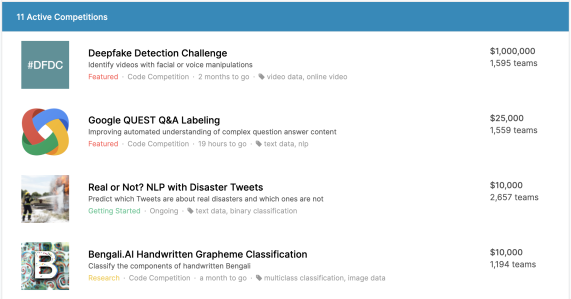
Typical workflow:

- Train a class of models.
- Test.
- Adjust class of models.
- Test.
- Adjust class of models.
- Cont...

Final model implicitly depends on test set because performance on the test set guided how we changed our model.

Adaptive data analysis

Popularity of ML benchmarks and competitions leads to adaptivity at a massive scale.



11 Active Competitions

Deepfake Detection Challenge — $1,000,000 · 1,595 teams
Identify videos with facial or voice manipulations
Featured · Code Competition · 2 months to go · video data, online video

Google QUEST Q&A Labeling — $25,000 · 1,559 teams
Improving automated understanding of complex question answer content
Featured · Code Competition · 19 hours to go · text data, nlp

Real or Not? NLP with Disaster Tweets — $10,000 · 2,657 teams
Predict which Tweets are about real disasters and which ones are not
Getting Started · Ongoing · text data, binary classification

Bengali.AI Handwritten Grapheme Classification — $10,000 · 1,194 teams
Classify the components of handwritten Bengali
Research · Code Competition · a month to go · multiclass classification, image data

Kaggle (various competitions)

IMAGENET

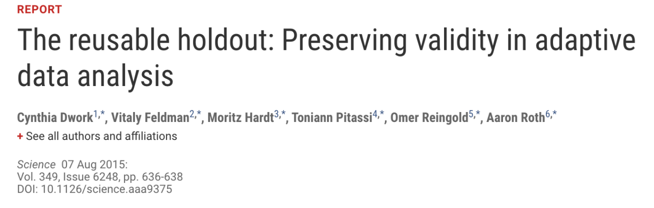
14,197,122 images, 21841 synsets indexed

Explore Download Challenges Publications Updates About
Not logged in. Login I Signup

Imagenet (image classification and categorization)

Adaptive data analysis

Is adaptivity a problem? Does it lead to over-fitting? How much? How can we prevent it? All current research.



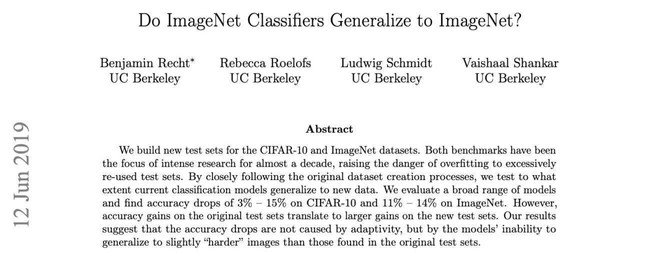
REPORT

The reusable holdout: Preserving validity in adaptive data analysis

Cynthia Dwork[1,*], Vitaly Feldman[2,*], Moritz Hardt[3,*], Toniann Pitassi[4,*], Omer Reingold[5,*], Aaron Roth[6,*]

+ See all authors and affiliations

Science 07 Aug 2015:
Vol. 349, Issue 6248, pp. 636-638
DOI: 10.1126/science.aaa9375



Do ImageNet Classifiers Generalize to ImageNet?

Benjamin Recht*	Rebecca Roelofs	Ludwig Schmidt	Vaishaal Shankar
UC Berkeley	UC Berkeley	UC Berkeley	UC Berkeley

12 Jun 2019

Abstract

We build new test sets for the CIFAR-10 and ImageNet datasets. Both benchmarks have been the focus of intense research for almost a decade, raising the danger of overfitting to excessively re-used test sets. By closely following the original dataset creation processes, we test to what extent current classification models generalize to new data. We evaluate a broad range of models and find accuracy drops of 3% – 15% on CIFAR-10 and 11% – 14% on ImageNet. However, accuracy gains on the original test sets translate to larger gains on the new test sets. Our results suggest that the accuracy drops are not caused by adaptivity, but by the models' inability to generalize to slightly "harder" images than those found in the original test sets.

Imagenet dataset

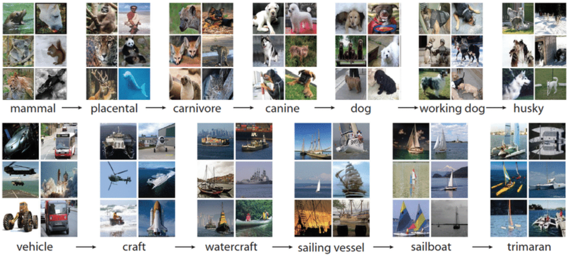
IMAGENET

14,197,122 images, 21841 synsets indexed

Explore Download Challenges Publications Updates About

Not logged in. Login I Signup

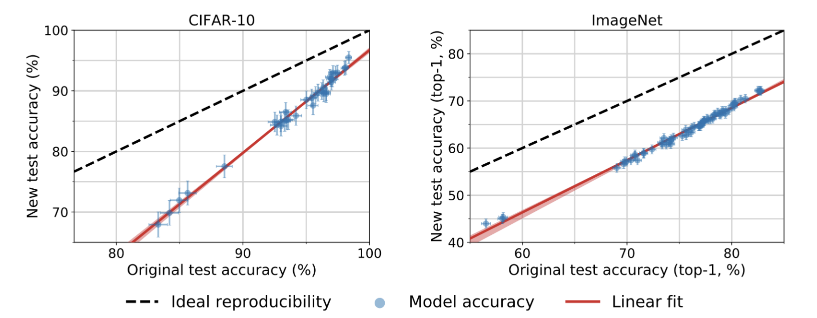
Collected by Fei-Fei Li's group at Stanford in 2006ish and labeled using Amazon Mechanical Turk.


mammal → placental → carnivore → canine → dog → working dog → husky
vehicle → craft → watercraft → sailing vessel → sailboat → trimaran

We now have neural network models that can solve these classification problems with > 95% accuracy.

Adaptive data analysis

Do ImageNet Classifiers Generalized to ImageNet?



Interestingly, when comparing popular vision models on "fresh" data, while performance dropped across the board, the relative rank of model performance did not change significantly.