An Introduction to Machine Learning

The implementation and interpretation of key models

Terence Parr
MSDS program
University of San Francisco

UNIVERSITY OF SAN FRANCISCO

Course topics

- Regularization of linear models (finishes off linear regression topic)
- Models (Naïve Bayes, kNN, Decision trees, Random Forests, DL)
- Model interpretation (feature importance)
- Clustering (k-means, hierarchical clustering)
- In ML Lab (sister course):
  - Data clean up, feature engineering, dealing with missing data
  - Model assessment (metrics, ROC/PR curves)
- A book to become familiar with:
  - The Elements of Statistical Learning:
    https://web.stanford.edu/~hastie/Papers/ESLII.pdf

UNIVERSITY OF SAN FRANCISCO

Before we jump in…

- Let's take a quick high-level overview, identify the key ideas
    - What problem are we solving?
    - What does it mean to train a model?
    - Training data, features
    - What doesn't look like in Python?
    - Model assessment
    - Train, validate, test

# Central problem of ~~supervised~~ machine learning

- Given: X is *explanatory matrix*, y is the *target* or *response*

|	bedrooms	bathrooms	latitude	longitude	price
	3	1.5	40.7145	-73.9425	3000
observations	2	1.0	40.7947	-73.9667	5465
or	1	1.0	40.7388	-74.0018	2850
records	1	1.0	40.7539	-73.9677	3275
	4	1.0	40.8241	-73.9493	3350

- Build a system that makes accurate future predictions based upon *training data* (X, y) from the past
- BUT, w/o being overly-specific to this training data (don't *overfit*)

UNIVERSITY OF SAN FRANCISCO

Regressor vs Classifier; 2 sides of same coin

- If target is numerical, model is a *regressor*
- If target is *categorical*, model is a *classifier*
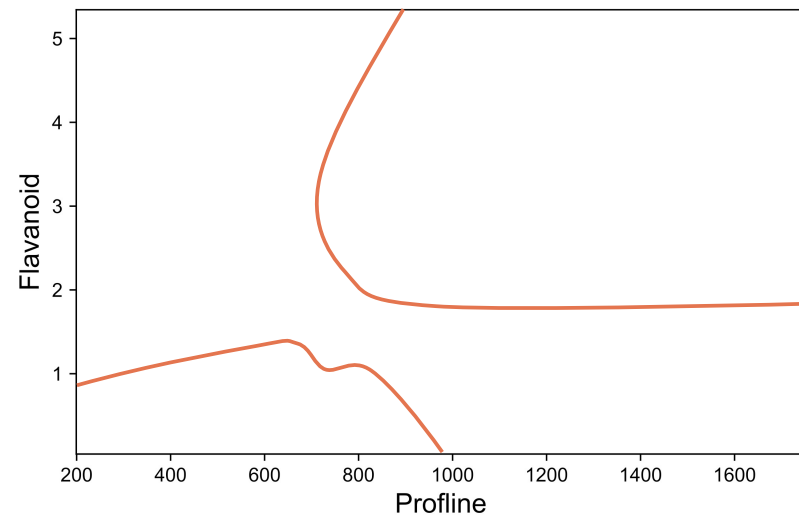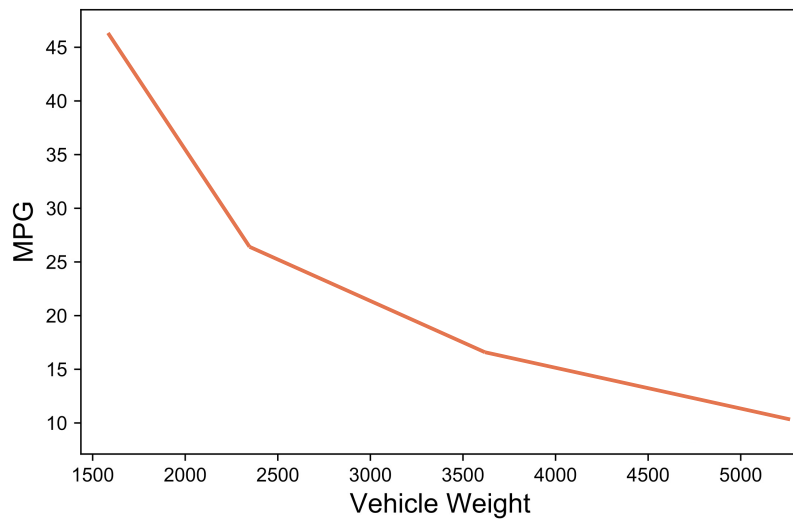- Regressors draw through data, classifiers draw between clusters
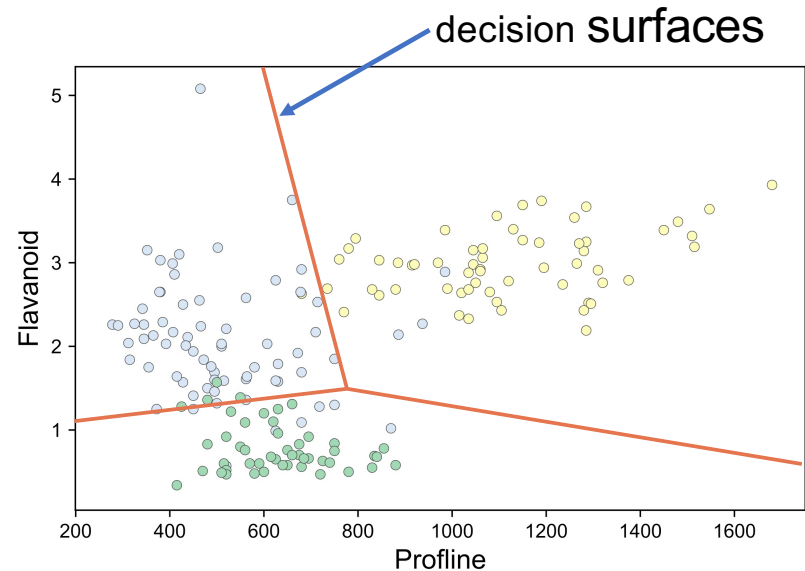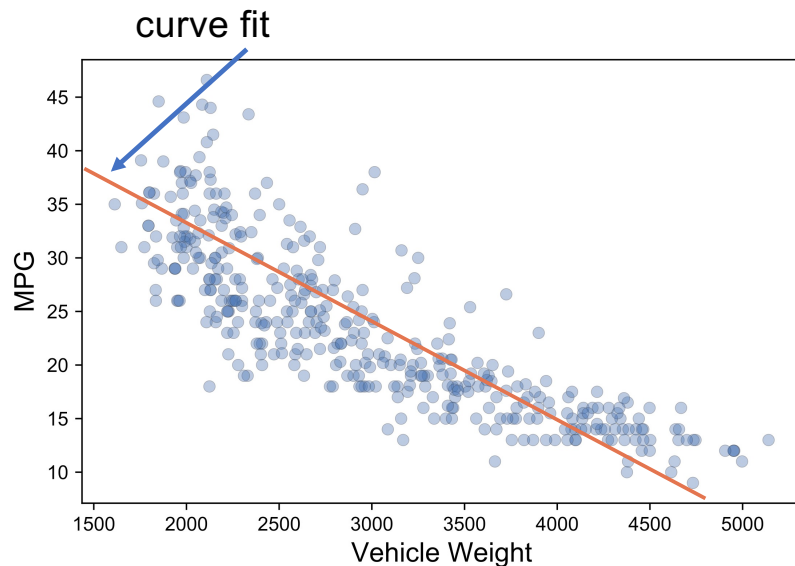
Different models have different surfaces
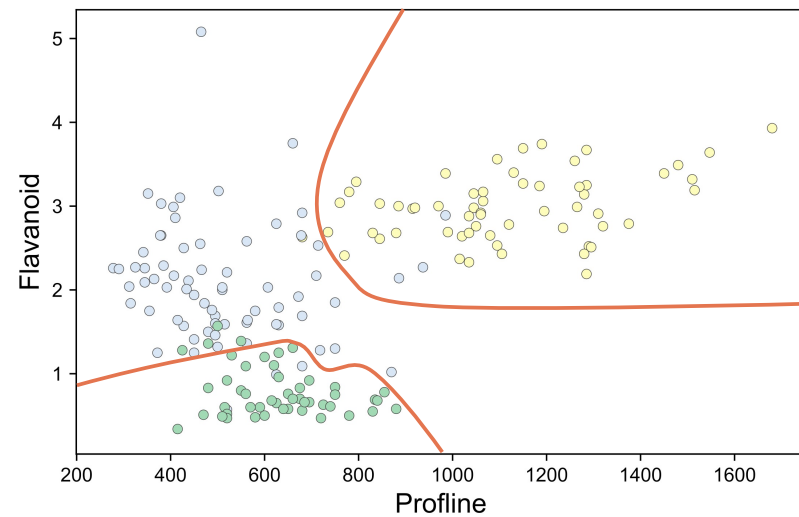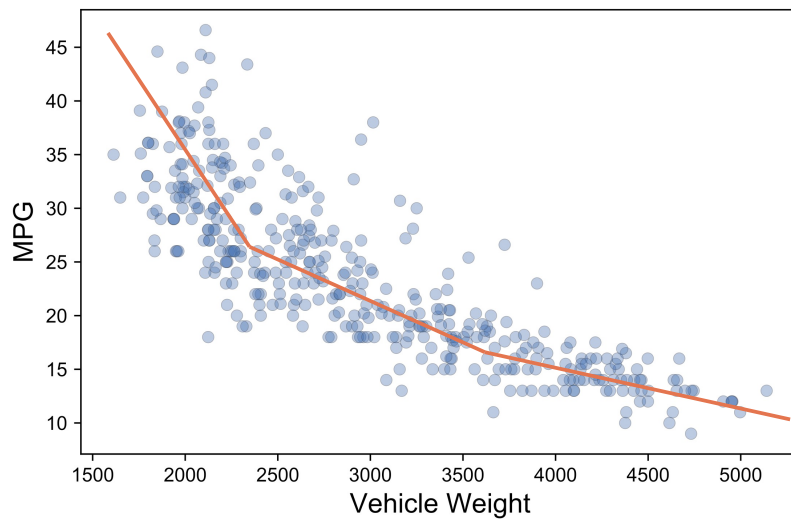
We try to find a function, $y = f(x)$, that predicts a value or class;
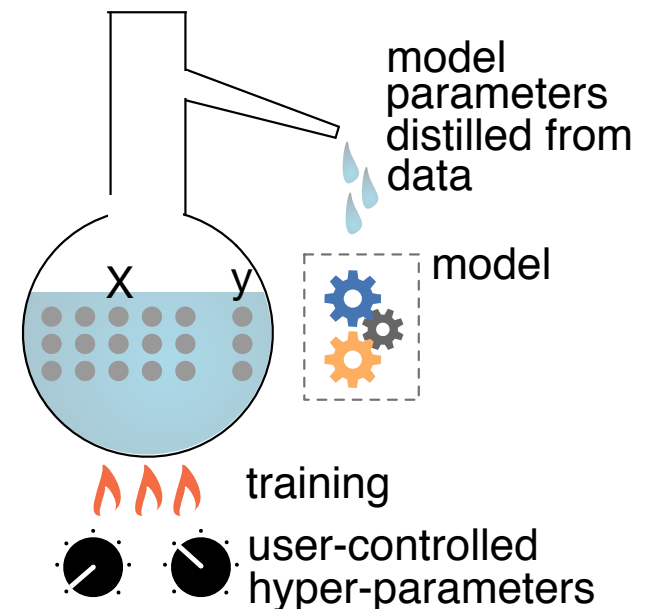f is called the model and is a function of *model parameters*

UNIVERSITY OF SAN FRANCISCO

The model is just the decision surface(s)

In a sense, a model is a condensation or compression of training data

UNIVERSITY OF SAN FRANCISCO

Training (*fitting*) a model

- Distill training data into model *parameters*
- *Hyperparameters* control distillation, *parameters* are the condensate
- Parameters:
  - beta coefficients for linear model
  - tree structure (split vars/vals, …) for decision tree
  - kNN is extreme where data are the parameters
- Hyperparameters: num trees, learning rate, …
- A *model* = algorithm + parameters
- Algorithm could be linear math equation or decision tree walker etc…
- Training is usually a lossy compression; e.g., linear regression of 2 vars condenses any amount of data down to 3 floats!!

model parameters distilled from data

model

X   y

training

user-controlled hyper-parameters

UNIVERSITY OF SAN FRANCISCO

A good model is <u>all about the features</u>

- Good features are usually more important than the model
- Example: 3-word voice recognition, HMM vs Rocchio
- Focus on *feature engineering* not choosing the model
- Your default models:
    - For *structured* data, use *random forests* or *boosted trees*
    - For *unstructured* (like images), use *neural networks* (nets of linear models)
- Generally speaking, these models are tolerant of noise and superfluous features
- Means we can throw every feature we can think of at model
- Caveat: deep learning computes its own features from raw data

Feature engineering

- Synthesize new features from existing features
- A few common synthesized features:
  - frequency encoding; e.g., getting info about records from ID feature
  - e.g., derive age from sale and manufacturing dates

	modelid	modelfreq	saledate	builddate	age
0	101	0.25	2012-02-03	2010-01-28	736 days
1	992	0.10	2012-04-19	2005-09-10	2413 days

Synthesized features

- Or, derive from external sources; e.g., isholiday from date

UNIVERSITY OF SAN FRANCISCO

What training, prediction look like in code

- In scikit-learn, swapping out model is trivial:

```
lm = LinearRegression()
lm.fit(X, y)
```

```
rf = RandomForestRegressor()
rf.fit(X, y)
```

```
x = [[2, 1, 40.794, -74.00]]
y_pred = lm.predict(x)
```

```
x = [[2, 1, 40.794, ...]]
y_pred = rf.predict(x)
```

- LinearRegression and RandomForest are objects representing models and hyperparameters go in as args to constructor
- We're going to build our own versions as class projects

UNIVERSITY OF SAN FRANCISCO

Is our model any good?

- Define good? Good at what?
- My answer: good if model makes **useful** predictions on unknown, **future** feature vectors (it *generalizes*)
- Might not be super accurate, but if it's better than a human can do, might still be useful
- We measure how close predictions are to known true responses, but on data <u>not used to train model</u>
- If inaccurate on training data, model is **biased**
- If inaccurate on test set, model doesn't generalize (high **variance**)
- Regressors: R^2, MAE, MSE, RMSE, RMSLE
- Classifiers: accuracy, precision & recall, F1, confusion matrix, …

Train, validate, test

3-fold cross-validation
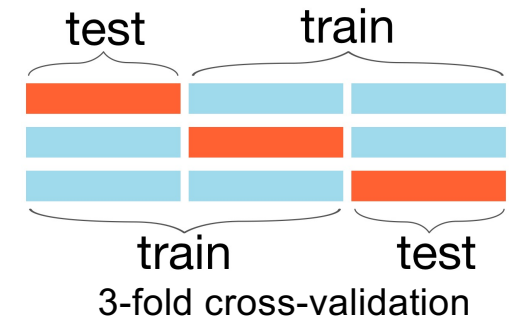
- One of the most important, fundamental ideas in ML
- See "The testing trilogy" in MML book[1]
- 3 sets of observations: *training*, *validation*, and *test* sets
- Model trains on training set; assess and tune with validation set
- NEVER peek at the test set; lock it away as first step
- Assess model w/test set as last step; it's only true measure of generality
- Every change to model after testing with a data set tailors it to that set
- Validation strategies: k-fold CV, hold out, leave-1-out, out-of-bag (RF), …

[1] https://mlbook.explained.ai/bulldozer-testing.html#sec:trilogy

UNIVERSITY OF SAN FRANCISCO

Simplified ML process

- Know what problem you're solving; what is business case?
- Acquire data, do we have everything we need?
- Split into train, validation, test sets
- Sniff data, clean, deal with missing data
- Convert non-numeric features to numeric
- Repeat until satisfied
  - Train a model using training set and specific hyperparameters
  - Tune model (tweak hyperparameters), do feature engineering with validation set
- Last step: assess model performance on test set
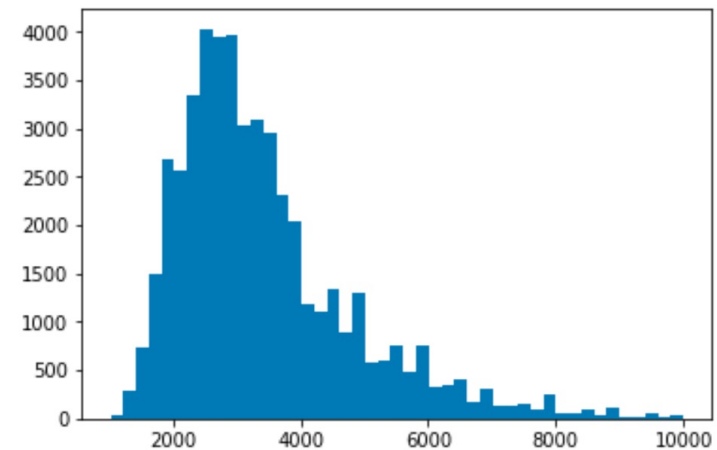
Your model development environment

- See https://mlbook.explained.ai/tools.html
- Pandas, NumPy, matplotlib, scikit-learn (sklearn)
- Jupyter lab (or notebook)
- Install latest Anaconda for Python 3

```
In [4]: import pandas as pd
        import matplotlib.pyplot as plt
        fig, ax = plt.subplots()
        ax.hist(df.price, bins=45)
        plt.show()
```



```
[2]: import pandas as pd
     df = pd.read_csv("data/rent-ideal.csv")
     df.head(5) # print the first 5 rows of data
```
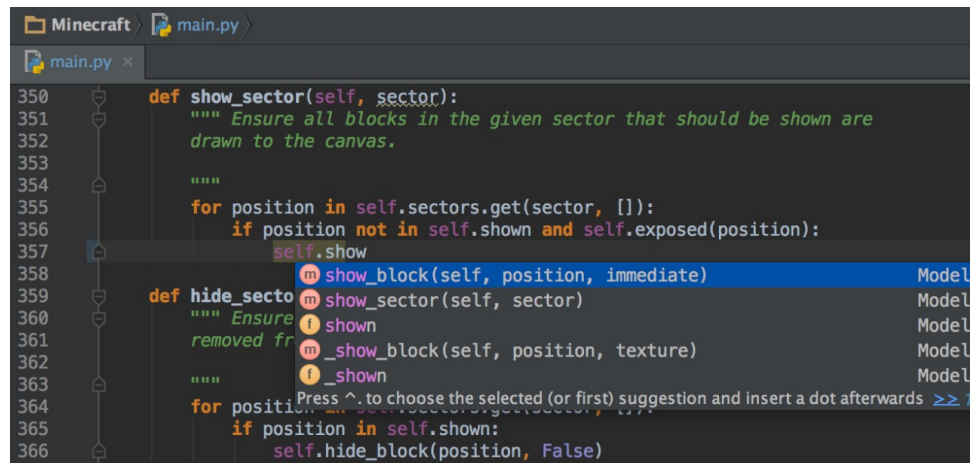
[2]:

	bedrooms	bathrooms	latitude	longitude	price
0	3	1.5	40.7145	-73.9425	3000
1	2	1.0	40.7947	-73.9667	5465
2	1	1.0	40.7388	-74.0018	2850
3	1	1.0	40.7539	-73.9677	3275
4	4	1.0	40.8241	-73.9493	3350

Your library development environment

- You will create separate Python .py scripts and use unit tests as part of the projects
- I recommend PyCharm development environment (free) for creating python files, but you are free to use whatever you like

https://www.jetbrains.com/pycharm/download

UNIVERSITY OF SAN FRANCISCO

Getting started in MSDS621

- We'll start with *regularization* to finish off linear models
- Then we'll try to reinvent some common machine learning models
- As part of this class, you will build up a library of models
- Then, learn how to interpret model results
- Then an intro to deep learning with pytorch
- Etc…

UNIVERSITY OF SAN FRANCISCO