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
- Models (kNN, Naïve Bayes, Decision trees, Random Forests)
- Data clean up, dealing with missing data
- Model assessment (metrics, ROC/PR curves)
- Model interpretation (feature importance, partial dependence)
- k-means clustering

- Course book (in progress):
  The Mechanics of Machine Learning (MML)
  https://mlbook.explained.ai/

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 machine learning

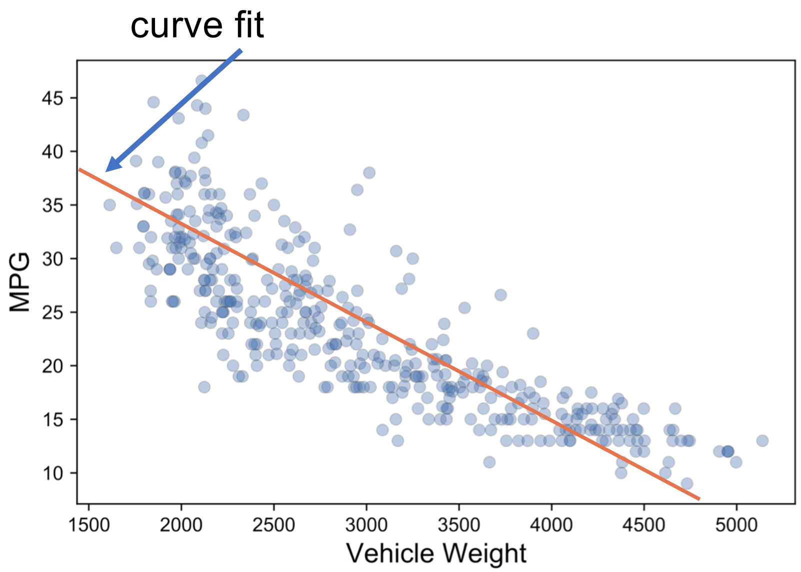
- Build a system that makes accurate future predictions based upon *training data* (X, y) from the past
- BUT, w/o being overly-specific to this training data (don't *overfit*)
- X is *explanatory matrix*, y is the *target* or *response* vector

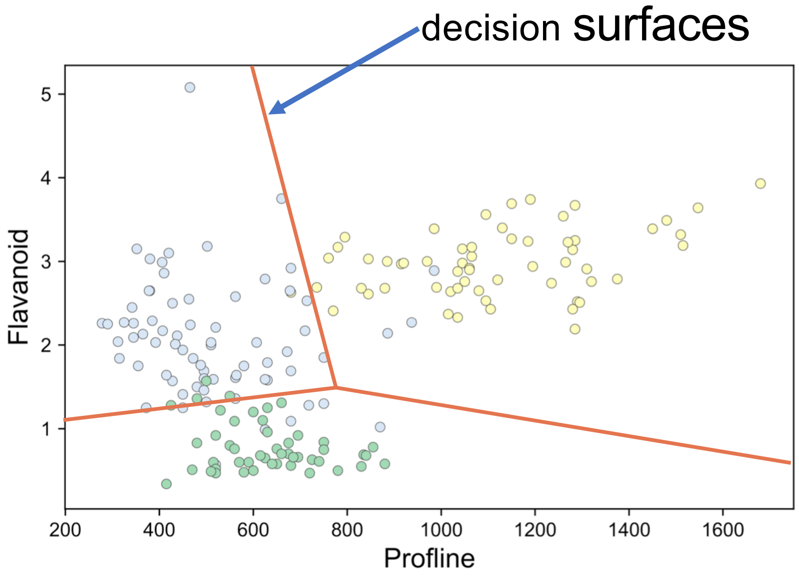
observations or *records* …

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

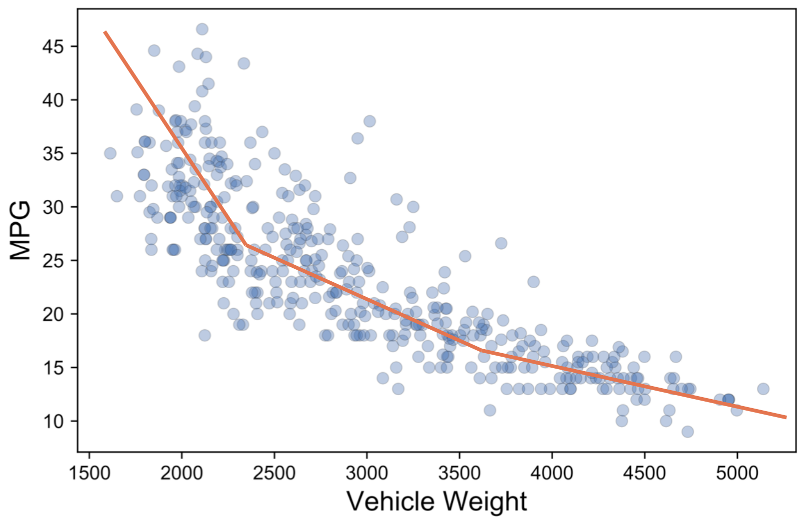
Classifier vs regressor; 2 sides of same coin

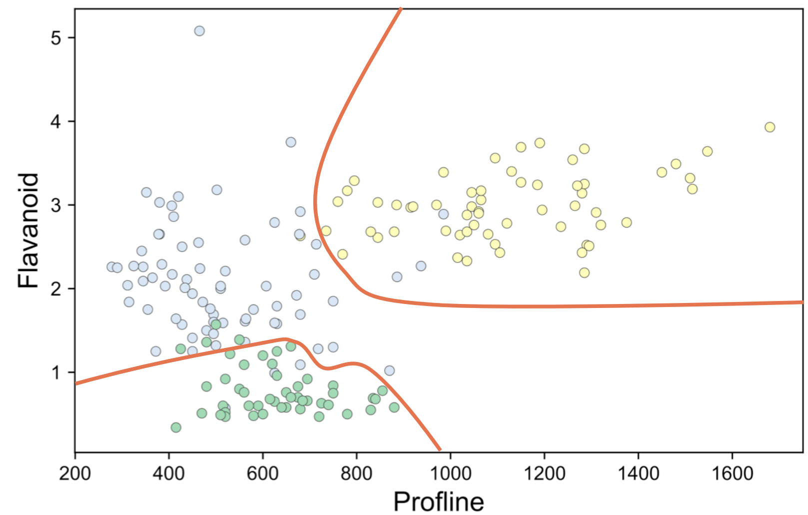
- If target is numerical, model is a *regressor*
- If target is *categorical*, model is a *classifier*
- Regressors go through data, classifier goes between clusters



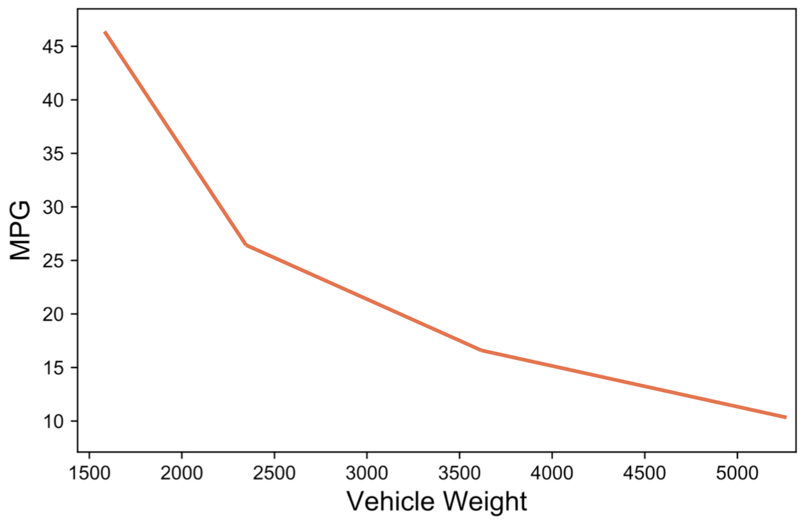


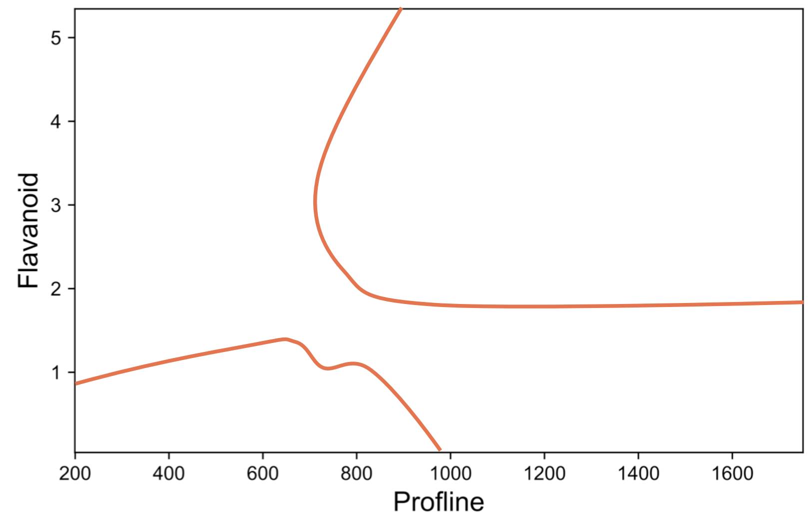
Different models have different surfaces





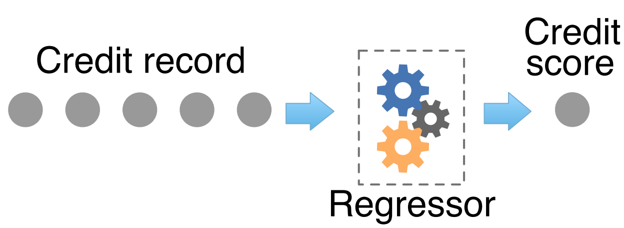
Model is just the decision surface(s)

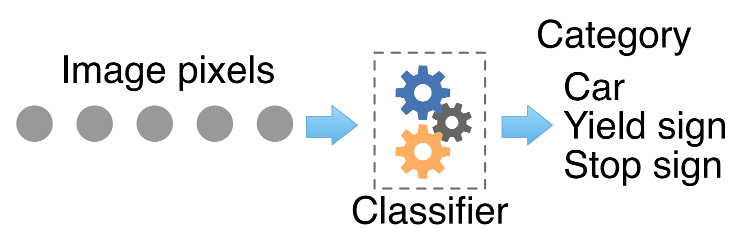




Models make predictions

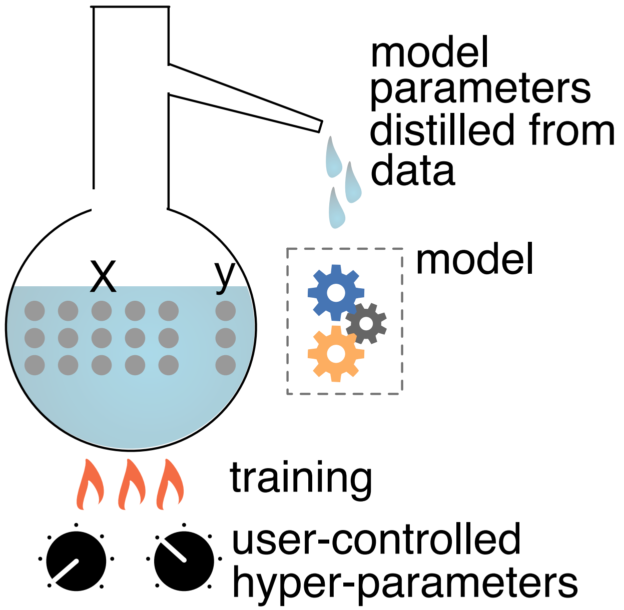
- Feature vectors go in, predictions come out
- Response is either a numeric value or class/category
- Response categories encoded as ints but not treated as numbers





Training (*fitting*) a model

- Distill training data into model *parameters*
- *Hyperparameters* control distillation, *parameters* are the condensate
- Parameters:
  - beta coefficients for linear model
  - trees for decision tree
  - kNN is extreme where data are the parameters
- 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!!



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*, 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

Feature engineering

- Synthesize new features from existing features
- Or, derive from external sources; e.g., isholiday from date
- A few common synthesized features:
  - frequency encoding; e.g., getting info about records from ID
  - 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

What training, prediction look like in code

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

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

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

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

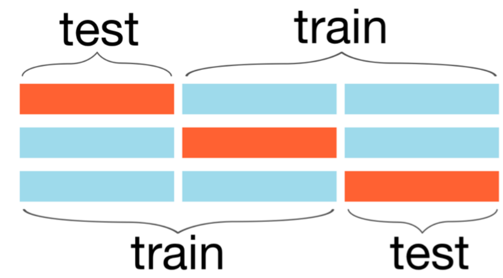
```
x = [[2, 1, ...]]
y_pred = rf.predict(x)
```

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

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*)
- If inaccurate, model is **biased**; tradeoff with generality usually
- Might not be super accurate, but if it's better than a human can do, might be useful nonetheless
- Most commonly we measure how close predictions are to known true responses, but on data <u>not used to train model</u>
- Classifiers: accuracy, precision & recall, F1, confusion matrix, …
- Regressors: R^2, MAE, MSE, RMSE, RMSLE

Train, validate, test



3-fold cross-validation

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

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

ML process

- Know what problem you're solving
- Acquire data
- Sniff data, clean, deal with missing data
- Convert non-numeric features to numeric
- Split into train, validation, test sets
- Repeat until satisfied
  - Train a model using training set
  - Tune model and do feature engineering with validation set
- Last step: assess model performance on a test set

Your 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

[2]:

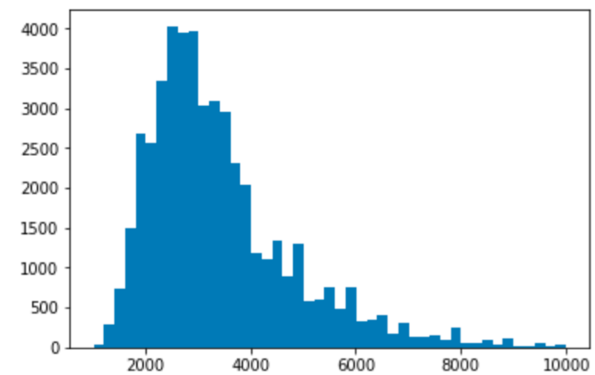
```
import pandas as pd
df = pd.read_csv("data/rent-ideal.csv")
df.head(5) # print the first 5 rows of data
```

[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

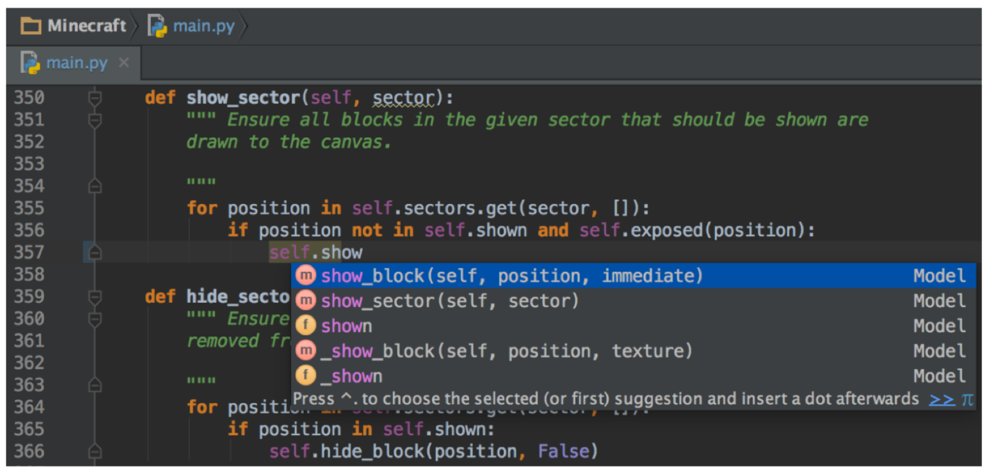
In [4]:

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



But, you will submit scripts not notebooks

- You will learn to create separate Python 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

Getting started in MSDS621

- We'll start with *regularization* to finish off linear models (last class)
- 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, we'll dig into how these models actually work
- Then, learn how to interpret model results
- Etc…