

# QSS20: Modern Statistical Computing

## Unit 12: Supervised Machine Learning, Part I (Intro)

# Goals for this week

## **Today: Introduction to supervised machine learning**

- ▶ Recap of web-scraping
- ▶ Lecture on basics of supervised ML
- ▶ Walkthrough of supervised ML in Python
- ▶ Activity on basic ML

## **Wednesday: Decision Trees and optimizing supervised ML models**

- ▶ Recap of basic supervised ML
- ▶ Overfitting and its solutions
- ▶ Decision Trees
- ▶ Optimization
- ▶ Activity on DTs & optimization

## Pset 3 grade distribution

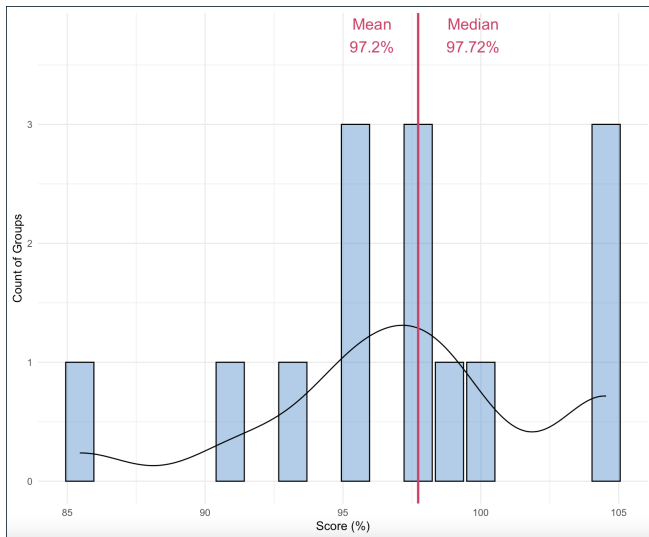


Figure credit: Ramsey

## Remaining four classes

- ▶ W 02/22: Supervised ML part II
  - ▶ Innovation 1: Flipped classroom! See Piazza announcement with link to recorded lecture to **watch before class** (required)
  - ▶ Innovation 2: Ramsey will lead the class in completing the usual sorts of activities (Prof. on honeymoon)
  - ▶ In-class exercise **due by midnight that same day** (no late submissions), so be on task!
- ▶ M 02/27: Introduction to SQL
- ▶ W 03/01: Final project work session
- ▶ M 03/06: Final status-update presentations, each 8-10 mins then 5 mins Q&A—**we will likely use X-hour, please plan to stay through**

# Goals for today

## **Today: Introduction to supervised machine learning**

- ▶ **Recap of web-scraping**
- ▶ Lecture on basics of supervised ML
- ▶ Walkthrough of supervised ML in Python
- ▶ Activity on basic ML

## Wednesday: Decision Trees and optimizing supervised ML models

- ▶ Recap of basic supervised ML
- ▶ Overfitting and its solutions
- ▶ Decision Trees
- ▶ Optimization
- ▶ Activity on DTs & optimization

# Recap of web-scraping

What do you remember?

# Recap of web-scraping

## Tips:

- ▶ Scraping is hard, so check first for existing data or an API
- ▶ HTML exists in **tags** like `<a>` for links and `<p>` for paragraphs
- ▶ Custom scraping uses HTML structure: right-click → Inspect in browser
- ▶ Web-scraping can take one of two basic approaches:
  1. extracting specific content from some pages with a custom, fragile scraper: most common and time-consuming because HTML changes
  2. extracting everything from those pages with a more resilient but messier scraper—filtering may take time; some methods OK with mess)
- ▶ Is scraping 'ethical'? **Usually**—assuming you're polite: scrape slowly, respect privacy, don't share the data

## Useful commands:

```
response = requests.get(url) # get content
soup = BeautifulSoup(response.text) # read content
exclusions = ['meta', 'script', etc.] # define junk elems to avoid
tag.extract() # take tag out, i.e. append to list
tag.decompose() # obliterate junk tag
[tag.decompose() for tag in soup(exclusions)] # iterate over exclusions
```

## Example from last class: Scraping fact check description

Check out this url:

[https://www.politifact.com/factchecks/2021/apr/02/  
citizens-united/  
citizens-united-calls-bidens-infrastructure-plan-g/](https://www.politifact.com/factchecks/2021/apr/02/citizens-united/citizens-united-calls-bidens-infrastructure-plan-g/)

**Challenge:**

Use your browser to inspect the HTML and identify any unique classes that enclose the explanation (and nothing else).



## Example from last class: Scraping fact check description

```
1 # Solution:
2
3 # Set URL to scrape
4 url = 'https://www.politifact.com/factchecks/2021/apr/02/citizens-united/citizens-united-calls-bidens-infrastructure-plan-g/'
5
6 # Scrape HTML with requests and BeautifulSoup
7 response = requests.get(url)
8 soup = BeautifulSoup(response.text)
9
10 explanation = soup.find('article', class_='m-textblock').get_text() # identify this class from looking at HTML
11
12 import re
13 explanation = re.sub(r"\n+", "\n", explanation)
```

# Where we are

## Today: Introduction to supervised machine learning

- ▶ Recap of web-scraping
- ▶ **Lecture on basics of supervised ML**
- ▶ Walkthrough of supervised ML in Python
- ▶ Activity on basic ML

## Wednesday: Decision Trees and optimizing supervised ML models

- ▶ Recap of basic supervised ML
- ▶ Overfitting and its solutions
- ▶ Decision Trees
- ▶ Optimization
- ▶ Activity on DTs & optimization

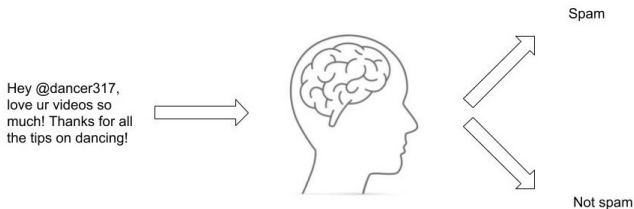
## Example of supervised ML: classifying spam emails

**Which of these are spam, and which aren't?** What do you notice that can help make this distinction?

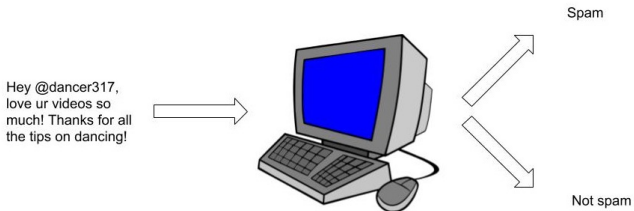
- ▶ *Hey @dancer317, love ur videos so much! Thanks for all the tips on dancing!*
- ▶ *OUR LASER PRINTER/FAX/COPIER TONER CARTRIDGE PRICES NOW AS LOW AS 39 DOLLARS. SPECIALS WEEKLY ON ALL LASER PRINTER SUPPLIES. WE CARRY MOST ALL LASER PRINTER CARTRIDGES, FAX SUPPLIES AND COPIER TONERS AT WAREHOUSE PRICES*
- ▶ *I'm not sold on your first point about crossing national boundaries, but I see what you mean about non-economic alternatives.*
- ▶ *Some of the most beautiful women in the world bare it all for you. Denise Richards, Britney Spears, Jessica Simpson, and many more. [CLICK HERE FOR NUDE CELEBS](#)*

# Example of supervised ML: classifying spam emails

What you just learned to do is text classification!



We want our machines to learn to do this, like so:



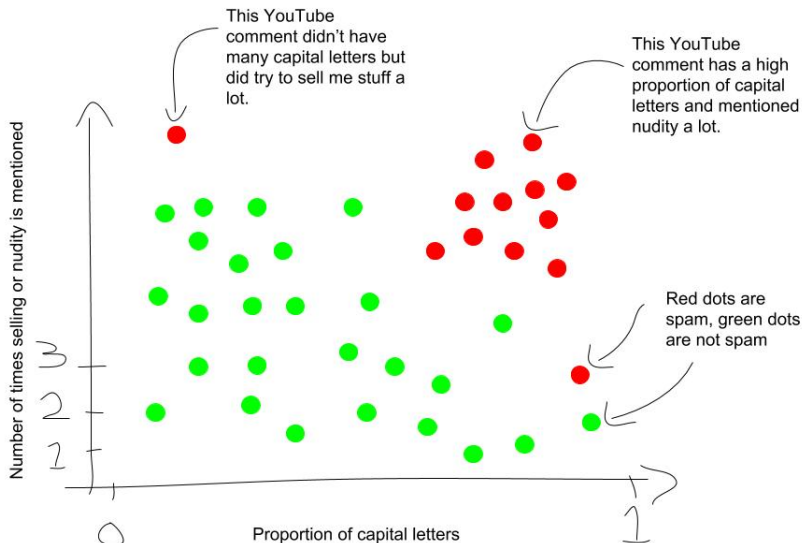
## Example of supervised ML: classifying spam emails

How our machines see the data:

Comment	Selling or nudity	Proportion capital letters	Is it spam?
Hey @dancer317, love ur videos so much! Thanks for ...	0	0.1	No
OUR LASER PRINTER/FAX/COPIER TONER CARTRIDGE PRICES ...	4	1.0	Yes
I'm not sold on your first point ...	1	0.05	No
Some of the most beautiful women in the world ...	3	0.15	Yes

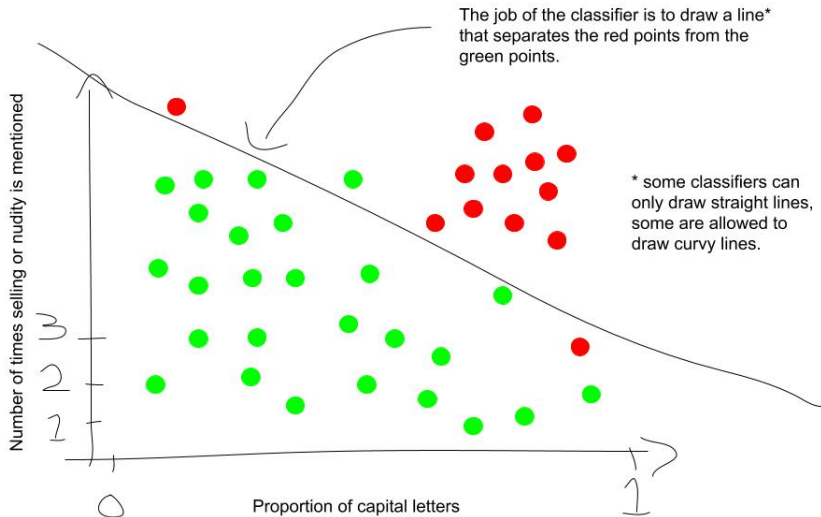
(1) Whether the comment mentions selling or nudity and (2) the proportion of capital letters here are the *features* we want to train a model to recognize when trying to recognize spam. The last column is our human-judged *label*.

## Example cont'd: Plotting our features



Graphics source

## Example cont'd: Plotting our features



Graphics source

# Where has machine learning come up already?

- ▶ **Sentiment classification:** VADER scores sentiment using a dictionary, but dictionaries work more or less well depending on the context
  - ▶ **Solution:** supervised machine learning where we:
    1. Take data labeled with positive/negative sentiment (binary) or sentiment score (continuous)
    2. Create a document-term matrix
    3. Predict sentiment using the terms as predictors
    4. Use that to score the sentiment of unlabeled data
- ▶ **Topic modeling:** LDA is a form of unsupervised machine learning that takes an unlabeled input—a document-term matrix—and returns high probability topics and words



# Why do supervised ML?

- ▶ Once we have a good number of labeled texts (say 200-500 cases), can use supervised ML to train a model to predict the labels
- ▶ **Two benefits:**
  1. Identifies what features (e.g., words) are most defining of each category
  2. Can scale up almost indefinitely to predict labels for unobserved cases
- ▶ **Main goal:** *generalization*, or a model that predicts the response on novel inputs/new data it hasn't yet seen (e.g., new or future cases)
- ▶ **Two main types:**
  1. **Regression:** predict numeric values (not the focus this week)
  2. **Classification:** predict categories
    - ▶ **Binary classification:** predict yes/no or True/false category, e.g.: fraudulent or not; experiencing homelessness or not; positive sentiment or not (focus this week)
    - ▶ **Multiclass** > 2 categories

# What is 'supervised machine learning?': Introductory terminology

- ▶ **Supervised machine learning:** Building a model to infer classifications based on features. Learning a function that maps features to outputs.
- ▶ **Unsupervised versus supervised learning:** unsupervised learning is a form of dimension reduction/clustering on **unlabeled data**; supervised learning is used for prediction on **labeled data**
- ▶ **Labeled data:** is associated with outcome/result we are interested in, also called the 'target' (in CS) or the 'dependent variable' (social science). For ex: pos/neg sentiment, spam/not spam, voting yes/no,
- ▶ **Features/predictors:** what we use to predict the label, also called 'covariates' or 'independent variables' (in social science). For images, features are often pixels. For text, features could be word counts or weighted word counts, parts of speech, proportion of capitals, etc.

## Four DataCamp chapters

1. **Classification:** uses k-nearest neighbors; in activities, we'll shift to other classifiers (ridge; LASSO; Decision Tree; Random Forest)
2. **Regression:** covers K-fold cross-validation and ridge/lasso
3. **Fine-tuning your model:** ROC curve with different  $\hat{y}$  thresholds for continuous logistic regression predictions; hyperparameter tuning with GridSearchCV and RandomizedSearchCV
4. **Preprocessing and pipelines:** transforming categorical features into dummy indicators; imputation; standardizing inputs

# DataCamp: pros/cons of their approach

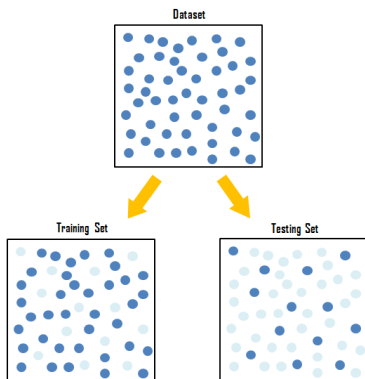
## ► What I liked:

- Emphasis on importance of preprocessing for model accuracy
- Start with a model where the hyperparameters (e.g.,  $\alpha/\lambda$  in LASSO/ridge) are fixed and move to models where we iterate over a grid of hyperparameters and choose the one that maximizes eval metric

## ► What I liked less:

- **Reliance on black-box functions:** rather than using functions like `train_test_split` and `score`, it's more transparent to do these steps ourselves (since we're already black-boxing the algorithms):
  - **Splitting:** can use `random` module; increased flexibility to (1) balance features/outcomes across the splits; (2) account for temporal or row-based clustering (eg if we observe a person or employer multiple times in the data, want all their rows in one split)
  - **Evaluation:** calculate metrics using  $\hat{y}$  (predicted response; binary or continuous 0-1) versus  $y$
- **Rationale for imputation:** less about “losing data”; more about biases if rows missing some values differ systematically from those w/ complete data

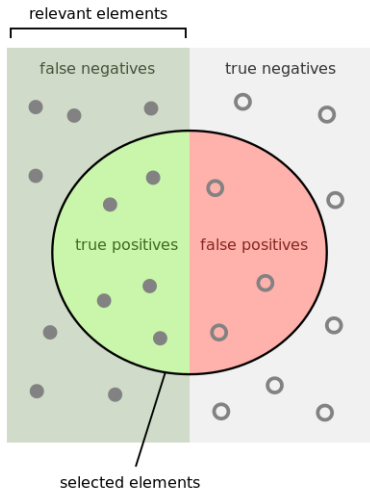
# Key task for ML: Train-test split



## What split to use?

- ▶ 80/20 rule is common (see [Pareto principle](#)), but depends on the amount and complexity of your data and the complexity of your model
- ▶ with more or simpler data and/or simpler models, you can afford a smaller test set

# Evaluating ML models: precision and recall



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

# Evaluating ML models: how to compute metrics

**Accuracy:**

$$\frac{\sum \text{true positives} + \sum \text{true negatives}}{\sum \text{total population}}$$

**Precision:**

$$\frac{\sum \text{true positives}}{\sum \text{predicted positives}}$$

**Recall:**

$$\frac{\sum \text{true positives}}{\sum \text{condition positives}}$$

**F1-score:**

$$\frac{\sum \text{true positives}}{\sum \text{true positives} + 1/2(\sum \text{false positives} + \sum \text{false negatives})}$$

# Evaluating ML: terminology

- ▶ **Train-test split:** Dividing labeled data into a training set and a test set. Often done once in supervised machine learning model training, but can be done several times independently as in cross-validation (see Wednesday)
- ▶ **Training set:** A selection of labeled data that is used to train the machine learning algorithm
- ▶ **Test set:** A selection of labeled data that is used to test the accuracy of the machine learning algorithm
- ▶ **Accuracy:** The ratio of true positives and true negatives (the numerator) to the total number of cases (the denominator)
- ▶ **Precision:** How close a models' guesses are to the mark; the ratio of true positives to predicted positives
- ▶ **Recall:** How many actual positives a model guessed right; the ratio of true positives to condition positives



# Math behind today's model: logistic regression (logit)

Logistic regression models fit where the probability of being positive ( $y = 1$ ) is described by a sigmoid function of the form:

$$f(X) = P(y = 1|X) = \frac{\exp(-X'\theta)}{1 + \exp(-X'\theta)} \quad (1)$$

Once  $\theta$ —a vector of word *weights* or *loadings*—is estimated, we can predict outcomes  $\hat{y}$  conditional on observed word count  $X$ .

If  $\hat{y} > 0.5$ , then the observation is classified as positive. This is a *linear classifier* as the decision boundary is defined by  $\frac{\exp(-X'\theta)}{1 + \exp(-X'\theta)} = 0.5$ , which after rearranging and taking logs appears equivalent to  $-X'\theta = 0$ —a linear function of the features  $X$ .

This uses penalized Maximum Likelihood as a loss function (estimates inaccuracy in relationship between  $X$  and  $y$ )

# Where we are

## Today: Introduction to supervised machine learning

- ▶ Recap of web-scraping
- ▶ Lecture on basics of supervised ML
- ▶ **Walkthrough of supervised ML in Python**
- ▶ Activity on basic ML

## Wednesday: Decision Trees and optimizing supervised ML models

- ▶ Recap of basic supervised ML
- ▶ Overfitting and its solutions
- ▶ Decision Trees
- ▶ Optimization
- ▶ Activity on DTs & optimization

# Example classification task

## Predicting positive sentiment (1 = positive) using text of Yelp reviews

metadata_label	raw_text
0	<p>Unfortunately, the frustration of being Dr. Goldberg's patient is a repeat of the experience I've had with so many other doctors in NYC -- good doctor, terrible staff. It seems that his staff simply never answers the phone. It usually takes 2 hours of repeated calling to get an answer. Who has time for that or wants to deal with it? I have run into this problem with many other doctors and I just don't get it. You have office workers, you have patients with medical needs, why isn't anyone answering the phone? It's incomprehensible and not work the aggravation. It's with regret that I feel that I have to give Dr. Goldberg 2 stars.</p>
1	<p>Been going to Dr. Goldberg for over 10 years. I think I was one of his 1st patients when he started at MHMG. He's been great over the years and is really all about the big picture. It is because of him, not my now former gyn Dr. Markoff, that I found out I have fibroids. He explores all options with you and is very patient and understanding. He doesn't judge and asks all the right questions. Very thorough and wants to be kept in the loop on every aspect of your medical health and your life.</p>
0	<p>I don't know what Dr. Goldberg was like before moving to Arizona, but let me tell you, STAY AWAY from this doctor and this office. I was going to Dr. Johnson before he left and Goldberg took over when Johnson left. He is not a caring doctor. He is only interested in the co-pay and having you come in for medication refills every month. He will not give refills and could less about patients's financial situations. Trying to get your 90 days mail away pharmacy prescriptions through this guy is a joke. And to make matters even worse, his office staff is incompetent. 90% of the time when you call the office, they'll put you through to a voice mail, that NO ONE ever answers or returns your call. Both my adult</p>

Labeled Yelp data (subsample of 15k): [Zhang, Zhan, Lecun, 2015. "Character-level Convolutional Networks for Text Classification"](#)

# Notebook to follow along

[08\\_ML\\_intro\\_activity\\_blank.ipynb](#)

## Step 1: preprocess the data

See [slides here](#) for process of document-term matrix creation

# Step 1: terms versus negative label

## Uses word “frustrat” and negative label:

metadata_label	raw_text	frustrat
0	Say \"BITES.\" Say \"WINGS.\" Apparently, my waiter did not know the difference between \"WINGS\" and \"BITES,\" because when my roommate and I ordered buffalo BITES, we got WINGS instead. We hate wings and love boneless buffalo BITES. It was even more frustrating when our buddy came to join us at happy hour and the waiter spoke for two minutes about how he prefers the BITES over the WINGS. I know I probably sound dramatic, but I will NOT be back...	1
0	This used to be my \"go to\" place for groceries. However, lately I'm becoming more and more frustrated by the store. If you go between 4pm and 6pm on a weekday or at any time on Saturday or Sunday, be prepared to navigate a mess of a parking lot, an over crowded store, and a 20 minute wait in a checkout lane. On a Sunday, it's impossible to find parking even though there is outdoor parking as well as a 2 story garage. Once you get in the store, you have to navigate through an obstacle course of a produce section and around overwhelmed first-timers who are trying to figure out where everything is. It's entirely too small for a place that markets itself towards the \"fresh foods\" end of things (as opposed to a regular Giant Eagle). Then you have to figure out where everything is in the aisles. I get the impression that things were just randomly arranged around the store. Once you actually have your groceries, the waiting begins...	1

# Step 1: terms versus positive label

## Uses word “frustrat” and positive label:

metadata_label		raw_text	frustrat
1		<p>This review is based on this location's truck delivery alone. I jumped on their 50% off patio furniture sale last month and I took advantage of their delivery for \$56. As long as you order everything online, they will deliver it all for that one fee...even if they have to make multiple trips. Score!\n\nIf that isn't fantastic enough, when the delivery guys came, they even took extra time to put together my furniture. Seriously, that saved me hours of work and frustration. They were so nice to do it and they took the cardboard boxes with them. \n\nThe managers that called to schedule the deliveries or confirm a return were so courteous! They didn't waste any time getting me my items. t could not have asked for a more smooth process and enjoyable online purchase!</p>	1
1		<p>Perry has been cutting my hair for about 4 to 5 years. Whenever I walk out of there I cannot believe how my hair looks, and I often get complete strangers complimenting my cut. Perry does a great job of taking my preferences and developing cuts that flatter me. What more can I ask?\n\nThe massage before the shampoo is standard and more than once I've fallen asleep. Once a couple years ago, in the late afternoon of a particularly frustrating day at work, they asked if I wanted anything to drink. I said, only if you have vodka. Next thing I know, a clear drink on ice shows up...and tasted suspiciously like vodka. No that's</p>	1

## Step 2: split into train and test set (more manual; notebook also has `train_test_split`)

```
1 ## define the number of rows in training set (20\% test)
2 ## and sample
3 nrows_train = round(X.shape[0]*0.8)
4 nrows_test = X.shape[0] - nrows_train
5 random.seed(221)
6 train_ids = random.sample(set(X['metadata_rowid']),
7                             nrows_train)
8 ## function fed ids and name of id col
9 def my_split(train_ids, id_col):
10     test_ids = set(X[id_col]).difference(train_ids)
11     X_train_man = X[X[id_col].isin(train_ids)].copy()
12     X_test_man = X[X[id_col].isin(test_ids)].copy()
13     y_train_man = y[y.index.isin(train_ids)].iloc[:,
14                                                         0].to_numpy()
15     y_test_man = y[y.index.isin(test_ids)].iloc[:,
16                                                         0].to_numpy()
17     return(X_train_man, X_test_man, y_train_man, y_test_man)
```



## Step 3: estimate model in training data (hardcode hyperparameters for now)

```
1 ## list of nonfeatures
2 non_feat = ['metadata_rowid', 'raw_text', 'process_text']
3 ## initialize classifier
4 logit_lasso = LogisticRegression(penalty = "l1",
5     max_iter=100, C = 0.01, solver='liblinear')
6 ## estimate the model on training data/training label
7 logit_lasso.fit(X_train_man[[col for col in
8     X_train.columns if col not in
9     non_feat]], y_train_man)
```

### Breaking things down:

- ▶ `non_feat`: want to shield things like our id variable from being treated as a feature; keep it in matrix for later post-modeling dx
- ▶ `logit_lasso`: initialize the logistic regression classifier; this is a `LogisticRegression` class within `sklearn`; telling it to use level-1 regularization (lasso), 'liblinear' solver (good for small datasets)
- ▶ `logit_lasso.fit()`: use the `fit()` method from class; feed it training data and training label
- ▶ **Note**: skipped standardization bc features already on common scale

## Step 4: apply the model to test set to generate binary and continuous predictions

```
1 y_pred = logit_lasso.predict(X_test_man[[col for col
2         in X_test_man.columns if "metadata_rowid"
3         not in col]])
4 y_predprob = logit_lasso.predict_proba(X_test_man[[col for col
5         in X_test_man.columns if "metadata_rowid"
6         not in col]])
```

Breaking things down:

- ▶ `predict`: predicted binary class (default cutoff  $\sim 0.5$ )
- ▶ `predict_proba`: continuous predicted probability (0-1 range for classification)
- ▶ **To discuss**: why don't I need `y_test` at this stage?
- ▶ **To discuss**: what needs to be the same about the training and test set in order for me to generate these predictions?

## Output of step 4: one or two-D array with predicted labels or probabilities

```
: ## print the results  
y_pred[0:5]  
y_predprob[0:5]  
  
: array([0, 0, 1, 0, 0])  
  
: array([[0.70544662, 0.29455338],  
        [0.70544662, 0.29455338],  
        [0.1918429 , 0.8081571 ],  
        [0.70544662, 0.29455338],  
        [0.70544662, 0.29455338]])
```

## Step 5: use the $\hat{y}$ from step 4 to evaluate accuracy

```
1 ## dataframe of binary, continuous, and true labels
2 y_pred_df = pd.DataFrame({'y_pred_binary': y_pred,
3                             'y_pred_continuous':
4                                 [one_prob[1] for one_prob in y_predprob],
5                                 'y_true': y_test_man})
6
7 ## use np.select to code each obs to its error cat
8 error_cond = [ (y_pred_df['y_true'] == 1) &
9                 (y_pred_df['y_pred_binary'] == 1),
10                (y_pred_df['y_true'] == 1) &
11                (y_pred_df['y_pred_binary'] == 0),
12                (y_pred_df['y_true'] == 0) &
13                (y_pred_df['y_pred_binary'] == 0)]
14 error_codeto = ["TP", "FN", "TN"]
15
16 y_pred_df['error_cat'] = np.select(error_cond,
17                                     error_codeto, default = "FP")
18
19 ## see notebook for calculations using the error_cat column
```

## Step 6: interpret the model

```
1 feat_imp = pd.DataFrame({'coef': logit_lasso.coef_[0],  
2                           'feature_name':  
3                           [col for col in X_train.columns  
4                           if col not in non_feat]}))
```

Breaking it down:

- ▶ `coef_` is an attribute of the estimated logit model
- ▶ Feature importance in this model has both magnitude and direction, so we can tell from the sign whether an “important” feature strongly predicts positive sentiment (1) or negative sentiment (0)
- ▶ **Note:** In contrast, tree-based feature importances (we’ll see these next class; e.g., `dt.feature_importances_`) has **magnitude but not direction**

# Where we are

## Today: Introduction to supervised machine learning

- ▶ Recap of web-scraping
- ▶ Lecture on basics of supervised ML
- ▶ Walkthrough of supervised ML in Python
- ▶ **Activity on basic ML**

## Wednesday: Decision Trees and optimizing supervised ML models

- ▶ Recap of basic supervised ML
- ▶ Overfitting and its solutions
- ▶ Decision Trees
- ▶ Optimization
- ▶ Activity on DTs & optimization

## Break for activity (section 4 in notebook)

Notebook: [https://github.com/jhaber-zz/QSS20\\_public/blob/main/activities/08\\_ML\\_intro\\_activity\\_blank.ipynb](https://github.com/jhaber-zz/QSS20_public/blob/main/activities/08_ML_intro_activity_blank.ipynb)

1. Read the documentation here to initialize a ridge regression (l2 penalty)- you can use the same cost parameter (C) and number of iterations as in the lasso example above: [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
2. Fit the model using `X_train_main`, `y_train_main`
3. Generate binary and continuous predictions
4. Create a function that takes in a dataframe of binary predictions and true labels and calculates the  $F_1$  score using this formula:

$$F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + 0.5(FP + FN)}$$

5. Apply that function to calculate the F1 score for the decision tree and lasso (from above), and ridge regression (from the activity)
6. **Challenge exercise:** parametrize the model fitting with a function that takes in a classifier as an argument and returns coefficients or feature importances and certain eval metrics (eg precision, recall, and F1)

# QSS20: Modern Statistical Computing

Unit 12: Supervised Machine Learning, Part II (Decision Trees & Optimization)



# Learning goals

- ▶ Get comfortable with vocabulary of and rationale for model optimization
- ▶ Learn the basic mechanics of a few machine learning models: logistic regression, decision tree, and random forest models
- ▶ Understand cross-validation for model evaluation and how to interpret
- ▶ Gain experience implementing, comparing, and optimizing supervised machine learning models
- ▶ Get familiar with visualization to compare model performance

# Goals for today

Monday: Introduction to supervised machine learning

- ▶ Recap of web-scraping
- ▶ Lecture on basics of supervised ML
- ▶ Walkthrough of supervised ML in Python
- ▶ Activity on basic ML

**Today: Decision Trees and optimizing supervised ML models**

- ▶ Recap of basic supervised ML
- ▶ Overfitting and its solutions
- ▶ Decision Trees
- ▶ Optimization
- ▶ Activity on DTs & optimization

# Goals for today

Monday: Introduction to supervised machine learning

- ▶ Recap of web-scraping
- ▶ Lecture on basics of supervised ML
- ▶ Walkthrough of supervised ML in Python
- ▶ Activity on basic ML

**Today: Decision Trees and optimizing supervised ML models**

- ▶ **Recap of basic supervised ML**
- ▶ Overfitting and its solutions
- ▶ Decision Trees
- ▶ Optimization
- ▶ Activity on DTs & optimization

# Recap of supervised machine learning

What do you remember?

## Recap of supervised machine learning: Tips

- ▶ Supervised ML builds a model on **features** (e.g., words) to predict a **label** (e.g., spam/not spam)
- ▶ Benefits: Scalable, flexible, can tell you what features matter most
- ▶ Do you care more about avoiding false negatives (e.g., detecting cancer) or false positives (e.g., detecting spam)? The former is a case for recall, the latter precision
- ▶ It often pays to use multiple metrics and try multiple hyperparameters (optimization)
- ▶ ML models try to avoid both overfitting (for generalizability) and underfitting (for accuracy); regularized models can strike a balance
- ▶ Splitting labeled data into training set (to build model) and test set (to evaluate it) reduces risk of overfitting; cross-validation even better

# Recap of supervised machine learning: Steps with code

1. Preprocess the data—with text data, this often means creating a DTM

```
vectorizer = CountVectorizer(lowercase = True)
dtm_sparse = vectorizer.fit_transform(list_of_strings)
```
2. Split into train and test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size)
```
3. Estimate model on training data

```
logit_ridge = LogisticRegression(C = 0.01, solver='liblinear')
logit_ridge.fit(X_train, y_train)
```
4. Apply model to get predictions

```
y_pred = logit_ridge.predict(X_test)
y_predprob = logit_ridge.predict_proba(X_test)
```
5. Evaluate model performance with metric(s) of choice

```
sklearn.metrics.accuracy_score(y_pred, y_test) # or metrics.f1_score
```
6. Interpret model to identify important features

```
ridge_coef = pd.DataFrame({'coef': logit_ridge.coef_[0],
                           'feature_name': [col for col in X_train.columns]})
```

# Where we are

Monday: Introduction to supervised machine learning

- ▶ Recap of web-scraping
- ▶ Lecture on basics of supervised ML
- ▶ Walkthrough of supervised ML in Python
- ▶ Activity on basic ML

**Today: Decision Trees and optimizing supervised ML models**

- ▶ Recap of basic supervised ML
- ▶ **Overfitting and its solutions**
- ▶ Decision Trees
- ▶ Optimization
- ▶ Activity on DTs & optimization

# A big problem with ML: Overfitting

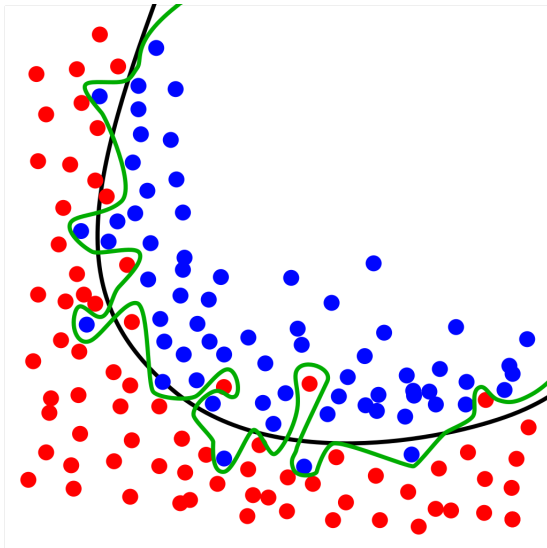
Consider a traditional linear or logistic regression with two predictors trying to predict a sentiment label:

$$\text{positive sentiment}_i = \alpha + \beta_1 \times \text{uses word great} (1 = \text{yes})_i + \beta_2 \times \text{uses word terrible} (1 = \text{yes})_i + \epsilon_i$$

What would happen if we tried to scale this to a large corpus of text data? How many terms might we end up including? How might that number compare to the number of observations?



# A big problem with ML: Overfitting



# A big problem with ML: Overfitting

**The problem:** When a model gets overly complex (e.g., the number of terms modeled exceeds the number of observations), we risk **overfitting** because the model would pick up on language use or other idiosyncratic trends/noise particular to some observed cases but meaningless outside the observed data

For example, if you trained a linear model on reviews of fantasy movies, it might decide that the word 'hilarious' associates with negative reviews while the phrase 'hobbit\_feet' indicates a positive review.

Would these words still be meaningful if you used the model you just trained to make predictions on a *broader corpus of movies*?

# Solutions to overfitting

Two tools help us establish robust patterns/avoid overfitting:

1. **More flexible classifiers than ordinary least squares:**  $y = f(x)$ , where  $f$  is a function mapping some input (e.g., a document-term matrix) to a label ( $y$ ); go beyond OLS or generalized linear models (GLM) for the mapping function
  - ▶ One way to do this: **Regularization** to prevent models from overfitting on training data/decrease their variance or sensitivity to noise (by adding a penalty term for model complexity; read more [here from scikit-learn](#) or from [this blog](#))
  - ▶ Another way: **Tree-based methods** to learn non-linear relationships
2. **Sample splitting:** estimate/train the model on one random split of the data, then evaluate its performance on a separate split
  - ▶ Can do this split once (like we did last class) or many times and average the result, called **"k-fold cross-validation"**

# Where we are

Monday: Introduction to supervised machine learning

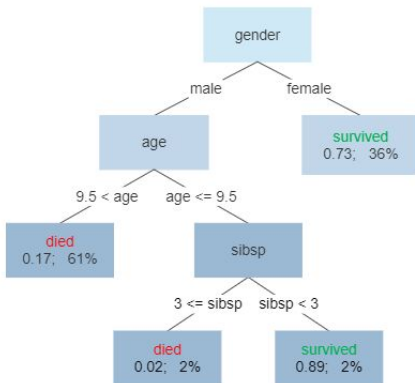
- ▶ Recap of web-scraping
- ▶ Lecture on basics of supervised ML
- ▶ Walkthrough of supervised ML in Python
- ▶ Activity on basic ML

**Today: Decision Trees and optimizing supervised ML models**

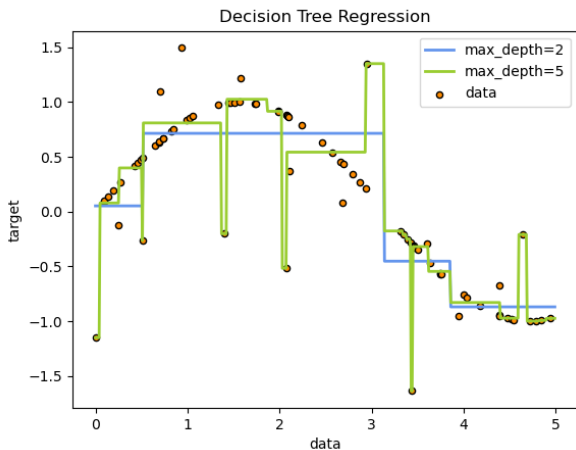
- ▶ Recap of basic supervised ML
- ▶ Overfitting and its solutions
- ▶ **Decision Trees**
- ▶ Optimization
- ▶ Activity on DTs & optimization

# Tree-based methods use if-then-else decision rules

## Survival of passengers on the Titanic



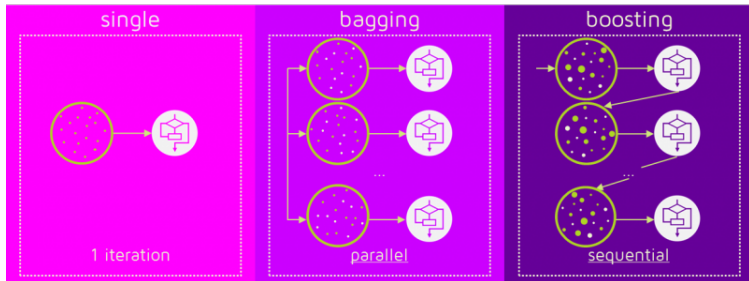
# Tree-based methods learn non-linear relationships



# Logic behind classification trees

- ▶ **Classification and regression trees (CART):** take an input space—in the case of the Titanic, one comprised of gender; age; number of family on ship—and recursively partition that space into regions with different local models (in the Titanic case, binary prediction of survival or mortality)
- ▶ **Decision Trees** learn simple decision rules from the data features
  - ▶ A decision tree is made up of *decision nodes* (which identify feature values) and *leaf nodes* (which assign labels)
  - ▶ The former checks input features and selects a branch (usually this is binary: True/False), starting with the *root node* and continuing until arriving at a leaf node, which stops the evaluation and assigns a label
- ▶ **Random forests:** randomly sample from rows (observations) and columns (inputs/predictors), fit many trees, and **take average over predictions (bagging)** to aggregate over the trees; slower but less overfitting

# “Bagging” vs. “boosting” with classification trees



- ▶ Random forest is an example of the **bagging** approach to ensemble learning, where we fit many trees in parallel with subsets of observations/predictors
- ▶ Another form of ensemble learning is **boosting**, or a **sequential** process where we start with a shallow or “weak learner” (basically, something only slightly better than random guess) to predict the outcome, examine the predictions or residuals, and target misclassification in next iteration

Diagram source QuantDare



# Where we are

Monday: Introduction to supervised machine learning

- ▶ Recap of web-scraping
- ▶ Lecture on basics of supervised ML
- ▶ Walkthrough of supervised ML in Python
- ▶ Activity on basic ML

**Today: Decision Trees and optimizing supervised ML models**

- ▶ Recap of basic supervised ML
- ▶ Overfitting and its solutions
- ▶ Decision Trees
- ▶ **Optimization**
- ▶ Activity on DTs & optimization

# Optimization from last class: code

```

1 def one_las(one_c):
2     """takes cost parameter, estimates model, returns preds"""
3     one_lasso = LogisticRegression(penalty = "l1", max_iter=100,
4                                     C = one_c, solver='liblinear')
5     one_lasso.fit(X_train_man[[col for col in X_train.columns if
6                                 col not in non-feat]], y_train_man)
7     ...
8     return(y_pred_df)
9
10 all_pred = [one_las(one_c) for one_c in c_list] # bind to DF
11 all_pred_df = pd.concat(all_pred)
12
13 def score_onedf(one_c, all_c): # score one cost level
14     one_df = all_c[all_c.cost == one_c].copy()
15     prec_onec = precision_score(
16         one_df['y_true'], one_df['y-pred'], zero_division = 0)
17     return(prec_onec)
18
19 c_list = np.linspace(4, 0.0001, 10) # define range to search
20 pd.DataFrame({'cost': c_list,
21               'precision': [score_onedf(one_c, all_pred_df)
22                             for one_c in c_list]})

```

# Optimization from last class: output

	<b>cost</b>	<b>precision</b>
<b>0</b>	4.000000	0.860497
<b>1</b>	3.555567	0.860193
<b>2</b>	3.111133	0.860096
<b>3</b>	2.666700	0.863260
<b>4</b>	2.222267	0.863889
<b>5</b>	1.777833	0.860996
<b>6</b>	1.333400	0.861188
<b>7</b>	0.888967	0.864232
<b>8</b>	0.444533	0.867495
<b>9</b>	0.000100	0.000000

## More automated approach to optimization: gridsearch

```
1 # define dictionary with possible model parameters
2 param_grid = {'min_samples_split': [2,10],
3               'min_samples_leaf': [2,10]}
4
5 # define gridsearch object
6 model_dt = GridSearchCV(dt_classifier ,
7                           param_grid ,
8                           cv=3,
9                           return_train_score=True)
10
11 # fit model
12 model_dt.fit(X_train , y_train)
13
14 # find best performing candidate model over gridsearch
15 best_index = np.argmax(
16     model.cv_results_["mean_train_score"])
```

# K-fold cross-validation

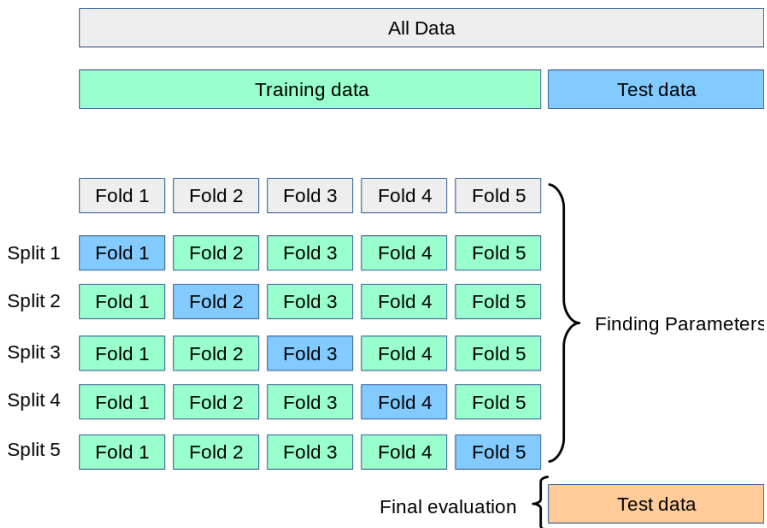
Instead of building a model from the training data in one swoop,  $k$ -fold cross-validation splits the training data into  $k$  smaller sets or "folds", trains a separate model using each fold, and averages the result ([read more](#)). In other words, the procedure is:

- ▶ a model is trained using  $k - 1$  of the folds as training data
- ▶ the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy)
- ▶ Performance of these  $k$  models is averaged to get a better estimation of model accuracy than a single split would give

## Advantages:

- ▶ reduces risk of overfitting due to idiosyncrasies of one random split
- ▶ maximizes our available data; don't need to split into train/test/validation—this means more signal available and less risk of underfitting as well

# K-Fold Cross-Validation



# Optimizing ML: terminology

- ▶ **overfitting:** When a model is trained on too little data to generalize to new data, usually when trained and tested on the same sample. In this case, the model just repeats the labels of the samples that it has just seen, and is evaluated to have a perfect score—but it fails to predict anything useful on yet-unseen data.
- ▶ **underfitting:** When a model fails to adequately capture the underlying structure of the data, usually when trained on too little data
- ▶ **cross-validation:** A technique for assessing model performance using repeated train-test splits on multiple slices of a training set. Simulates model performance on unseen data
- ▶ **parameter:** An algorithm setting that may be selected for performance or substantive reasons. If a parameter is not learned directly within an estimator but instead must be set explicitly (passed as an argument), then it's called a *hyper-parameter*.
- ▶ **optimization:** Selecting the best element from some set of alternatives such that the result maximizes some criterion (e.g., model accuracy)
- ▶ **grid-search:** An exhaustive search over model parameters to discover the optimal configuration, maximizing model performance

# Where we are

Monday: Introduction to supervised machine learning

- ▶ Recap of web-scraping
- ▶ Lecture on basics of supervised ML
- ▶ Walkthrough of supervised ML in Python
- ▶ Activity on basic ML

**Today: Decision Trees and optimizing supervised ML models**

- ▶ Recap of basic supervised ML
- ▶ Overfitting and its solutions
- ▶ Decision Trees
- ▶ Optimization
- ▶ **Activity on DTs & optimization**



# Coding activity

**Notebook:** [https://github.com/jhaber-zz/QSS20\\_public/blob/main/activities/08\\_ML\\_optimization\\_activity\\_blank.ipynb](https://github.com/jhaber-zz/QSS20_public/blob/main/activities/08_ML_optimization_activity_blank.ipynb)