

# Data Science Cribsheet



# Contents

<b>1</b>	<b>Workflow</b>	<b>5</b>
1.1	Preamble . . . . .	5
1.2	Annotated Checklist . . . . .	5
<b>2</b>	<b>Import &amp; Tidy</b>	<b>9</b>
2.1	Import . . . . .	9
2.2	Tidy . . . . .	10
<b>3</b>	<b>Transform &amp; Visualize</b>	<b>11</b>
3.1	Transform . . . . .	11
3.2	Visualize . . . . .	12
<b>4</b>	<b>Model</b>	<b>13</b>
4.1	General . . . . .	13
4.2	Pre-processing . . . . .	15
4.3	Feature Engineering . . . . .	15
4.4	Feature Selection . . . . .	16
4.5	Other . . . . .	16
<b>5</b>	<b>Model</b>	<b>19</b>
5.1	Linear Regression . . . . .	19
5.2	Ridge/Lasso . . . . .	23
5.3	Logistic Regression . . . . .	23
5.4	Poisson Regression . . . . .	26
5.5	Generalized Additive Models . . . . .	27
5.6	K-Means Clustering [US] . . . . .	27
5.7	Hierarchical Clustering . . . . .	28
5.8	PCA . . . . .	29
5.9	KNN . . . . .	30
5.10	SVM . . . . .	31
5.11	Decision Tree . . . . .	33
5.12	Bagging . . . . .	34
5.13	Random Forest . . . . .	35
5.14	Boosting . . . . .	36
5.15	Association Rule Mining . . . . .	37
5.16	Naïve Bayes . . . . .	38
5.17	LDA . . . . .	39
5.18	QDA . . . . .	40
5.19	Model Selection . . . . .	40
<b>6</b>	<b>Communicate, Deploy, Maintain</b>	<b>45</b>
6.1	Communicate . . . . .	45

6.2	Deploy . . . . .	45
6.3	Maintain . . . . .	45
<b>7</b>	<b>Frameworks</b>	<b>47</b>
7.1	PySpark . . . . .	47
7.2	Sparklyr . . . . .	51
7.3	Caret . . . . .	53
7.4	Data Table . . . . .	57
7.5	Keras . . . . .	57
7.6	LME4 . . . . .	59
7.7	Infer Package . . . . .	60
7.8	Networks . . . . .	60
<b>8</b>	<b>Code Snippets</b>	<b>63</b>
8.1	Python . . . . .	63
8.2	R . . . . .	67
8.3	Various . . . . .	68
8.4	Bash . . . . .	77
<b>9</b>	<b>Time Series</b>	<b>79</b>
9.1	Notes . . . . .	79
9.2	Packages . . . . .	79
9.3	Data Conversion . . . . .	79
9.4	Models . . . . .	79
<b>10</b>	<b>Stats</b>	<b>87</b>
<b>11</b>	<b>Deep Learning</b>	<b>97</b>
<b>12</b>	<b>Probabilistic Programming</b>	<b>101</b>
<b>13</b>	<b>NLP</b>	<b>103</b>
<b>14</b>	<b>Hierarchical Models</b>	<b>105</b>

# Chapter 1

## Workflow

### 1.1 Preamble

A typical data science workflow can be framed as following these steps:

- Business: start with a business question, define the goal and measure of success
- Data: find, access and explore the data
- Features: extract, assess and evaluate, select and sort
- Models: find the right model for the problem at hand, compare, optimize, and fine tune
- Communication: Interpret and communicate the results
- Production: transform the code into production ready code, integrate into current ecosystem and deploy
- Maintain: adapt the models and features to the evolution of the environment

More succinctly, as per Hadley Wickham:

import -> tidy -> understand [ transform <-> visualize <-> model ] -> communicate

My combination of the two:

preparation -> import -> tidy -> understand [ transform <-> visualize <-> model ] -> [communicate + deploy + maintain]

Another note is that one way to structure projects is to write user stories a la software engineering.

What lies below is a succinct version of a workflow. Theoretical specifics can be found in other sections of this collection of cribsheets, and libraries or packages to use not included in the Stack can be found on this page.

### 1.2 Annotated Checklist

#### 1.2.1 Preparation

- Understand the business question, goals, and measures of success
- See if the data to answer the question exists, and if it is useful.
- Structure project as user stories

- Initialize report to track and summarise work and to do list.
- Remember to use atomic commit messages

### 1.2.2 Import

- Character Encoding Precedence: utf-8, iso-8859-1, utf-16
- Sampling from data if its too large: subsample package in Python, sqldf library in R
- Sqlite Converter
- Easy Database Management
- Writing sql in pandas

### 1.2.3 Tidy

- Follow tidy data principles. `widyr`, `tidyr`, `dplyr` in R and `pandas` in Python.
- Figure out what you're trying to do with the data.
- See what the data looks like: types of variables, the first and last few observations, missing values or outliers. `xray`, `skimr`, `xda`, and `janitor` in R. `pandas` and `scipy.stats.describe()` in Python.
- Outlier Detection: interquartile range, kernel density estimation, bonferroni test

### 1.2.4 Transform

- `library(DataExplorer)`
- Look at missingness. `missingno` in Python. `VIM`, `hmisc::naclus/naplot()`, `naniar`, `visdat::vis_dat()` in R.
- Imputation. Options include the Mean, Mode, KNN, Random, and Regression. `mice`, `hmisc`, `alice`, and `zoo` in R.
- An R library for categorical dissimilarity
- Easy Regex
- Check for blank spaces and replace with NA. In R: `df[df==""] = NA`.
- Binned Stats
- Anomalies: `xda::num/catSummary(.)` | `xray::anomalies(.)`

### 1.2.5 Visualize

- Quick Exploration Guide: S.O.C.S (Shape, Outlier, Center, Spread)
- Trendlines & Histograms
- Confidence intervals
- Compare the distributions of variables with overlaid density plots
- Scatterplots: Pairwise and color/size the dots by other variables to try to identify any confounding relationships. `GGally::ggpairs()` or `seaborn`.

- Dimensionality reduction (PCA, Kernel PCA, TSNE, Isomap) or hierarchical clustering for multivariate data to get a feel for high dimensional structure.

### 1.2.6 Model

- Stick to regression models for prescriptive analysis. Validate assumptions and tune appropriately. If using Python leverage statsmodels.
- Pre-processing: vtreat in R.
- Use binning + chisquare / mutual information to see correlation between feature and target.
- Pipelines & Feature Unions: sklearn in Python, recipes in R.
- Feature Selection: Variance Threshold, Univariate Feature Selection (`skl_fs.chi2` | `f_regression` | `f_classification` in sklearn), Multivariate Feature Selection (`SelectKBest`, `SelectPercentile`, `feature_selection.RFECV` in sklearn). See also the Rebate and Boruta packages in Python.
- Feature Engineering: Standardization, dimensionality reduction, dummyming. `sklearn.preprocessing.binarize`, `pandas.factorize`, and `featuretools` for automation in Python.
- Model Selection: yellowbrick, Sklearn Evaluation, and `skl-plot` in Python.
- Model Explainability: eli5 or SHAP in Python but concentrate on the latter. SHAP also exists in R's `xgboost` library.
- Tuning: Do bayesian tuning instead of grid or random search. `hyperopt-sklearn`, `sklearn-deap`, and `scikit-optimize`.
- Deal with potential class imbalance: Gradient boosting or `inbalanced learn`.
- Create ensembles: `mlxtend` in Python
- Choose Statistical Test: 1 | 2 | `infer` library
- Ensembling: `mlxtend` in Python, and `caretEnsemble` in R.
- Look at learning curves: 1 | 2
- Use SHAP for model explainability

### 1.2.7 Communicate

Reference this blog.

### 1.2.8 Deployment / Maintenance

Git Workflow

Write helper functions, do testin ( 1 | 2 | 3 | 4 ) and validation (1 | 2 | 3 ). Testing libraries include: `pytest`, `hypothesis`, and `testthat`.

A really useful feature is sharing environments so others can install all the packages used in your code, with the correct versions. You can save the packages to a YAML file with:

```
conda env export > environment.yaml
```

The first part `conda env export` writes out all the packages in the environment, including the Python version. Above you can see the name of the environment and all the dependencies (along with versions) are listed. The second part of the export command, `> environment.yaml` writes the exported text to a YAML file

environment.yaml. This file can now be shared and others will be able to create the same environment you used for the project. To create an environment from an environment file use:

```
conda env create -f environment.yaml
```

This will create a new environment with the same name listed in environment.yaml.

<https://github.com/Quartz/bad-data-guide#margin-of-error-is-too-large>

[https://github.com/ianozsvald/data\\_\\_science\\_\\_delivered](https://github.com/ianozsvald/data__science__delivered)

<https://github.com/jtleek/datasharing>



## Chapter 2

# Import & Tidy

### 2.1 Import

- In tidy data:
  - Each variable forms a column.
  - Each observation forms a row.
  - Each type of observational unit forms a table.
- A schema is a blueprint for storing data. For a table every row has same number of columns, and each column is of the same type.
- `conda create -n yourenvname python=x.x anaconda`
- json normalize
- Temporary pandas Columns
- Sampling a massive CSV file:
  - `pip install subsample`
  - `gzcat ginormous.csv.gz | subsample -n 100000 > sampled.csv.gz`
- Codebook
- Data file naming convention: `{iso-date}_{concise_description}.{file_extension}`
- Variable naming conventions: Consistency and balance are important

#### 2.1.1 SQL

- CSV to DB V1:
  - `$ sqlite3 db_name.db`
  - `sqlite> .mode csv db_name`
  - `sqlite> .import data.csv db_name`
- CSV to DB V2
- SQL UNION stacks one dataset on top of the other. It only appends distinct values. More specifically, when you use UNION, the dataset is appended, and any rows in the appended table that are exactly identical to rows in the first table are dropped. If you'd like to append all the values from the second table, use UNION ALL.

- List tables: `SELECT name FROM sqlite_master WHERE type='table'; [sqlite]`
- Show columns: `.headers ON [sqlite]`
- `ROW_NUMBER()`, `RANK()`, `DENSE_RANK()`, `NOW()`, `INTERVAL`, `x IS (NOT) NULL`
- `COUNT(DISTINCT month)`: returns count of non-null values like table in R
- The CASE statement is followed by at least one pair of WHEN and THEN statements: `CASE WHEN year = 'SR' THEN 'yes' ELSE NULL END`. You can also define a number of outcomes in a CASE statement by including as many WHEN/THEN statements as you'd like
- If you include two (or more) columns in a `SELECT DISTINCT` clause, your results will contain all of the unique pairs of those two columns

### 2.1.2 Scraping

- Access robots.txt: `website_url.domain/robots.txt`
- Tips: 1) Use bot net, 2) user agent switching
- Try polling in your wait about every 500 milliseconds. Also use presence of element located. I generally have better luck with that expected condition than visibility. Also, submit the search using the search button.
- Selenium: `save_screenshot`, `forward()`, `back()`, `maximize()`, use `action_chains` to do stuff like scroll `find_element_by_link_name`, `element_to_be_clickable`, `visibility_of_element_located`

```
from pyvirtualdisplay import Display
from selenium import webdriver
display = Display(visible=0, size=(800, 600))
display.start()
browser = webdriver.Firefox()
browser.get('http://www.google.com')
print(browser.title)
```

## 2.2 Tidy

- Tidy Data: R | Py1 | Py2
- `import pandas_profiling; pandas_profiling.ProfileReport(data)`
- Optimal Bins For Histogram
- **The Coefficient of Unalikeability:** (Unalikeability is defined as how often observations differ from one another). It varies from 0 to 1. The higher the value, the more unlike the data are.

# Chapter 3

## Transform & Visualize

### 3.1 Transform

#### 3.1.1 General

- There is no one rule about when a number is not accurate enough to use, but as a rule of thumb, you should be cautious about using any number with a MOE (Margin-of-error) over 10%.

A good formula is: if your sample is picked uniformly at random and is of size at least:

$$\frac{-\log(\frac{\alpha}{2})}{2e^2}$$

then with probability at least  $1-d$  your sample measured proportion  $q$  is no further away than  $e$  from the unknown true population proportion  $p$  (what you are trying to estimate). Need to estimate with 99% certainty the unknown true population rate to  $\pm 10\%$  precision? That is  $d=0.01$ ,  $e=0.1$  and a sample of size 265 should be enough. A stricter precision of  $\pm 0.05$  causes the estimated sample size to increase to 1060, but increasing the certainty requirement to 99.5% only increases the estimated sample size to 300.

- use spatial sign to transform data with outliers.
- Too many levels for a category?
  - limit number of categories through feature selection
  - <https://stats.stackexchange.com/questions/95212/improve-classification-with-many-categorical-variables>
  - bucket groups by number of samples
- <https://cran.r-project.org/web/packages/vtreat/index.html>
- Long story short, if these outliers are really such (i.e. they appear with a very low frequency and very likely are bad/random/corrupted measurements) and they do not correspond to potential events/failures that your model should be aware of, you can safely remove them. In all other cases you should evaluate case by case what those outliers represent.
- Otherwise, assuming levels of the categorical variable are ordered, the polyserial correlation (here it is in R), which is a variant of the better known polychoric correlation. Note the latter is defined based on the correlation between the numerical variable and a continuous latent trait underlying the categorical variable.
- Use runif to do sampling
- You should use a logarithmic scale when percent change, or change in orders of magnitude, is more important than changes in absolute units. You should also use a log scale to better visualize data that is heavily skewed. Taking the logarithm only works if the data is non-negative. There are other

transforms, such as  $\text{arcsinh}$  or signed log, that you can use to decrease data range if you have zero or negative values.

- Normalizing by mean and standard deviation is most meaningful when the data distribution is roughly symmetric.
- Monetary amounts are often log-<https://github.com/ColinFay/erum2018> normally distributed—the log of the data is normally distributed. This leads us to the idea that taking the log of the data can restore symmetry to it.

### 3.1.2 Imputation

### 3.1.3 Missingness & Imputation

There are three types of missingness: 1) Completely at Random, 2) At Random, 3) Not at random

- The pros of imputation: 1) Helps retain a larger sample size of your data, 2) Does not sacrifice all the available information in an observation because of sparse missingness, 3) Can potentially avoid unwanted bias.
- The cons of imputation: 1) The standard errors of any estimates made during analyses following imputation can tend to be too small, 2) The methods are under the assumption that all measurements are actually “known,” when in fact some were imputed, 3) Can potentially induce unwanted bias.
- Other: 1) `dplyr::case_when`, 2) mean of each column = `df.mean(axis = 0) || df.mean(axis = 'index' )`, 3) mean of each row = `df.mean(axis = 1) || df.mean(axis = 'columns' )`

Some solutions:

- Mean: Simple but can distort distribution, underestimate standard deviation, and distort variable relationships by dragging correlation to zero.
- Random: Can amplify outlier observation and induce bias.
- Regression: Use observed variables and relationships between these variables. Must make assumptions and can badly extrapolate.

multiple stochastic regression imputation: `mice` R, `fancy impute` Python

## 3.2 Visualize

- logging converts multiplicative relationships to additive relationships, and by the same token it converts exponential (compound growth) trends to linear trends. By taking logarithms of variables which are multiplicatively related and/or growing exponentially over time, we can often explain their behavior with linear models.
- `swarmplot`
- <https://www.jumpingrivers.com/blog/styling-ggplot2-r-graphics/>
- Rule of thumb for bins in histogram: `ggplot() + geom_histogram(aes(x), binwidth = diff(range(x)) / (2 * IQR(x) / length(x)^(1/3)))`.

# Chapter 4

## Model

### 4.1 General

- Model Template: intro/packages, fs/fe, tuning, pros/cons, notes
- Don't use brier loss for ordinal predictions
- brier score: .25 if say .5 and .33 if randomly assign probs
- R uses class encoded as 1 in classification to make predictions. Use levels() function on factor to determine what 1 is.
- 20 observations per feature is pretty good for making predictions.
- $y = b_0 + b_1x_1 + b_2x_2 + b_3x_1x_2$ . ( $b_1 + b_2x_2$ ) is the increase in y with a unit increase in  $x_1$ .
- <http://ml-cheatsheet.readthedocs.io/en/latest/index.html>
- Algorithms for Rec Systems: Linear/Logistic/Elastic Net Regression, Restricted Boltzmann Machines, Singular Value Decomposition, Markov Chains, Latent Dirichlet Allocation, Association Rules, Gradient Boosted Decision Trees, Random Forests, Affinity Propagation, K-Means, Matrix Factorization, Alternating Least Squares.
- For example, multilevel models themselves may be referred to as hierarchical linear models, random effects models, multilevel models, random intercept models, random slope models, or pooling models.
- <https://www.listendata.com/2018/03/regression-analysis.html>
- In general, raising the classification threshold reduces false positives, thus raising precision.
- Multi-Modal Classification: In the machine learning community, the term Multi-Modal is used to refer to multiple kinds of data. For example, consider a YouTube video. It can be thought to contain 3 different modalities: 1) The video frames (visual modality), 2) The audio clip of what's being spoken (audio modality), 3) Some videos also come with the transcription of the words spoken in the form of subtitles (textual modality)
- Less samples and more features increase the chance of overfitting.
- You can choose either of the following inference strategies: offline inference, meaning that you make all possible predictions in a batch, using a MapReduce or something similar. You then write the predictions to an SSTable or Bigtable, and then feed these to a cache/lookup table. online inference, meaning that you predict on demand, using a server.

- A static model is trained offline. That is, we train the model exactly once and then use that trained model for a while. A dynamic model is trained online. That is, data is continually entering the system and we're incorporating that data into the model through continuous updates.
- How would multiplying all of the predictions from a given model by 2.0 (for example, if the model predicts 0.4, we multiply by 2.0 to get a prediction of 0.8) change the model's performance as measured by AUC? No change. AUC only cares about relative prediction scores. Yes, AUC is based on the relative predictions, so any transformation of the predictions that preserves the relative ranking has no effect on AUC. This is clearly not the case for other metrics such as squared error, log loss, or prediction bias (discussed later).
- AUC is desirable for the following two reasons: AUC is scale-invariant. It measures how well predictions are ranked, rather than their absolute values. UC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen. However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases: Scale invariance is not always desirable. For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that. Classification-threshold invariance is not always desirable. In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.
- Raising our classification threshold will cause the number of true positives to decrease or stay the same and will cause the number of false negatives to increase or stay the same. Thus, recall will either stay constant or decrease.
- <https://www.dataquest.io/blog/learning-curves-machine-learning/>
- A feature cross is a synthetic feature formed by multiplying (crossing) two or more features. Crossing combinations of features can provide predictive abilities beyond what those features can provide individually.
- Bias: Error Introduced by approximating real-life problem by using a simple method
- Imagine a linear model with two strongly correlated features; that is, these two features are nearly identical copies of one another but one feature contains a small amount of random noise. If we train this model with L2 regularization, what will happen to the weights for these two features? L2 regularization will force the features towards roughly equivalent weights that are approximately half of what they would have been had only one of the two features been in the model.
- L2 regularization will encourage many of the non-informative weights to be nearly (but not exactly) 0.0.
- F1 Score =  $TP / (TP + FP + FN)$ . Essentially how good you are at identifying the positive class.
- Active Learning: Finding the optimal data to manually label
- Simulated Annealing: Gradient descent extension
- A variable that is completely useless by itself can provide a significant performance improvement when taken with others.
- So, if your problem involves kind of searching a needle in the haystack when for ex: the positive class samples are very rare compared to the negative classes, use a precision recall curve. Otherwise use a ROC curve because a ROC curve remains the same regardless of the baseline prior probability of your positive class (the important rare class).
- Variance: The amount the model output will change if the model was trained using a different data
- Flexible Methods: higher variance, low bias, Example: KNN ( $k = 1$ )
- Inflexible Methods: low variance, high bias, Example: Linear regression

- Sequence mining Algorithms: spade, gsp, freespan; hmmlearn
- Network models: graphical lasso, ising model, granger causality test, network hypothesis testing, bayesian networks
- Parametric: Model has a function shape and form, Model is trained/fit using training data to determine parameter values, reduces the problem of training a model down to training a set of parameter values, Examples: linear regression
- Non-Parametric: No assumption about model form is made, Example: KNN
- Set Random state by hand in sklearn.
- Generally need 3 rows of data per variable (to prevent model from seeing signal where there is only noise) [Nina Zumel]
- Domain knowledge for feature selection and random forest feature importance (5-10) best variables. [Nina Zumel]
- Because we have a couple thousand data points, even though salary can best be represented by a Poisson distribution, I'm going to use Gaussian distribution. (With bigger datasets, these two distribution start to become very similar).
- We can also tell that our input, smart\_1\_normalized, doesn't have a very strong relationship to our output because the standard error (0.009) is more than 1/10 of our estimate (0.02).
- The less complex an ML model, the more likely that a good empirical result is not just due to the peculiarities of the sample.
- The ML Fine Print: Three basic assumptions: 1) We draw examples independently and identically (i.i.d.) at random from the distribution, 2) The distribution is stationary: It doesn't change over time, 3) We always pull from the same distribution: Including training, validation, and test sets. In practice, we sometimes violate these assumptions. For example: 1) Consider a model that chooses ads to display. The i.i.d. assumption would be violated if the model bases its choice of ads, in part, on what ads the user has previously seen. 2) Consider a data set that contains retail sales information for a year. User's purchases change seasonally, which would violate stationarity. When we know that any of the preceding three basic assumptions are violated, we must pay careful attention to metrics.
- empirical risk minimization: In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss.
- Loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater.

## 4.2 Pre-processing

## 4.3 Feature Engineering

- Dealing with cyclical features: Hours of the day, days of the week, months in a year, and wind direction are all examples of features that are cyclical. Many new machine learning engineers don't think to convert these features into a representation that can preserve information such as hour 23 and hour 0 being close to each other and not far. Keeping with the hour example, the best way to handle this is to calculate the sin and cos component so that you represent your cyclical feature as (x,y) coordinates of a circle. In this representation hour, 23 and hour 0 are right next to each other numerically, just as they should be.

```
import numpy as np
df['hr_sin'] = np.sin(df.hr*(2.*np.pi/24))
df['hr_cos'] = np.cos(df.hr*(2.*np.pi/24))
df['mnth_sin'] = np.sin((df.mnth-1)*(2.*np.pi/12))
df['mnth_cos'] = np.cos((df.mnth-1)*(2.*np.pi/12))
```

- robust scaler python: median & quantiles

## 4.4 Feature Selection

- Variance Threshold: Use this to exclude features that don't meet a variance threshold.
- Univariate Feature Selection: For classification can use chi-squared and F-Test while F-Test for regression. Functions: `chi2`, `f_regression`, `f_classification` from `sklearn.feature_selection`.
- SelectKBest/SelectPercentile: Keep the k highest scoring features and keep a user-specified highest scoring percentage of features. Can use the univariate feature selection in these functions.
- `sklearn.feature_selection.RFE(CV)`
- To work around magic values, convert the feature into two features: One feature holds only quality ratings, never magic values. One feature holds a boolean value indicating whether or not a quality\_rating was supplied. Give this boolean feature a name like `is_quality_rating_defined`.
- Make sure that your test set meets the following two conditions: 1) Is large enough to yield statistically meaningful results, 2) Is representative of the data set as a whole. In other words, don't pick a test set with different characteristics than the training set.
- Handling extreme outliers: Given data with a very long tail, log scaling does a slightly better job, but there's still a significant tail of outlier values. Let's pick yet another approach. What if we simply "cap" or "clip" the maximum value at an arbitrary value, say 4.0? Clipping the feature value at 4.0 doesn't mean that we ignore all values greater than 4.0. Rather, it means that all values that were greater than 4.0 now become 4.0. This explains the funny hill at 4.0. Despite that hill, the scaled feature set is now more useful than the original data. Binning.
- Know your data. Follow these rules: 1) Keep in mind what you think your data should look like. 2) Verify that the data meets these expectations (or that you can explain why it doesn't). 3) Double-check that the training data agrees with other sources (for example, dashboards).

## 4.5 Other

### 4.5.1 Unbalanced Classes

Research on imbalanced classes often considers imbalanced to mean a minority class of 10% to 20%.

That said, here is a rough outline of useful approaches. These are listed approximately in order of effort:

Do nothing. Sometimes you get lucky and nothing needs to be done. You can train on the so-called natural (or stratified) distribution and sometimes it works without need for modification.

Balance the training set in some way: Oversample the minority class. Undersample the majority class. Synthesize new minority classes.

Throw away minority examples and switch to an anomaly detection framework.



At the algorithm level, or after it: Adjust the class weight (misclassification costs). Adjust the decision threshold. Modify an existing algorithm to be more sensitive to rare classes.

Construct an entirely new algorithm to perform well on imbalanced data.

Use AUC, F1 Score, Cohen's Kappa, ROC curve, a precision-recall curve, a lift curve, or a profit (gain) curve for evaluation.

Use probability estimates and not the default .5 threshold.

Undersampling, oversampling, increase weight of minority class

Decision trees often perform well on imbalanced datasets because their hierarchical structure allows them to learn signals from both classes.

Reframe as Anomaly Detection

- Sklearn Eg: `wclf = svm.SVC(kernel='linear', class_weight={1: 10})` & `svm.SVC(kernel='linear', class_weight='balanced')`



# Chapter 5

## Model

### 5.1 Linear Regression

#### 5.1.1 Intro

- The standard least squares coefficient estimates are scale equivariant: if we multiply a predictor variable by a constant, the corresponding least squares coefficient estimate will be scaled down by the same constant.
- Used when you're predicting a continuous value.
- In regression, it is often recommended to center the variables so that the predictors have mean 0. This makes it so the intercept term is interpreted as the expected value of  $Y_i$  when the predictor values are set to their means. Otherwise, the intercept is interpreted as the expected value of  $Y_i$  when the predictors are set to 0, which may not be a realistic or interpretable situation.
- When  $p > n$ , we can no longer calculate a unique least square coefficient estimate, the variances become infinite, so OLS cannot be used at all.
- In presence of few variables with medium / large sized effect, use lasso regression. In presence of many variables with small / medium sized effect, use ridge regression.
- The fitted regression line using a sample of data gives imperfect predictions for future observations due to sampling variability and randomness in  $Y$  that is not related to  $X$ .
- Linear Correlation  $\Rightarrow$  Causation if covariates respect: linearity normality indep homoscedasticity + all confounders in model
- In summary, "pooling" your data and fitting one combined regression function allows you to easily and efficiently answer research questions concerning the binary predictor variable.
- If two independent variables are measured in exactly the same units, we can assess their relative importance in their effect on  $y$  quite simply: the larger the coefficient, the stronger the effect.
- Okay, so many of the features are strongly correlated. We can make use of this in our model by including polynomial features (e.g. `PolynomialFeatures(degree=2)` from `scikit-learn`). Adding these brings our validation loss down to 0.69256 (-0.05% from baseline).
- We prefer natural logs (that is, logarithms base  $e$ ) because, as described above, coefficients on the natural-log scale are directly interpretable as approximate proportional differences: with a coefficient of 0.06, a difference of 1 in  $x$  corresponds to an approximate 6% difference in  $y$ , and so forth.

```
import statsmodels.api as sm
X = sm.add_constant(X)
rgr_lr = sm.OLS(Y, X).fit(displ=False)
print(results.summary())
```

### 5.1.2 Assumptions

The regression assumption that is generally least important is that the errors are normally distributed. In fact, for the purpose of estimating the regression line (as compared to predicting individual data points), the assumption of normality is barely important at all. Thus, in contrast to many regression textbooks, we do not recommend diagnostics of the normality of regression residuals. [Gelman]

Linear regression errors are not autocorrelated. The Durbin-Watson statistic detects this; if it is close to 2, there is no autocorrelation.

A linear regression model is only as good as the validity of its assumptions, which can be summarized as:  $y = b_0 + b_1x + \epsilon$ ,  $\epsilon$  is i.i.d from  $N(0, \sigma)$ . More verbosely:

- **Linearity:** This is a linear relationship between the predictor and the response variables. If this relationship is not clearly present, transformations (log, polynomial, exponent and so on) of the X or Y may solve the problem. Check with a scatterplot.
- **Normality:** The error terms are drawn from an identical Gaussian distribution, i.e a normal distribution of the dependent variable for each value of the independent variable. Inspect the quantile-quantile plot of the residuals. Also Shapiro-Wilk Test for Normality.
- **Non-correlation or independence of errors:** A common problem in the time series and panel data where  $\epsilon_t = \beta_1 \epsilon_{t-1}$ ; if the errors are correlated, you run the risk of creating a poorly specified model. The residual value for an arbitrary observation is not predictable from knowledge of another observation's residual value; they are uncorrelated. Inspect the residual plot after fitting the regression. Ideally, there would be no clear pattern in the fluctuation of the error terms.
- **Homoscedasticity (constant variance):** Normally the distributed and constant variance of errors, which means that the variance of the errors is constant across the different values of inputs. Violations of this assumption can create biased coefficient estimates, leading to statistical tests for significance that can be either too high or too low. This, in turn, leads to the wrong conclusion. This violation is referred to as heteroscedasticity. The error terms have the same variance. Check the residual plot which is a scatterplot of the residuals vs the fitted values.
- **No multi-collinearity:** No linear relationship between two predictor variables, which is to say that there should be no correlation between the features. This, again, can lead to biased estimates. If not coefficient estimates could be unstable, introduce redundancies within predictors, and make it more difficult to make inferences. Could inflate standard errors, decrease power and reliability of regression coefficients, and create need for a larger sample size. Check the variance inflation factors. Tells the factor by which the estimated variance of a variable is larger in comparison to it it were completely uncorrelated with other variables in the model. If the VIF is more than five then the variable is a candidate to remove from the model. Its information is contained in the other variables. Variance Inflation Factor: Measures how the model factors influence the uncertainty of coefficient estimates.  $\geq 5$  is bad.
- **Minimal outliers:** Outliers can severely skew the estimation and, ideally, must be removed prior to fitting a model using linear regression; this again can lead to a biased estimate.

A practical summary using R.

```
# general check
plot(rgr_lr_st1)
```

```

# linearity check
avPlots(rgr_lr_st1)

# repeat of outlier analysis
#influencePlot(rgr_lr_st1)

# multi-collinearity
vif(model) %>% data.frame() %>% mutate(VIF = `GVIF..1..2.Df..`, VIF = VIF^2)

# Test for Autocorrelated Errors
durbinWatsonTest(rgr_lr_st1)

set.seed(0)

data = rnorm(100) # Generating data from a standard normal distribution.
shapiro.test(data) # P-value insignificant; retain H0, conclude data is normally distributed.

# Inspect this same data using the QQ-plots (For normal data, the QQ-plot produces a straight-line rela
qqnorm(normal.data) + qqline(normal.data)
#Breusch-Pagan (BP) test lmtest package: is used to test for heteroskedasticity in a linear regression

```

Outliers are observations that have high residual values. Leverage points are observations that have unusually small or large independent variable values. Cook's distance helps to measure the effect of deleting an observation from the #dataset and rerunning the regression. Observations that have large residual values and also high leverage tend to pose threats to the accuracy of the regression line and thus need to be further investigated. Look at influence plot. Confidence and prediction intervals. For avplot distinct patterns are indications of good contributions. Influence plot to look at hat values (lower is better). Use F-test to compare models. F-test & partial f-test for simple and multiple linear regression.

Shapiro-Wilk Test for Normality:  $H_0$  says the data is normally distributed while  $H_a$  says it is not. To run this test in R, the syntax is quite simple:

And a guide for python.

```
statsmodels.stats.outliers_influence.variance_inflation_factor
```

Fixing assumptions:

Transformations: Can remedy assumption violations and strengthen linear relationships between variables, but can lead to overfitting and less interpretability. square root correction: Take square root of x variable(s). log transformation: Take log of y. General rule of thumb: powers > 1 for skewed left data, powers < 1 for data skewed right.

Box-Cox: Iterates along values of lambda by maximum likelihood so as to maximize the fit of the transformed variable to a normal distribution. Does not guarantee normality since it minimizes standard deviation. Can only be used on positive values. Use on negative values by shifting by a constant value. Use Box-Cox transformations when predictions are skewed.

### 5.1.3 Selection

- Forward and Backward Stepwise Selection.
- For simple linear regression, the principal hypothesis test is as follows:  $H_0 : \beta_1 = 0$ ,  $H_A : \beta_1 \neq 0$ . If  $H_0$  is true then the population mean of Y would be  $\beta_0$  no matter what the value of X. (X has no effect on Y.)

- F-Test:  $H_0$  says that all/some coefficients are zero versus  $H_a$  which says that there is at least one non-zero coefficient. If  $H_0$  fail to reject then F value close to 1, else F value greater than 1.
- A.I.C and B.I.C: Smaller values are better. Can be used to compare multiple models at the same time. They reward goodness of fit and penalizes complexity. Favor A.I.C for prediction and B.I.C for descriptive power (penalty term more stringent and favors a parsimonious model)
- R-squared increases whenever a predictor is added, and it doesn't take model complexity into consideration which makes it prone to overfitting. Use adjusted R-squared instead. Additional predictors only make this increase if they are statistically significant. It's easy to show that given no other data for a set of numbers, the predictor that minimises the MAE is the median, while the predictor that minimises the MSE is the mean.
- Compare RSME to sd of data for sense of how good it is. (If RSME < sd then good.)

#### 5.1.4 Pros/Cons

##### Pros

- Very fast (runs in constant time)
- Easy to understand the model
- Less prone to overfitting

##### Cons

- Unable to model complex relationships
- Unable to capture nonlinear relationships without first transforming the inputs

#### 5.1.5 Other

- <https://twitter.com/alexpghayes/status/1033824995905077250>
- Quantile regression is an alternative for quantiles.
- Type 1 Error: Accept  $H_a$  when  $H_0$  is true.
- RSS is The sum of the squared error terms for each observation in our dataset. RSE is The standard deviation of the residuals about the regression surface and is an estimate of  $\sigma$ . TSS is a measure of the total squared deviation of our response variable from its mean value.  $R^2 = 1 - \text{RSS}/\text{TSS}$ .
- Increasing  $x_1$  by 1 means you are increasing the underlying  $X_1$  by 1 standard deviation  $x_1 + 1 = (X_1 + \sigma)/\sigma$ . Therefore you can say that  $b_1$  is the effect on  $y$  of increasing  $X_1$  by 1 sigma.
- If there is an ascending or descending order to your dependent variable, ordinal regression is more appropriate as it has more power than multinomial logistic regression.
- leaps lm package in R is a greedy way to find the best predictors by cycling through all the combinations.
- Linear Regression by Year

```
library(purrr)
pga %>%
  select(name, year, driveavg, drivepct, score.avg) %>%
  split(.$year) %>%
  map(~lm(score.avg ~ driveavg + drivepct, data=..)) %>%
  map(summary)
```

## 5.2 Ridge/Lasso

### 5.2.1 Intro

Minimizes RSS but also has shrinkage penalty (L2 penalty). A small lambda penalizes the RSS more than the shrinkage penalty. A large lambda penalizes the shrinkage penalty more than the RSS. By shrinking the coefficient estimates towards 0 by increasing lambda, the bias increases slightly but remains relatively small, the variance reduces substantially, and the mean squared error of the predictions drops. Not scale invariant due to the shrinkage penalty. To avoid the issue of overvaluing or undervaluing certain predictor variables simply based on their magnitudes, we must standardize the variables prior to performing ridge regression. The main disadvantage of ridge regression is that, while parameter estimates are shrunk, they only asymptotically approach 0 as we increase the value of lambda.

Standardize for ridge/lasso alpha makes them not scale invariant

Minimizes RSS but also has L1 penalty (norm). This necessarily forces some coefficient estimates to be exactly 0 (when lambda is sufficiently large). It has the added advantage of essentially performing variable selection, yielding models that are both accurate and parsimonious. Restricting ourselves to simpler models by including a Lasso penalty will generally decrease the variance of the fits at the cost of higher bias.

Note: In both ridge and lasso regression, is important to select an appropriate value of lambda by means of cross-validation.

### 5.2.2 Assumptions

### 5.2.3 Evaluation

### 5.2.4 Pros/Cons

## 5.3 Logistic Regression

### 5.3.1 Intro

logistic is the same as probit

wilkinson rogers logistic regression

weighted logistic regression

The right-hand predictor side of the equation must be linear with the left-hand outcome side of the equation. You must test for linearity in the logit (in logistic regression the logit is the outcome side). This is commonly done with the Box-Tidwell Transformation (Test): Add to the logistic model interaction terms which are the crossproduct of each independent times its natural logarithm  $[(X)\ln(X)]$ . If these terms are significant, then there is nonlinearity in the logit. This method is not sensitive to small nonlinearities.

Logistic regression assumes that  $\text{logit}(y)$  is linear in the values of  $x$ . Like linear regression, logistic regression will find the best coefficients to predict  $y$ , including finding advantageous combinations and cancellations when the inputs are correlated. In other words, you can think of logistic regression as a linear regression that finds the log-odds of the probability that you're interested in. Logistic Regression can also be used with kernel methods.

Logistic: When you're predicting which category your observation is in. (Is this is a cat or a dog?)

The link function is the logit function. Logit is the inverse of the sigmoid and gives the log odds, i.e  $p/(1-p)$ .

Easy to incorporate prior knowledge, number of features are small, training is fast, precision not critical.

No closed form expression to maximize coefficients with maximum likelihood estimation, so one uses gradient descent or stochastic gradient descent. The second one is faster.

For logistic regression value of  $c$  penalizes features. Low  $c$  penalizes a lot and vice versa. A large  $c$  decreases the effect of the regularization term (the  $l_2$  penalization).

Makes a linear boundary between classes.

For multiclass classification it will build multiple models. Given 3 classes 0,1, and 2, there will be models 0 and not 0, 1 and not 1, and 2 and not 2.

Standardization isn't required for logistic regression. The main goal of standardizing features is to help convergence of the technique used for optimization. For example, if you use Newton-Raphson to maximize the likelihood, standardizing the features makes the convergence faster. Otherwise, you can run your logistic regression without any standardization treatment on the features.

### 5.3.2 Assumptions

- The linear regression analysis requires all variables to be multivariate normal. This assumption can best be checked with a histogram or a Q-Q-Plot.
- Requires the dependent variable to be binary and ordinal logistic regression requires the dependent variable to be ordinal.
- The observations to be independent of each other. In other words, the observations should not come from repeated measurements or matched data.
- There to be little or no multicollinearity among the independent variables.
- Assumes linearity of independent variables and logit of the probabilities ( $\text{logit}(p) = \log(p/(1-p))$ ). Check in R.
- There is no influential values (extreme values or outliers) in the continuous predictors

Notice the points fall along a line in the middle of the graph, but curve off in the extremities. Normal Q-Q plots that exhibit this behavior usually mean your data have more extreme values than would be expected if they truly came from a Normal distribution.

The Durbin-Watson test is used to test the null hypothesis that linear regression residuals are uncorrelated, against the alternative that autocorrelation exists

<https://data.library.virginia.edu/diagnostic-plots/>

#### No Multicollinearity

`multicol` shows which columns could be removed for multi-collinearity. (The `vif` is not working because of multicollinearity.)

```
# vif(clf_lr$finalModel)
# multicol <- alias(clf_lr$finalModel)$Complete
# multicol
```

#### Linear With The Logit Of Predictions

```
# box tidwell doesn't work since features have negative values.

# min_max_scaler <- function(x) (x - min(x))/(max(x) - min(x))
#
# df_num_selected <- df2 %>%
#   select(
#     OverallMap,
```



```

#   StrandMap,
#   lesson_length,
#   g_avg_overall_map,
#   g_avg_strand_map,
#   g_avg_score,
#   g_num_students,
#   cohort_avg_overall_map,
#   cohort_avg_strand_map
# ) %>%
#   map(min_max_scaler) %>%
#   data.frame() %>%
#
# formula_all_num_features <- as.formula(glue('pass ~ {paste(df_num_selected %>% names, collapse = " +
# car::boxTidwell(formula_all_num_features, data = df_num_selected )
#
# Visual test too many features to work well
#
# lin_test <- df2 %>%
#   dplyr::select_if(is.numeric)
#
# predictors <- colnames(lin_test)
#
# lin_test <- lin_test %>%
#   mutate(logit = log(clf_lr_final$fitted.values/(1-clf_lr_final$fitted.values))) %>%
#   gather(key = "predictors", value = "predictor.value", -logit)
#
# ggplot(lin_test, aes(logit, predictor.value))+
#   geom_point(size = 0.5, alpha = 0.5) +
#   geom_smooth(method = "loess") +
#   theme_bw() +
#   facet_wrap(~predictors, scales = "free_y")

```

### 5.3.3 Evaluation

- Wald Test:  $h_0: b = 0$  and  $h_a: b \neq 0$ .  $h_0$  means the log odds are unaffected by  $x$  and so  $x$  has no bearing on the prediction of success.
- Deviance  $G^2$  and Goodness of Fit: The deviance associated with a given logistic regression model  $M$  is based on comparing the maximum log-likelihood of model  $M$  against the saturated model  $S$ . The smaller the deviance the better. For goodness of fit  $h_0$  says the model is appropriate while  $H_a$  says the opposite.
- McFadden's Pseudo  $R^2$ .
- A.I.C / B.I.C
- Absolute scale odds instead of relative

### 5.3.4 Pros/Cons

#### Pros

- Highly interpretably
- Model training and prediction are fast

- No tuning outside of regularization required
- Features don't need scaling
- Can perform with a small number of observations
- Outputs well-calibrated predicted probabilities
- low variance

#### Cons

- Presumes a linear relationship between the features and the log-odds of the response.
- Performance is generally not competitive with the best supervised learning methods
- Can't automatically learn feature interactions
- Breaks down when classes are perfectly separable.
- High bias

### 5.3.5 Code Example

```
import statsmodels.api as sm
X = sm.add_constant(X)
clf_lr = sm.Logit(Y, X).fit(dispatch=False)
print(results.summary())
```

## 5.4 Poisson Regression

### 5.4.1 Intro

Poisson: When you're predicting a count value. (How many dogs will I see in the park?)

Poisson Regression: The Poisson distribution is used to model variation in count data (that is, data that can equal 0,1,2,...).

Remember that you must specify `family = poisson` or `family = quasipoisson` when using `glm()` to fit a count model.

### 5.4.2 Assumptions

One of the assumptions of Poisson regression to predict counts is that the event you are counting is Poisson distributed: the average count per unit time is the same as the variance of the count. In practice, "the same" means that the mean and the variance should be of a similar order of magnitude.

When the variance is much larger than the mean, the Poisson assumption doesn't apply, and one solution is to use quasipoisson regression, which does not assume that  $\text{variance} = \text{mean}$ .

### 5.4.3 Evaluation

### 5.4.4 Pros/Cons

#### Pros

Cons

## 5.5 Generalized Additive Models

### 5.5.1 Intro

`mgcv::gam`: More complex than `lm`, more likely to overfit, best used on larger data sets. use `s(var)` to denote non-linear relationship. Don't use `s()` with categorical variable. Use `type = terms` for y values of plots to get predictions `type = response`.

Also remember that `gam()` from the package `mgcv` has the calling interface

`gam(formula, family, data)` For standard regression, use `family = gaussian` (the default).

For GAM models, the `predict()` method returns a matrix, so use `as.numeric()` to convert the matrix to a vector.

### 5.5.2 Assumptions

### 5.5.3 Evaluation

### 5.5.4 Pros/Cons

Pros

Cons

## 5.6 K-Means Clustering [US]

### 5.6.1 Intro

K-Means clustering method considers two assumptions regarding the clusters – first that the clusters are spherical and second that the clusters are of similar size.

k-means assumes the variance of the distribution of each attribute (variable) is spherical; all variables have the same variance;

the prior probability for all k clusters is the same, i.e., each cluster has roughly equal number of observations;

Must choose k for clusters. Initialize cluster centres randomly, assign each point to its closest cluster by a distance metric. Recalculate centroids. Halt when cluster assignments no longer change. Clusters will be distinct and non-overlapping. Goal to minimize within-cluster variation. The within-cluster variation for the kth cluster is the sum of all of the pairwise squared Euclidean distances between the observations in the kth cluster divided by the total number of observations in the kth cluster. Limitation is that it depends on initialization of centroids. Guaranteed convergence but only to a local minima. Run the algorithm several times and pick the solution that yields the smallest within-cluster variance. Increasing k will decrease variance and increase bias and vice versa. K-means: Jaccard coefficient.

Note: Not enough to use within-cluster variance as this decreases as one increases k. Use elbow method or scree plot to choose optimal k where the variance no longer decreases dramatically. To choose how many groups to use, I ran a k-means analysis for each possible number of groups from 5 to 35 and found that 20 was right about the spot that the amount of variance stopped decreasing consistently. This is called the

elbow test and while the results weren't totally definitive, they fit in well with the general rule of thumb for determining # of groups which is the square root of the # of observations/2.

One way to assess whether a cluster represents true structure is to see if the cluster holds up under plausible variations in the dataset. The `fpc` package has a function called `clusterboot()` that uses bootstrap resampling to evaluate how stable a given cluster is.[3] `clusterboot()` is an integrated function that both performs the clustering and evaluates the final produced clusters. It has interfaces to a number of R clustering algorithms, including both `hclust` and `kmeans`.

In K-means, we assume that each cluster fits a Gaussian distribution (normal distribution)

We can set K to optimally cluster the data by starting with a small number of clusters, and then iteratively splitting them until all clusters fit a normal distribution.

### 5.6.2 Assumptions

### 5.6.3 Characteristics

### 5.6.4 Evaluation

Davies–Bouldin index: This is an internal evaluation scheme, where the validation of how well the clustering has been done is made using quantities and features inherent to the dataset. This has a drawback that a good value reported by this method does not imply the best information retrieval.

### 5.6.5 Pros/Cons

#### Pros

Uses simple principle which can be explained in non-statistical terms. It is efficient.

#### Cons

Less sophisticated, random choices of centers, and requires guess for centers.

## 5.7 Hierarchical Clustering

### 5.7.1 Intro

Generates dendrograms. The lower down a fusion occurs the more similar the group of observations and vice versa. Begin with  $n$  observations and a distance measure of all pairwise dissimilarities. Evaluate pairwise intercluster dissimilarities among the clusters and fuse pair of clusters that are least dissimilar. Repeat process for remaining clusters. Continue for  $i=n$  to  $i=2$ . Need to pick a dissimilarity measure and a linkage method. Complete linkage: maximal inter-cluster dissimilarity, single linkage: minimal inter-cluster dissimilarity, average linkage: mean inter-cluster dissimilarity. Complete is sensitive to outliers but tends to produce compact clusters. Distance between groups: complete, average, and ward. Single not as sensitive to outliers but susceptible to chaining effect where clusters can often not represent intuitive groups. Average strikes a balance between the two. Usually standardize data before clustering. `Sklearn.metrics` has pairwise distances.

K-Means vs Hierarchical Clustering: K-means clustering needs the number of clusters to be specified. Hierarchical clustering doesn't need the number of clusters to be specified. K-means clustering is usually more efficient run-time wise.

### 5.7.2 Assumptions

### 5.7.3 Characteristics

### 5.7.4 Evaluation

### 5.7.5 Pros/Cons

## 5.8 PCA

### 5.8.1 Intro

### 5.8.2 Assumptions

### 5.8.3 Characteristics

### 5.8.4 Evaluation

### 5.8.5 Pros/Cons

- Center and scale data. The eigenvectors yield orthogonal directions of greatest variability (principal components). The eigenvalues correspond to the magnitude of variance along the principal components. Interpretability is a negative. PCA is completely nonparametric: any data set can be plugged in and an answer comes out, requiring no parameters to tweak and no regard for how the data was recorded. From one perspective, the fact that PCA is non-parametric (or plug-and-play) can be considered a positive feature because the answer is unique and independent of the user. From another perspective the fact that PCA is agnostic to the source of the data is also a weakness. When using dimensional reduction we restrict ourselves to simpler models. Thus, we expect bias to increase and variance to decrease. Getting latent components. Visualize high dimensional data. Reduce noise. Preprocessing. scale and center data before doing. Needs regularization. Too many principal components F1 score starts to drop after increase. Best F1 is 1. set number of components. Use `fit_transforms`, use `explained_variance_ratio_` to see how much of the variance is explained by each component. Choose features that explain most of the variances, then use a `inverse_transform` to reconstruct your x. Minimum number of principal components is  $\min(n, p)$ .
- `pca`: It accounts for as much of the variability in the data as possible by considering highly correlated features. Each succeeding component in turn has the highest variance using the features that are less correlated with the first principal component and that are orthogonal to the preceding component.

### 5.8.6 Other

- 1) look at variance: `apply(USArrests, 2, var)` | 2) `pca` & scale: `pca = prcomp(USArrests, scale = TRUE)` | 3) plot: `biplot(pca, scale = 0)`
- `pca$rotation` contains the principal component loadings matrix which explains proportion of each variable along each principal component.
- Kaiser-Harris criterion suggests retaining PCs with eigenvalues  $> 1$ ; PCs with eigenvalues  $< 1$  explain less variance than contained in a single variable. Cattell Scree test visually inspects the elbow graph for diminishing return; retain PCs before a drastic drop-off.

PC columns contain loadings; correlations of the observed variables with the Pcs. h2 column displays the component communalities; amount of variance explained by the components.

Note: Look at factorplot. Shows pot of relationship between variables and principal components.

It is also possible to decompress the reduced dataset back to 784 dimensions by applying the inverse transformation of the PCA projection. Of course this won't give you back the original data, since the projection lost a bit of information (within the 5% variance that was dropped), but it will likely be quite close to the original data. The mean squared distance between the original data and the reconstructed data (compressed and then decompressed) is called the reconstruction error.

It turns out that many things behave very differently in high-dimensional space. For example, if you pick a random point in a unit square (a  $1 \times 1$  square), it will have only about a 0.4% chance of being located less than 0.001 from a border (in other words, it is very unlikely that a random point will be "extreme" along any dimension). But in a 10,000-dimensional unit hypercube (a  $1 \times 1 \times \dots \times 1$  cube, with ten thousand 1s), this probability is greater than 99.999999%. Most points in a high-dimensional hypercube are very close to the border.

Before looking at the PCA algorithm for dimensionality reduction in more detail, let's summarize the approach in a few simple steps: Standardize the  $d$ -dimensional dataset.

Construct the covariance matrix. Decompose the covariance matrix into its eigenvectors and eigenvalues. Select eigenvectors that correspond to the largest eigenvalues, where is the dimensionality of the new feature subspace  $k$ .

Construct a projection matrix from the "top"  $k$  eigenvectors.

Transform the  $d$ -dimensional input dataset using the projection matrix to obtain the new  $k$ -dimensional feature subspace.

Probabilistic PCA introduces the Gaussian distribution to the PCA modeling framework.

Kernel PCA must project data into a higher dimensional space than that of the original data.

## 5.9 KNN

### 5.9.1 Intro

Calculate distance between each observation and all the others. Determine the  $k$  nearest observations to the observation. Classify the observation as the most frequent class of the  $k$  nearest observations. Small  $k$ 's are not robust to outliers, highlight local variations and induce unstable decision boundaries. Large  $k$ 's are robust to outliers, highlight global variations and induce stable decision boundaries. Good value to choose is  $k = \sqrt{n}$ . Can use maximum prior probability to decide ties. Use Euclidean distance for numerical data and Hamming distance for categorical data. Latter treats dimensions equally and is symmetric. Low  $k$  is low bias, high variance and high  $k$  is high bias low variance. 1NN can't adapt to outliers and has no notion of class frequencies. Can use weighted KNN. Easy in sklearn.

curse of dimensionality, overfitting, correlated features, cost to update, sensitivity of distance metrics

smaller  $k$  means smaller training error, ut larger  $k$  is more stable.

## 5.9.2 Assumptions

## 5.9.3 Characteristics

## 5.9.4 Evaluation

## 5.9.5 Pros/Cons

### Pros

Only assumption is proximity. Non-parametric.

The only assumption we are making about our data is related to proximity (i. e., observations that are close by in the feature space are similar to each other in respect to the target value). We do not have to fit a model to the data since this is a non-parametric approach.

- Powerful
- No training involved (“lazy”)
- Naturally handles multiclass classification and regression

What is a common drawback of 3NN classifiers? Prediction on large data sets is slow.

### Cons

Have to decide  $k$  and distance metric. Can be sensitive to outliers or irrelevant attributes. Computationally expensive.

knn is terrible with high dimensions. Bad with categorical data.

Expensive and slow to predict new instances - Must define a meaningful distance function - Performs poorly on high-dimensionality datasets

We have to decide on  $K$  and a distance metric.

Can be sensitive to outliers or irrelevant attributes because they add noise.

Computationally expensive; as the number of observations, dimensions, and  $K$  increases, the time it takes for the algorithm to run and the space it takes to store the computations increases dramatically.

## 5.10 SVM

### 5.10.1 Intro

Selection of margin in SVM is a convex optimization problem. What is the objective of the maximum margin approach? To ensure small prediction error when the underlying distribution is not known

The LinearSVC class regularizes the bias term, so you should center the training set first by subtracting its mean. This is automatic if you scale the data using the StandardScaler. Moreover, make sure you set the loss hyperparameter to “hinge”, as it is not the default value. Finally, for better performance you should set the dual hyperparameter to False, unless there are more features than training instances (we will discuss duality later in the chapter).

With so many kernels to choose from, how can you decide which one to use? As a rule of thumb, you should always try the linear kernel first (remember that LinearSVC is much faster than SVC(kernel=“linear”)), especially if the training set is very large or if it has plenty of features. If the training set is not too large, you should try the Gaussian RBF kernel as well; it works well in most cases. Then if you have spare time and computing power, you can also experiment with a few other kernels using cross-validation and grid search, especially if there are kernels specialized for your training set’s data structure.

Here mostly talking about a maximal margin classifier. A separating hyperline of data of two classes. Maximizes distance to the nearest point in either class, i.e, the margin. Points which delineate the boundary are called support vectors. Correct classification first and then maximizing the margin. SVM only really works well with linear hyperplanes. The  $c$  hyperparameter deals with the tradeoff between a smooth decision boundary and correct classification. The larger the  $c$  the more points classified correctly as it increases the penalty for wrong classification. A smaller  $c$  gives more room for initial error which helps improve accuracy for not easily separable data. Work well in complicated domains with clear line of separation. Not so well in for large datasets because of training and not when there is a lot of noise. Can solve nonlinear decision boundaries by using polynomial terms. Kernel trick that draws boundaries by looking into higher dimensions.

Extracting the coefficients from the hyperplane equation (not including the intercept) yields what is called a normal vector. This vector points in a direction orthogonal to the surface of the hyperplane and essentially defines its orientation.

Big data set and lot of features SVM might be slow and prone to overfitting. SVM slower than naïve bayes. A kernel is a function that quantifies the similarity of two observations. The beauty of the “kernel trick” is that, even if there is an infinite-dimensional basis, we need only look at the  $n^2$  inner products between training data points. SVM not scale invariant. Check if library normalizes by default. RBF kernel is a good default. Try exponential sequences for parameters.

Extension of above info about relaxing margin. Doesn't perfectly separate classes but is more robust to outliers. Helps better classify observations. Take a penalty by potentially misclassifying some observations. Have better predictive power. Called a soft margin. The  $C$  parameter is the budget for the slack variables which tell where observations are relative to the margin and hyperplane. Only observations that either fall on the margin or violate the margin affect the solution to the optimization problem. More robust than SVM.

When the support vector classifier is combined with a non-linear kernel, the resulting classifier is called a support vector machine.

Note: Important to make sure the normal vector is of unit length. For the radial kernel, suppose we have a test observation. If it is far from a training observation, the Euclidean distance will be large, but the value of the radial kernel will be small. using kernels is much more computationally efficient because we only need to compute the kernel for distinct pairs of observations in our dataset. Furthermore, we need not actually work in the enlarged feature space. Intuitively, the gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

Multiclass

1v1: Construct a support vector machine for each pair of categories. For each classifier, record the prediction for each observation. Have the classifiers vote on the prediction for each observation.

1vAll: Construct a support vector machine for each individual category against all other categories combined. Assign the observation to the classifier with the largest function value.

## 5.10.2 Assumptions

## 5.10.3 Characteristics

## 5.10.4 Evaluation

## 5.10.5 Pros/Cons

### Pros

- Performs similarly to logistic regression when linear separation



- Performs well with non-linear boundary depending on the kernel used
- Handle high dimensional data well
- Can model complex, nonlinear relationships
- Robust to noise (because they maximize margins)

#### Cons

- Susceptible to overfitting/training issues depending on kernel
- Need to select a good kernel function
- Model parameters are difficult to interpret
- Sometimes numerical stability problems
- Requires significant memory and processing power

## 5.11 Decision Tree

### 5.11.1 Intro

So should you use Gini impurity or entropy? The truth is, most of the time it does not make a big difference: they lead to similar trees. Gini impurity is slightly faster to compute, so it is a good default. However, when they differ, Gini impurity tends to isolate the most frequent class in its own branch of the tree, while entropy tends to produce slightly more balanced trees.

Decision Trees make very few assumptions about the training data (as opposed to linear models, which obviously assume that the data is linear, for example). Such a model is often called a nonparametric model, not because it does not have any parameters (it often has a lot) but because the number of parameters is not determined prior to training, so the model structure is free to stick closely to the data.

More generally, the main issue with Decision Trees is that they are very sensitive to small variations in the training data. For example, if you just remove the widest Iris-Versicolor from the iris training set (the one with petals 4.8 cm long and 1.8 cm wide) and train a new Decision Tree, you may get the model represented in Figure 6-8. As you can see, it looks very different from the previous Decision Tree (Figure 6-2). Actually, since the training algorithm used by Scikit-Learn is stochastic<sup>6</sup> you may get very different models even on the same training data (unless you set the `random_state` hyperparameter).

Construct solutions that stratify the feature space into relatively easy to describe regions. Partition predictor space into rectangular or box-like regions. For regression predict the mean of the response variables in each region for the training observations in that space. Top down and greedy approach called recursive binary splitting to find best solution. Greedy because splits based on best result. The recursive binary splitting process depends on the greatest reduction in the RSS. Entropy very important. Decides where tree splits. Measure of impurity in examples. Entropy between 0 and 1. 0 means no impurity. Maximal is 1 when there is an even split between options. Key term: information gain. Decisions tree maximize this. Easy to interpret, prone to overfitting. Where each data point has its own node, the variance is high and the training error is 0. Deciding on the number of splits prior to building a tree isn't the best strategy. The best subtree will be the one that yields the lowest test error rate. Given a subtree, we can estimate the test error by implementing the crossvalidation process, but this is too cumbersome because the large number of possible subtrees. The tuning parameter alpha helps balance the tradeoff between the overall complexity of the tree and its fit to the training data. Small values of alpha yield trees that are quite extensive (have many terminal nodes). Large values of alpha yield trees that are quite limited (have few terminal nodes). Similar to ridge/lasso regression. Ultimately, the subtree that is used for prediction is built using all of the available data with the determined optimal value of alpha.

Regression: Use cost complexity pruning to build set of subtrees as a function of lambda. Classification: Use Gini index, measures total variance among classes.

Note: Duplicate splits happen when they increase node purity. While pruning reduces the variance of the overall tree model upon repeated builds with different datasets, we induce bias because the trees are much simpler.

### 5.11.2 Assumptions

### 5.11.3 Characteristics

### 5.11.4 Evaluation

### 5.11.5 Pros/Cons

#### Pros

- Fast
- Robust to noise and missing values
- Accurate

#### Cons

- Weak predictive accuracy and prone to overfitting.
- Complex trees are hard to interpret
- Duplication within the same sub-tree is possible

## 5.12 Bagging

### 5.12.1 Intro

used to decrease variance in decision trees

How it works: train multiple trees using bootstrapped datasets, regression trees: average the predictions from all the trees, classification: take a majority vote of the predictions, majority vote: the class which occurs most frequently, Out of bag (OOB) error, Only a portion of the data is used to train each tree, The remaining data which wasn't selected can be used to assess the tree's performance, (the response for each observation is predicted using only the trees that were not fit using that observation)

Bagging involves randomly generating Bootstrap samples from the dataset and trains the models individually. Predictions are then made by aggregating or averaging all the response variables. For example, consider a dataset  $(X_i, Y_i)$ , where  $i=1 \dots n$ , contains  $n$  data points. Now, randomly select  $B$  samples with replacements from the original dataset using Bootstrap technique. Next, train the  $B$  samples with regression/classification models independently. Then, predictions are made on the test set by averaging the responses from all the  $B$  models generated in the case of regression. Alternatively, the most often occurring class among  $B$  samples is generated in the case of classification.

Bagging stabilizes decision trees and improves accuracy by reducing variance.

Bagging reduces generalization error.

Take repeated samples of the same size from the single overall training dataset. Treat these different sets of data as pseudo-training sets. Fitting separate, independent decision trees to each of the bootstrapped

training data sets. We can then average all predictions (or take the majority vote) to obtain the bagged estimate. Instead of pruning back our trees, create very large trees in the first place. These large trees will tend to have low bias, but high variance. Retain the low bias, but get rid of the high variance by averaging. Con is that the trees are correlation. train multiple trees using bootstrapped data to reduce variance and prevent overfitting

### 5.12.2 Assumptions

### 5.12.3 Characteristics

### 5.12.4 Evaluation

### 5.12.5 Pros/Cons

#### Pros

- reduces variance in comparison to regular decision trees
- Can provide variable importance measures: classification: Gini index + regression: RSS
- Can easily handle qualitative (categorical) features
- Out of bag (OOB) estimates can be used for model validation

#### Cons

- Not as easy to visually interpret
- Does not reduce variance if the features are correlated

## 5.13 Random Forest

### 5.13.1 Intro

Build a number of decision trees using bootstrapped samples

At each split in the tree only a portion of the features are considered

a feature is selected from a subset of features not the whole feature space

the subset is usually  $\sqrt{n \cdot \text{features}}$

This allows trees to be built without some of the strong features decorrelates the trees, unlike bagging

Random forests are improvised supervised algorithms than bootstrap aggregation or bagging methods, though they are built on a similar approach. Unlike selecting all the variables in all the B samples generated using the Bootstrap technique in bagging, we select only a few predictor variables randomly from the total variables for each of the B samples. Then, these samples are trained with the models. Predictions are made by averaging the result of each model. The number of predictors in each sample is decided using the formula  $m = \sqrt{p}$ , where p is the total variable count in the original dataset. Here are some key notes: This approach removes the condition of dependency of strong predictors in the dataset as we intentionally select fewer variables.

Draw a random bootstrap sample of size n (randomly choose n samples from the training set with replacement). Grow a decision tree from the bootstrap sample. At each node: Randomly select d features without replacement. Split the node using the feature that provides the best split according to the objective function, for instance, by maximizing the information gain. Repeat the steps 1 to 2 k times. Aggregate the prediction

by each tree to assign the class label by majority vote. Majority voting will be discussed in more detail in Chapter 7, Combining Different Models for Ensemble Learning.

Random forests further improve decision tree performance by de-correlating the individual trees in the bagging ensemble.

Random forests decorrelate tree it generates and thus results in a reduction in variance. Similar to bagging, we first build various decision trees on bootstrapped training samples, but we split internal nodes in a special way. Each time a split is considered within the construction of a decision tree, only a random subset of the overall predictors are allowed to be candidates. At every split, a new subset of predictors is randomly selected. The random forest procedure forces the decision tree building process to use different predictors to split at different times. Should a good predictor be left out of consideration for some splits, it still has many chances to be considered in the construction of other splits. We can't overfit by adding more trees. The variance just ends up decreasing. Out of bag score is a good estimate of the test score and is the non-bootstrapped data. Need to turn on this scoring method. Error Rate: Depends on correlation between trees (higher is worse), strength of single trees (higher is better). Increasing number of features for each split increases correlation and strength of single trees.

### 5.13.2 Assumptions

### 5.13.3 Characteristics

### 5.13.4 Evaluation

### 5.13.5 Pros/Cons

#### Pros

Decorrelates trees (relative to boosted trees)

important when dealing with multiple features which may be correlated

reduced variance (relative to regular trees)

All purpose model that performs well on most problems. Automatically chooses the best features. High accuracy.

#### Cons

Hard to interpret and needs a lot of work to tune parameters.

Not as easy to visually interpret

## 5.14 Boosting

### 5.14.1 Intro

Unlike bagging and random forests, can overfit if number of trees is too large

Similar to bagging, but trees are grown sequentially. Each tree is created using information from previously grown trees

Each tree is generated using information from previously grown trees; the addition of a new tree improves upon the performance of the previous trees. The trees are now dependent upon one another. Boosting approach tends to slowly learn our data. Given a current decision tree model, we fit a new decision tree to the residuals of the current decision tree. The new decision tree (based on the residuals) is then added to

the current decision tree, and the residuals are updated. We limit the number of terminal nodes in order to sequentially fit small trees. By fitting small trees to the residuals, we slowly improve the overall model in areas where it does not perform well.

Note: Unlike in bagging and random forests, boosting can overfit if  $\alpha$  is large (although very slowly). Use cross-validation to select number of trees.  $\alpha$  controls the rate at which boosting learns and is usually between 0.01 to 0.001. A small  $\alpha$  usually goes with a large number of trees. Can also choose the number of splits. Typically using stumps (single splits  $d = 1$ ) is sufficient and results in an additive model.

Note on Variable Importance: For regression trees, we can use the reduction in the RSS. For classification trees, we can use the reduction in the Gini index. A relatively large value indicates a notable drop in the RSS or Gini index, and thus a better fit to the data; corresponding variables are relatively important predictors.

Similar to bagging, but learns sequentially and builds off previous trees

### 5.14.2 Assumptions

### 5.14.3 Characteristics

### 5.14.4 Evaluation

### 5.14.5 Pros/Cons

#### Pros

Somewhat more interpretable than boosted trees/random forest as the user can define the size of each tree resulting in a collection of stumps (1 level) which can be viewed as an additive model

Can easily handle qualitative (categorical) features

#### Cons

## 5.15 Association Rule Mining

### 5.15.1 Intro

- Used in Genetics, Fraud, and Market Basket Analysis, etc. A typical rule might be: if someone buys peanut butter and jelly, then that person is likely to buy bread as well.
- Incredibly big feature space ( $2^k - 1$ ).
- The Apriori algorithm employs a simple a priori belief as a guideline for reducing the association rule space.
- Support: the fraction of which each item appears within the dataset as a whole.  $\text{Support}(\text{item}) = \text{count}(\text{item})/N$ . Higher support is better.
- Confidence: the likelihood that a constructed rule is correct given the items on the left hand side of the transaction. A higher level of confidence implies a higher likelihood that Y appears alongside transactions in which X appears.
- Lift: the ratio by which the confidence of a rule exceeds the expected outcome. When  $\text{lift} > 1$ , the presence of X seems to have increased the probability of Y occurring in the transaction. When  $\text{lift} < 1$ , the presence of X seems to have decreased the probability of Y occurring in the transaction. When  $\text{lift} = 1$ , X and Y are independent.

- $t$  thresholds for support and confidence and then the algorithm goes from all 1-combinations to 2-combinations and up. Those subset below the threshold don't make it to the higher iterations.

### 5.15.2 Assumptions

### 5.15.3 Characteristics

### 5.15.4 Evaluation

### 5.15.5 Pros/Cons

#### Pros

- Ideally suited for working with very large amounts of transactional data.
- The results are rules that are generally easy to understand and have a high amount of interpretability.
- The process is useful for data mining and uncovering unexpected knowledge within a dataset.

#### Cons

- The outcome is usually not interesting when applied to smaller datasets.
- It is difficult to separate actual insights from common sense notions.
- The analyst might be compelled to draw spurious conclusions—remember that correlation doesn't imply causation!

## 5.16 Naïve Bayes

### 5.16.1 Intro

Practitioners do use Naive Bayes regularly for ranking, where the actual values of the probabilities are not relevant—only the relative values for examples in the different classes. Another advantage of Naive Bayes is that it is naturally an “incremental learner.” An incremental learner is an induction technique that can update its model one training example at a time. It does not need to reprocess all past training examples when new training data become available.

Document/spam classification is one use. Assumptions are that all the features are equally likely and are all important. Would be computational expensive without these assumptions with having to track all the joint probabilities. Gaussian for continuous variables, Bernoulli for binary input, and Multinomial for binary and more input. Use MLE to estimate parameters for Gaussian. Bernoulli/Multinomial used for spam classification. The Laplace Estimator (usually chosen to be 1) is a corrective measure that adds a small amount of error to each of the counts in the frequency table of words. The addition of error ensures that each resulting probability of each event will necessarily be nonzero, even if the event did not appear in the training data.

Note: In addition, we can use function “`partial_fit`” to fit on a batch of samples incrementally while we are using. MultinomialNB and Bernoulli NB also support sample weighting.

### 5.16.2 Assumptions

### 5.16.3 Characteristics

### 5.16.4 Evaluation

### 5.16.5 Pros/Cons

#### Pros

- It is relatively simple to understand. Training the classifier does not require many observations, and the method also works well with large amounts of data. It is easy to obtain the estimated probability for a classification prediction.
- Computationally fast
- Simple to implement
- Works well with high dimensions

#### Cons

- Relies on independence assumption and will perform badly if this assumption is not met
- While easily attainable, the estimated probabilities are often less reliable than the predicted class labels themselves.

## 5.17 LDA

### 5.17.1 Intro

Before we take a look into the inner workings of LDA in the following subsections, let's summarize the key steps of the LDA approach:

Standardize the  $d$ -dimensional dataset ( $d$  is the number of features).

For each class, compute the  $d$ -dimensional mean vector.

Construct the between-class scatter matrix and the within-class scatter matrix  $S_w$ .

Compute the eigenvectors and corresponding eigenvalues of the matrix  $S_w^{-1}S_b$ .

Choose the eigenvectors that correspond to the largest eigenvalues to construct a  $k$ -dimensional transformation matrix  $W$ ; the eigenvectors are the columns of this matrix.

Project the samples onto the new feature subspace using the transformation matrix  $W$ .

LDA similar to Naive Bayes but doesn't assume variables are independent, assumes that  $p(x|y)$  is a multivariate normal distribution, and assumes gaussian distributions for each class share the same covariance matrix. If they don't share the same covariance matrix, then we use Quadratic Discriminant Analysis which assumes each class has its own.

### 5.17.2 Assumptions

### 5.17.3 Characteristics

### 5.17.4 Evaluation

### 5.17.5 Pros/Cons

## 5.18 QDA

### 5.18.1 Intro

TBD

### 5.18.2 Assumptions

### 5.18.3 Characteristics

### 5.18.4 Evaluation

### 5.18.5 Pros/Cons

## 5.19 Model Selection

### 5.19.1 General

- Linear SVM often outperforms logistic regression and is explainable as well.
- However, the problem with cross-validation is that it is rarely applicable to real world problems, for all the reasons described in the above sections. Cross-validation only works in the same cases where you can randomly shuffle your data to choose a validation set.
- If the performance of a classification model on the test set (out-of-sample) error is poor, you can't just re-calibrate your model parameters to achieve a better model. This is cheating.
- When the classes are well-separated, the parameter estimates for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem. If  $n$  is small and the distribution of the predictors  $X$  is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.
- Roughly speaking, LDA tends to be a better bet than QDA if there are relatively few training observations and so reducing variance is crucial. In contrast, QDA is recommended if the training set is very large, so that the variance of the classifier is not a major concern, or if the assumption of a common covariance matrix for the  $K$  classes is clearly untenable.
- Two features interact if the effect on the prediction of one feature depends on the value of the other feature
- “No Free Lunch” theorem: no single classifier works best across all possible scenarios
- `Warning PolynomialFeatures(degree=d)` transforms an array containing  $n$  features into an array containing features, where  $n!$  is the factorial of  $n$ , equal to  $1 \times 2 \times 3 \times \dots \times n$ . Beware of the combinatorial explosion of the number of features!



- If a model suffers from overfitting, we also say that the model has a high variance, which can be caused by having too many parameters that lead to a model that is too complex given the underlying data. Similarly, our model can also suffer from underfitting (high bias), which means that our model is not complex enough to capture the pattern in the training data well and therefore also suffers from low performance on unseen data.
- It appears that each feature has modestly improved our model. There are certainly more features that we could add to our model. For example, we could add the day of the week and the hour of the posting, we could determine if the article is a listicle by running a regex on the headline, or we could examine the sentiment of each article. This only begins to touch on the features that could be important to model virality. We would certainly need to go much further to continue reducing the error in our model.
- Suppose Fixitol reduces symptoms by 20% over a placebo, but the trial you're using to test it is too small to have adequate statistical power to detect this difference reliably. We know that small trials tend to have varying results; it's easy to get 10 lucky patients who have shorter colds than usual but much harder to get 10,000 who all do.
- As I mentioned earlier, one drawback of the Bonferroni correction is that it greatly decreases the statistical power of your experiments, making it more likely that you'll miss true effects. More sophisticated procedures than Bonferroni correction exist, ones with less of an impact on statistical power, but even these are not magic bullets. Worse, they don't spare you from the base rate fallacy. You can still be misled by your p threshold and falsely claim there's "only a 5% chance I'm wrong." Procedures like the Bonferroni correction only help you eliminate some false positives.
- In step 1, we find where missing values are located. The `md.pattern()` function from the `mice` package is a really useful function. It gives you a clear view of where missing values are located, helping you in decisions regarding exclusions or substitution. You can refer to the next recipe for missing value substitution.
- "Product classification is an example of multicategory or multinomial classification. Most classification problems and most classification algorithms are specialized for two-category, or binomial, classification. There are tricks to using binary classifiers to solve multicategory problems (for example, building one classifier for each category, called a "one versus rest" classifier). But in most cases it's worth the effort to find a suitable multiple-category implementation, as they tend to work better than multiple binary classifiers (for example, using the package `mlogit` instead of the base method `glm()` for logistic regression)."
- Another possible indication of collinearity in the inputs is seeing coefficients with an unexpected sign: for example, seeing that income is negatively correlated with years in the workforce.
- The overall model can still predict income quite well, even when the inputs are correlated; it just can't determine which variable deserves the credit for the prediction. Using regularization (especially ridge regression as found in `lm.ridge()` in the package `MASS`) is helpful in collinear situations (we prefer it to "x-alone" variable preprocessing, such as principal components analysis). If you want to use the coefficient values as advice as well as to make good predictions.
- The degrees of freedom is thought of as the number of training data rows you have after correcting for the number of coefficients you tried to solve for. You want the degrees of freedom to be large compared to the number of coefficients fit to avoid overfitting.
- Overfitting is when you find chance relations in your training data that aren't present in the general population. Overfitting is bad: you think you have a good model when you don't.
- The probable reason for nonconvergence is separation or quasi-separation: one of the model variables or some combination of the model variables predicts the outcome perfectly for at least a subset of the training data. You'd think this would be a good thing, but ironically logistic regression fails when the variables are too powerful.

- For categorical variables (male/female, or small/medium/large), you can define the distance as 0 if two points are in the same category, and 1 otherwise. If all the variables are categorical, then you can use Hamming distance, which counts the number of mismatches.
- Your next step is to select a performance measure. A typical performance measure for regression problems is the Root Mean Square Error (RMSE). It measures the standard deviation of the errors the system makes in its predictions.
- You should save every model you experiment with, so you can come back easily to any model you want. Make sure you save both the hyperparameters and the trained parameters, as well as the cross-validation scores and perhaps the actual predictions as well. This will allow you to easily compare scores across model types, and compare the types of errors they make. You can easily save Scikit-Learn models by using Python's pickle module, or using `sklearn.externals.joblib`, which is more efficient at serializing large NumPy arrays.
- If all classifiers are able to estimate class probabilities (i.e., they have a `predict_proba()` method), then you can tell Scikit-Learn to predict the class with the highest class probability, averaged over all the individual classifiers. This is called soft voting. It often achieves higher performance than hard voting because it gives more weight to highly confident votes. All you need to do is replace `voting="hard"` with `voting="soft"` and ensure that all classifiers can estimate class probabilities.
- The standard value for  $k$  in  $k$ -fold cross-validation is 10, which is typically a reasonable choice for most applications. However, if we are working with relatively small training sets, it can be useful to increase the number of folds. If we increase the value of  $k$ , more training data will be used in each iteration, which results in a lower bias towards estimating the generalization performance by averaging the individual model estimates. However, large values of  $k$  will also increase the runtime of the cross-validation algorithm and yield estimates with higher variance since the training folds will be more similar to each other. On the other hand, if we are working with large datasets, we can choose a smaller value for  $k$ .
- Distance Metrics: Hamming distance for categorical variables, Cosine distance for text data, Gower distance for categorical data
- AIC x BIC: Explanatory power x Parsimonious
- Bias is how far off on the average the model is from the truth. Variance is how much the estimate varies about its average. With low model complexity bias is high because predictions are more likely to stray from the truth, and variance is low because there are only few parameters being fit. With high model complexity bias is low because the model can adapt to more subtleties in the data, and variance is high because we have more parameters to estimate from the same amount of data.
- The mean of highly correlated quantities has higher variance than the mean of uncorrelated quantities. The smaller the number of folds, the more biased the error estimates (they will be biased to be conservative indicating higher error than there is in reality) but the less variable they will be. On the extreme end you can have one fold for each data point which is known as Leave-One-Out-Cross-Validation. In this case, your error estimate is essentially unbiased but it could potentially have high variance.
- LDA vs Logistic Regression: When the classes are well-separated parameters of logistic regression are unstable and LDA doesn't have this problem. If the distribution of  $X$  is approximately normal then LDA is more stable again. LDA doesn't need to fit multiple models for multiclass classification.
- SVM vs Logistic Regression: SVM is likely to overfit the data and can dominate LR on the training set. Kernel method makes SVM more flexible.
- Decision Tree vs Linear Model: Tree has non-linear boundary, selects important features, easier to interpret but unstable especially for small data.
- Naive Bayes vs LDA: Both use Bayes theorem, both assume gaussian distribution, but LDA takes care of the correlations. LDA might out-perform Naive Bayes when the features are highly correlated.

- KNN vs Other Classification Methods: KNN is completely non-parametric, doesn't tell which features are important. Dominates LDA and Logistic Regression when the decision boundary is highly non-linear.
- When building, testing, and validating a suite of models, the optimal model (i.e., the optimal point in the ROC curve) is the spot where the overall accuracy of the model is essentially unchanged as we make small adjustments in the choice of model. That is to say, the change (from one model to the next) in the model's precision (specificity) is exactly offset by the change in the model's recall (sensitivity). Consequently, allowing for some imperfection, where the false positive rate and the false negative rate are in proper balance (so that neither one has too great of an impact on the overall accuracy and effectiveness of the model), is good fruit from your big data labors. The metric I used to guide my cross-validation is the F-score. This is a good metric when we have a lot more samples from one category than from the other categories.
- The cost of the holdout method comes in the amount of data that is removed from the model training process. For instance, in the illustrative example here, we removed 30% of our data. This means that our model is trained on a smaller data set and its error is likely to be higher than if we trained it on the full data set. The standard procedure in this case is to report your error using the holdout set, and then train a final model using all your data.
- A metric that minimizes false positives, by rarely flagging players and teams that fail to achieve the desired outcome, is specific. A metric that minimizes false negatives, by rarely failing to flag players and teams that achieve the desired outcome, is sensitive.
- Although the previous code example was useful to illustrate how k-fold cross-validation works, scikit-learn also implements a k-fold cross-validation scorer, which allows us to evaluate our model using stratified k-fold cross-validation more efficiently
- Sometimes it may be more useful to report the coefficient of determination ( $R^2$ ), which can be understood as a standardized version of the MSE, for better interpretability of the model performance. In other words, is the fraction of response variance that is captured by the model. The value is defined as follows:  $R^2 = 1 - \frac{SSE}{SST}$ , where SSE is the sum of squared errors and SST is the total sum of squares, or in other words, it is simply the variance of the response.
- Most people start working with data from exactly the wrong end. They begin with a data set, then apply their favorite tools and techniques to it. The result is narrow questions and shallow arguments. Starting with data, without first doing a lot of thinking, without having any structure, is a short road to simple questions and unsurprising results. We don't want unsurprising—we want knowledge.
- Remember that the positive class in scikit-learn is the class that is labeled as class 1. If we want to specify a different positive label, we can construct our own scorer via the `make_scorer` function, which we can then directly provide as an argument to the scoring parameter in `GridSearchCV`
- While the weighted macro-average is the default for multiclass problems in scikit-learn, we can specify the averaging method via the `average` parameter inside the different scoring functions that we import from the `sklearn.metrics` module, for example, the `precision_score` or `make_scorer` functions
- Transforming words into feature vectorsTo construct a bag-of-words model based on the word counts in the respective documents, we can use the `CountVectorizer` class implemented in scikit-learn. As we will see in the following code section, the `CountVectorizer` class takes an array of text data, which can be documents or just sentences, and constructs the bag-of-words model for us:
- Scikit-learn implements yet another transformer, the `TfidfTransformer`, that takes the raw term frequencies from `CountVectorizer` as input and transforms them into tf-idfs.
- Naive Bayes is a linear classifier, while k-NN is not. The curse of dimensionality and large feature sets are a problem for k-NN, while Naive Bayes performs well. k-NN requires no training (just load in the dataset), whereas Naive Bayes does.

### 5.19.2 Clustering

- Comparison
- Rules
- <https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68>
- <http://hunch.net/?p=224>

## Chapter 6

# Communicate, Deploy, Maintain

### 6.1 Communicate

Summarize the motivation and goals of the project. It should either be an executive summary or a technical abstract.

State results with details as needed. For an end user presentation show how the model fits into and improves the user's workflow, and how to use it.

Discuss recommendations, outstanding issues, and possible future work.

Introduction | Motivation | Summary | Results | Conclusion

### 6.2 Deploy

### 6.3 Maintain



# Chapter 7

## Frameworks

### 7.1 PySpark

A [cheatsheet](<https://www.qubole.com/resources/pyspark-cheatsheet/>).

Pipeline

Setup

```
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
sc = pyspark.SparkContext()
spark = SparkSession.builder.appName('example').getOrCreate()
import findspark
findspark.init()
import pyspark
import random
sc = pyspark.SparkContext(appName="Pi")
num_samples = 100000000
def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1
count = sc.parallelize(range(0, num_samples)).filter(inside).count()
pi = 4 * count / num_samples
print(pi)
sc.stop()
```

Libs

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
import pyspark.ml.evaluation as evals
import pyspark.ml.tuning as tune
import numpy as np
df = spark.read.csv('data.csv', header = True)
df.show(5)
df.columns
```

## Preprocessing

```
df = df.withColumn("label", df["target"].cast('integer'))
scf_indexer = StringIndexer(inputCol = "some_cat_feature", outputCol = "some_cat_feature_index")
scf_encoder = OneHotEncoder(inputCol = "some_cat_feature_index", outputCol = "some_cat_feature_fact")
feature_cols = ["some list of column names"]
vec_assembler = VectorAssembler(inputCols = feature_cols,
                                outputCol = "features")

pipe = Pipeline(stages = [scf_indexer, scf_encoder, vec_assembler])
piped_data = pipe.fit(df).transform(df)
training, test = piped_data.randomSplit([.8, .2])
```

## Model

```
clf_lr = LogisticRegression()
evaluator = evals.BinaryClassificationEvaluator(metricName = "areaUnderROC")
grid = tune.ParamGridBuilder()
grid = grid.addGrid(clf_lr.regParam, np.arange(0, .1, .01))
grid = grid.addGrid(clf_lr.elasticNetParam, [0, 1])
grid = grid.build()
clf_lr_cv = tune.CrossValidator(
    estimator = clf_lr,
    estimatorParamMaps = grid,
    evaluator = evaluator,
    numFolds = 5
)
best_clf_lr = clf_lr_cv.fit(training).bestModel
results = best_clf_lr.transform(training)
print(evaluator.evaluate(results))
#stop
sc.stop()
```

## Quick Test:

```
import findspark
findspark.init()
import pyspark
import random
sc = pyspark.SparkContext(appName="Pi")
num_samples = 100000000
def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1
count = sc.parallelize(range(0, num_samples)).filter(inside).count()
pi = 4 * count / num_samples
print(pi)
sc.stop()
```

First have to create a SparkSession object from your SparkContext. The SparkContext is ther connection to the cluster and the SparkSession as your interface with that connection.

```
import pyspark
from pyspark.sql import SparkSession
sc = pyspark.SparkContext()
spark = SparkSession.builder.getOrCreate() # SparkSession.builder.appName('chosenName').getOrCreate()
```

The rest of my notes:



```

## List tables
spark.catalog.listTables()
## Access and display data
query = "FROM flights SELECT * LIMIT 10"
flights10 = spark.sql(query)
flights10.show()
## Cast
df['g'] = df['g'].astype(str)
## Read from csv
df = spark.read.csv(file_path, header = True)
df = spark.read.csv('fileNameWithPath', mode="DROPMALFORMED", inferSchema=True, header = True)
df = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("john_doe.csv")
## Spark df to pandas and vice versa
toPandas()
spark_temp = spark.createDataFrame(pd_temp)
## Add to the catalog
spark_temp.createOrReplaceTempView("temp")
## Mutate
df = df.withColumn("newCol", df.oldCol + 1)
model_data = model_data.withColumn("arr_delay", model_data.arr_delay.cast('integer'))
model_data = model_data.withColumn("plane_age", model_data.year - model_data.plane_year)
## Create the DataFrame flights
flights = spark.table('flights')
## Get column names
spark_df.schema.names
spark_df.printSchema()
## Filter: takes either a Spark Column of boolean (True/False) values or the WHERE clause of a SQL expr
long_flights1 = flights.filter('distance > 1000')
long_flights2 = flights.filter(flights.distance > 1000)
model_data = model_data.filter("arr_delay is not NULL and dep_delay is not NULL and air_time is not NULL")
## Groupby examples
flights.filter(flights.origin == "PDX").groupBy().min("distance").show()
flights.filter(flights.carrier=="DL").filter(flights.origin=="SEA").groupBy().avg('air_time').show()
flights.withColumn("duration_hrs", flights.air_time/60).groupBy().sum('duration_hrs').show()
## Spark functions
import pyspark.sql.functions as F
by_month_dest.agg(F.stddev('dep_delay')).show()

## Drop column
final_test_data.drop('State')
## Dummying
#The first step to encoding your categorical feature is to create a StringIndexer. Members of this class
#The second step is to encode this numeric column as a one-hot vector using a OneHotEncoder. This works
## Create a StringIndexer
carr_indexer = StringIndexer(inputCol="carrier", outputCol="carrier_index")
## Create a OneHotEncoder
carr_encoder = OneHotEncoder(inputCol="carrier_index", outputCol="carrier_fact")
## Make a VectorAssembler
vec_assembler = VectorAssembler(inputCols=["month", "air_time", "carrier_fact", "dest_fact", "plane_age"])
## Import & Make Pipeline
from pyspark.ml import Pipeline
flights_pipe = Pipeline(stages=[dest_indexer, dest_encoder, carr_indexer, carr_encoder, vec_assembler])
## Fit and transform the data
piped_data = flights_pipe.fit(model_data).transform(model_data)

```

```

## Train-test split
training, test = piped_data.randomSplit([.6, .4])
## Tuning & Selection
## Import LogisticRegression
from pyspark.ml.classification import LogisticRegression
## Create a LogisticRegression Estimator
lr = LogisticRegression()
## Import the evaluation submodule
import pyspark.ml.evaluation as evals
## Create a BinaryClassificationEvaluator
evaluator = evals.BinaryClassificationEvaluator(metricName="areaUnderROC")
## Import the tuning submodule
import pyspark.ml.tuning as tune
## Create the parameter grid
grid = tune.ParamGridBuilder()
## Add the hyperparameter
grid = grid.addGrid(lr.regParam, np.arange(0, .1, .01))
grid = grid.addGrid(lr.elasticNetParam, [0, 1])
## Build the grid
grid = grid.build()
## Create the CrossValidator
cv = tune.CrossValidator(
    estimator=lr,
    estimatorParamMaps=grid,
    evaluator=evaluator
)
## Fit cross validation models
models = cv.fit(training)
## Extract the best model
best_lr = models.bestModel
## Use the model to predict the test set
test_results = best_lr.transform(test)
## Evaluate the predictions
print(evaluator.evaluate(test_results))

```

- Spark’s assumes the target is called ‘label’ in ML. Everything else is a feature.
- `alias()` method to rename a column you’re selecting.
- `cast()` method: nominative determinism on columns
- `withColumn()`: create new column
- `pyspark.ml.feature`
- At the core of the `pyspark.ml` module are the Transformer and Estimator classes. Transformer classes are used for pre-processing and have a `.transform()` method. eg. PCA
- You can create what are called ‘one-hot vectors’ to represent the carrier and the destination of each flight. A one-hot vector is a way of representing a categorical feature where every observation has a vector in which all elements are zero except for at most one element, which has a value of one (1).
- Estimator classes are for modeling and all implement a `.fit()` method. eg. `StringIndexerModel` for including categorical data saved as strings in your models
- `selectExpr()` takes SQL expressions as a string
- `show()` vs `collect()`

- Spark only handles numeric data
- In Spark it's important to make sure you split the data after all the transformations. This is because operations like `StringIndexer` don't always produce the same index even when given the same list of strings.

## 7.2 Sparklyr

```
library(sparklyr)

## Connect to your Spark cluster
spark_conn <- spark_connect(master = "local")

## Print the version of Spark
spark_version(sc = spark_conn)

## Disconnect from Spark
spark_disconnect(sc = spark_conn)

## See tables
src_tbls(sc)

## Copy track_metadata to Spark
track_metadata_tbl <- copy_to(spark_conn, track_metadata)

## Print 5 rows, all columns
print(track_metadata_tbl, n = 5, width = Inf)

## Write and run SQL query
query <- "SELECT * FROM track_metadata WHERE year < 1935 AND duration > 300"
(results <- dbGetQuery(spark_conn, query))

## General transformation structure and example
a_tibble %>%
  ft_some_transformation("x", "y", some_other_args)

hotttnesss <- track_metadata_tbl %>%
  # Select artist_hotttnesss
  select(artist_hotttnesss) %>%
  # Binarize to is_hottt_or_nottt
  ft_binarizer('artist_hotttnesss', 'is_hottt_or_nottt', threshold = .5) %>%
  # Collect the result
  collect() %>%
  # Convert is_hottt_or_nottt to logical
  mutate(is_hottt_or_nottt = as.logical(is_hottt_or_nottt))

## Get and transform the schema
(schema <- sdf_schema(track_metadata_tbl))
schema %>%
  lapply(function(x) do.call(data_frame, x)) %>%
  bind_rows()
```

```
## Train-test split
partitioned <- track_metadata_tbl %>%
  sdf_partition(training = 0.7, testing = 0.3)

## List ml functions
ls("package:sparklyr", pattern = "^ml")

## GBT Example

gradient_boosted_trees_model <- track_data_to_model_tbl %>%
  # Run the gradient boosted trees model
  ml_gradient_boosted_trees('year', feature_colnames)

responses <- track_data_to_predict_tbl %>%
  # Select the year column
  select(year) %>%
  # Collect the results
  collect() %>%
  # Add in the predictions
  mutate(
    predicted_year = predict(
      gradient_boosted_trees_model,
      track_data_to_predict_tbl
    )
  )
)
```

### Parquet Aside

Parquet files are quicker to read and write. parquet files can be used with other tools in the Hadoop ecosystem, like Shark, Impala, Hive, and Pig.

```
## The parquet_dir has been pre-defined
parquet_dir

## List the files in the parquet dir
filenames <- dir(parquet_dir, full.names = TRUE)

## Show the filenames and their sizes
data_frame(
  filename = basename(filenames),
  size_bytes = file.size(filenames)
)

#Import the data into Spark

timbre_tbl <- spark_read_parquet(spark_conn, 'timbre', parquet_dir)
```

- To collect your data: that is, to move it from Spark to R, you call `collect()`. `copy_to()` moves your data from R to Spark.
- Use `compute()` to compute the calculation, but store the results in a temporary data frame on Spark.
- If you want to delay returning the data, you can use `dbSendQuery()` to execute the query, then `dbFetch()` to return the results.
- feature transforms: `ft__`, ml functions: `ml__`, spark df functions: `sdf__`

- `ft_tokenizer()`: to lower and split into individual words, `ft_regex_tokenizer`, `sdf_sort`

## 7.3 Caret

`attributes(clf_lr)`: results, finalModel

### 7.3.1 Preprocessing

```
# Apply median imputation: model
model <- train(
  x = breast_cancer_x,
  y = breast_cancer_y,
  method = 'glm',
  trControl = myControl,
  preProcess = c('medianImpute', "knnImpute", 'center', 'scale', 'pca')
)

dotplot(resamples, metric = "ROC")

min(model$results$RSME)

# Identify near zero variance predictors: remove_cols
remove_cols <- nearZeroVar(bloodbrain_x, names = TRUE,
                           freqCut = 2, uniqueCut = 20)

# Get all column names from bloodbrain_x: all_cols
all_cols = colnames(bloodbrain_x)

# Remove from data: bloodbrain_x_small
bloodbrain_x_small <- bloodbrain_x[ , setdiff(all_cols, remove_cols)]
```

### 7.3.2 LM

```
# Set seed
set.seed(42)

# Fit lm model: model
model <- lm(price ~ ., data = diamonds)

# Predict on full data: p
p <- predict(model, diamonds)

# Compute errors: error
error <- p - diamonds$price

# Calculate RMSE
sqrt(mean(error^2))

# Fit lm model using 5 x 5-fold CV: model
```

```

model <- train(
  medv ~ .,
  Boston,
  method = "lm",
  trControl = trainControl(
    method = "cv", number = 5,
    repeats = 5, verboseIter = TRUE
  )
)

```

### 7.3.3 Lasso/Ridge

Alpha controls balance between lasso and ridge. Lambda controls penalty.

```

# Fit glmnet model: model
model <- train(
  y~., data = overfit,
  method = "glmnet",
  trControl = myControl
)

# Train glmnet with custom trainControl and tuning: model
model <- train(
  y~., overfit,
  tuneGrid = expand.grid(alpha = 0:1, lambda = seq(0.0001, 1, length = 20)),
  method = 'glmnet',
  trControl = myControl
)

# Print model to console
model

# Print maximum ROC statistic
max(model[["results"]][["ROC"]])

```

### 7.3.4 Logistic Regression

```

# Fit glm model
model = glm(Class ~ ., family = "binomial", train)

# Predict on test: p
p = predict(model, test, type = "response")

# Calculate class probabilities: p_class
p_class <- ifelse(p > 0.50, "M", "R")

# Create confusion matrix
confusionMatrix(p_class, test$Class)

```

### 7.3.5 Train-Test Split, Folds, & CV

```
# Shuffle row indices: rows
rows = sample(nrow(Sonar))

# Randomly order data: Sonar
Sonar = Sonar[rows,]

# Identify row to split on: split
split <- round(nrow(Sonar) * .6)

# Create train: 60%
train = Sonar[1:split,]

# Create test: 40%
test = Sonar[(split+1):nrow(Sonar),]

# Create custom indices: myFolds
myFolds <- createFolds(churn_y, k = 5)

# Create reusable trainControl object: myControl
myControl <- trainControl(
  summaryFunction = twoClassSummary, #default option is default summary
  classProbs = TRUE, # IMPORTANT!
  verboseIter = TRUE,
  savePredictions = TRUE,
  index = myFolds
)
```

### 7.3.6 Random Forest

```
# Fit random forest: model
model <- train(
  quality~.,
  tuneLength = 3,
  #tuneGrid = data.frame(mtry = c(2, 3, 7)),
  data = wine,
  method = 'ranger',
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE)
)

# Print model to console
model

# Plot model
plot(model)
```

### 7.3.7 SVM

```

clf_svm <- train(
  pass ~.,
  data = df3,
  method = "svmLinear",
  preProcess = c("center", "scale")
)

```

### 7.3.8 Model Selection

Now that you have fit two models to the churn dataset, it's time to compare their out-of-sample predictions and choose which one is the best model for your dataset.

You can compare models in caret using the `resamples()` function, provided they have the same training data and use the same `trainControl` object with preset cross-validation folds. `resamples()` takes as input a list of models and can be used to compare dozens of models at once (though in this case you are only comparing two models).

```

# Create model_list
model_list <- list(item1 = model_glmnet, item2 = model_rf)

# Pass model_list to resamples(): resamples
resamples = resamples(model_list)

# Summarize the results
summary(resamples)

# Create bwplot
bwplot(resamples, metric = 'ROC')

# Create xyplot
xyplot(resamples)

# Predict on test: p
p = predict(model, test, type = 'response')

# Make ROC curve
colAUC(p, test$Class, plotROC = TRUE)

```

- caret: finalModel of model object

### 7.3.9 Stacking

`caretEnsemble` provides the `caretList()` function for creating multiple caret models at once on the same dataset, using the same resampling folds. You can also create your own lists of caret models. Use the `caretStack()` function to make a stack of caret models, with the two sub-models (glmnet and ranger) feeding into another (hopefully more accurate!) caret model.

```

# Create ensemble model: stack
stack <- caretStack(model_list, method = 'glm')

# Look at summary
summary(stack)

```

& caretEnnsemble



## 7.4 Data Table

- Operations done by reference
- `dt[i, j, by]` : subset by i calculate by j grouped using by
- `1: numeric | 1L: integer`
- `NA_integer_`: integer
- `DT[N]` : prints last row
- `names(DT)`: colnames
- `dim(DT)`: dimensions
- `DT[, .(A, B)]`: returns two columns
- `DT[, c(A, B)]`: returns a concatenated vector
- `DT[, .(sum_c = sum(C))]`
- `DT[, plot(A, C)]`
- `DT[, A:=NULL]`: Remove column A
- `DT[, .(sumB = sum(B)), by = .(Grp = A%%2)]`
- `DT[, N, by = Sepal.Width]`: .N is the count of each group
- `DT[, lapply(.SD, median)]`: .SD is a placeholder for all the columns
- `dogs[, lapply(.SD, mean), .SDcols = 2:3]`: Find mean of columns 2 & 3
- `for (i in 1:5) set(DT, i, 3L, i+1)`: update first 5 rows of 3rd column
- `setnames(DT, 'y', 'z')`: changes colname from y to z
- `setkey(DT, A, B)`
- `DT[.('b')]`
- `DT[(c('b', 'c'))]`
- `DT[(c('b', 'c')), mult="first"]`
- `DT[c("b", "c"), .SD[c(1, .N)], by = .EACHI]`: First and last row of the "b" and "c" groups
- `DT[c("b", "c"), { print(.SD); .SD[c(1, .N)] }, by = .EACHI]`
- `roll=TRUE, 'nearest', -Inf, Inf | rollends=FALSE`

## 7.5 Keras

Here's a Hello World and the general framework:

- declare: sequential
- add layers input-hidden-output
- compile
- fit

Keras only uses numpy arrays.

- `keras.callbacks.EarlyStopping`: Stop training when validation score stop improving after a certain number of epochs (batches?)

### 7.5.1 Regression

- `loss = mean_squared_error`
- `metric: rmse`
- `activation_function = relu`

```
import keras
from keras.layers import Dense
from keras.models import Sequential
# Specify the model: Two hidden layers
model = Sequential()
## Input
n_cols = predictors.shape[1]
model.add(Dense(50, activation='relu', input_shape = (n_cols,)))
# Hidden
model.add(Dense(32, activation='relu'))
# Output
model.add(Dense(1))
# Compile the model
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
# Fit the model
model.fit(predictors, target)
# Look at summary
model.summary()
# Calculate predictions: predictions
predictions = model.predict(pred_data)
```

### 7.5.2 Classification

- `loss = categorical_crossentropy`
- `metric = accuracy`
- `activation_function = softmax`
- output layer with stuff equal to number of categorical groups

```
import keras
from keras.layers import Dense
from keras.models import Sequential
# Specify the model: Two hidden layers
n_cols = predictors.shape[1]
model = Sequential()
## Input
model.add(Dense(50, activation='relu', input_shape = (n_cols,)))
# Hidden
model.add(Dense(32, activation='relu'))
# Output
model.add(Dense(2, activation = 'softmax'))
# Compile the model
```



```

assumed that observations within each group were correlated.

# Include the AverageAgeofMother as a correlated random-effect slope parameter
ageMotherModelRandomUncorrelated <- lmer( BirthRate ~ AverageAgeofMother + (AverageAgeofMother || State)
      countyBirthsData)

# Extract the fixed-effect coefficients
fixef(out)

# Extract the random-effect coefficients
ranef(out)

# Estimate the confidence intervals
confint(out)

```

## 7.7 Infer Package

The infer library has the main verbs: `specify()` variables, `hypothesize()`, `generate()` data, `calculate()` metrics, `visualize()` results

```

set.seed(4747)
perm_slope <- twins %>%
  specify(Foster ~ Biological) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 10, type = "permute") %>%
  calculate(stat = "slope")

```

## 7.8 Networks

‘Classical SNA’ is mainly about descriptive network statistics: proximity, similarity, centrality, brokerage, positional measures, equivalence, etc.

Import necessary modules and draw graph.

```

import matplotlib.pyplot as plt
import networkx as nx
nx.draw(T_sub)
plt.show()

```

Get the nodes of interest.

```

noi = [n for n, d in T.nodes(data=True) if d['occupation'] == 'scientist']
### Use a list comprehension to get the edges of interest: eoi
eoi = [(u, v) for u, v, d in T.edges(data=True) if d['date'] < date(2010, 1, 1)]
### Set the weight of the edge
weight = 2
for u, v, d in T.edges(data=True):
    # Check if node 293 is involved
    if 293 in [u,v]:

```

```
# Set the weight to 1.1  
T[u][v]['weight'] = 1.1
```



# Chapter 8

## Code Snippets

### 8.1 Python

#### 8.1.1 General

- <https://chrisalbon.com/>
- pandas: `isin` | `crosstab`
- `pd.groupby` aggregate functions: `size`, `count`, `sum`, `mean`, `median`, `sd`, `var`, `min`, `max`, `prod`, `first`, `last`
- The only float that isn't equal to itself is `nan`.
- `%matplotlib` notebook: interactive plots
- Memory storage: use `pd.copy` to create a deep copy instead of a shallow copy
- IPy: `%who` or `%whos` shows defined variables & `%reset` resets them
- Append dict to df: use `ignore_index = True`. eg. `films_df = films_df.append(film_dict, ignore_index=1)`
- Debugging: `import pdb` | `pdb.set_trace()`,
- `dir()` gives a list of in scope variables
- `globals()` gives a dictionary of global variables
- `locals()` gives a dictionary of local variables
- <https://bugra.github.io/work/notes/2015-01-03/i-wish-i-knew-these-things-when-i-first-learned-python/>
- `np.dot = np.multiply` `%>% np.sum`
- flatten dict
  - `pd.io.json.json_normalize(stuff)` \*<https://stackoverflow.com/questions/6027558/flatten-nested-python-dictionaries-to-list-of-dicts> 41689055#41689055
  - <https://towardsdatascience.com/flattening-json-objects-in-python-f5343c794b10>

#### 8.1.2 Vanderplas Jupyter

- create helper functions and units tests as R/py for Rmd/ipynb
- `pytest`/`hypothesis` for testing

- use makefile to run cmd commands
- Write file with date: `df.to_csv('{ }_model.csv'.format(str(datetime.datetime.now()).split(' ')[0]))`
- Make package for workflow with data (**init.py**) and use this formation for function docs:

```
def fun(a):
    #Role
    #-----
    #Does something

    #Parameters
    #-----
    #Accepts something

    #Returns
    #-----
    #Returns something

    return(a)
```

### 8.1.3 Auto Feature Engineering

```
#https://stackoverflow.com/questions/50145953/how-to-apply-deep-feature-synthesis-to-a-single-table
import numpy as np
import pandas as pd
import featuretools as ft
df = pd.read_csv('iris.csv')
df = df.reset_index()
es = ft.EntitySet(id = "test") #.drop(columns = ['species'], axis = 1)
es = es.entity_from_dataframe(entity_id = 'd', dataframe = df, make_index=True, index='ind')
fm, features = ft.dfs(
    entityset = es,
    target_entity = 'd',
    agg_primitives = ['mean', 'max', 'percent_true', 'last'],
    trans_primitives = ['subtract', 'divide']
)
# produces 626 features
_, features2 = ft.dfs(
    entityset = es,
    target_entity = 'd',
    max_depth = 2
)
# produces no new features
```

### 8.1.4 SHAP Explainer

```
clf.fit(X_train, y_train)
explainer = shap.TreeExplainer(clf)
shap_values = explainer.shap_values(X_train)
shap.force_plot(explainer.expected_value, shap_values[0,:], X.iloc[0,:]) # one record explained
```



```
shap.summary_plot(shap_values, X_train, plot_type = "bar") # all records explained
shap.summary_plot(shap_values, X_train, plot_type = "bar") # shap mae for features across all records
```

### 8.1.5 DF To HTML Table

```
from flask import Flask
import pandas as pd
app = Flask(__name__)
pd.set_option('display.max_colwidth', -1)
@app.route("/")
def show_tables():
    data = pd.read_csv('squads.csv')
    return data.to_html(escape = False, index_names = False)
if __name__ == "__main__":
    app.run(debug=True)
```

### 8.1.6 Masking Array

You can mask your array using the `numpy.ma.array` function and subsequently apply any numpy operation:

```
import numpy as np
a = np.random.rand(10) # Generate random data.
a = np.where(a > 0.8, np.nan, a) # Set all data larger than 0.8 to NaN
a = np.ma.array(a, mask=np.isnan(a)) # Use a mask to mark the NaNs
a_norm = a / np.sum(a) # The sum function ignores the masked values.
a_norm2 = a / np.std(a) # The std function ignores the masked values.
a.data
```

### 8.1.7 Showing Data With Plotly

```
# plotly display data
from plotly.tools import FigureFactory as ff
import plotly
import pandas as pd
plotly.offline.init_notebook_mode()
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/2014_usa_states.csv')
table = ff.create_table(df.iloc[1:10, 1:5])
plotly.offline.plot(table, output_type = 'div', show_link = False, include_plotlyjs = False)
```

### 8.1.8 Hyperband Cross Validation

```
from civismlexth.hyperband import HyperbandSearchCV
#https://github.com/civisanalytics/civismlexth/blob/master/civismlexth/hyperband.py
```

### 8.1.9 Pandas Display Options

```
# Display floats with 2 decimal places
pd.options.display.float_format = '{:,.2f}'.format

# Expand display limits
pd.options.display.max_rows = 200
pd.options.display.max_columns = 100
```

### 8.1.10 Infix Operators

```
# Define new operator
class Infix:
    def __init__(self, function):
        self.function = function
    def __ror__(self, other):
        return Infix(lambda x, self = self, other = other: self.function(other, x))
    def __or__(self, other):
        return self.function(other)
    def __rlshift__(self, other):
        return Infix(lambda x, self=self, other=other: self.function(other, x))
    def __rshift__(self, other):
        return self.function(other)
    def __call__(self, value1, value2):
        return self.function(value1, value2)
pipe = Infix(lambda x,y: y(x))
4 |pipe| (lambda z: z**2)
```

### 8.1.11 Generators

```
# Generators Error
iterator = iter(iterable)
try:
    while True:
        item = next(iterator)
        do_stuff(item)
except StopIteration:
    pass
finally:
    del iterator
```

### 8.1.12 Rowwise Operations In Pandas

```
# Apply a function to every row in a pandas dataframe
rectangles = [
    { 'height': 40, 'width': 10 },
    { 'height': 20, 'width': 9 },
    { 'height': 3.4, 'width': 4 }
]
```

```

rectangles_df = pd.DataFrame(rectangles)
def calculate_area(row):
    return row['height'] * row['width']
rectangles_df.apply(calculate_area, axis=1)
# Apply to single column
df['SAL-RATE'].apply(money_to_float)

```

### 8.1.13 Additional Functions Args In Map

```

# Extra arguments to function
import functools
def add(x, y):
    return x + y

a = [1, 2, 3]
map(functools.partial(add, y=2), a)

```

### 8.1.14 Check CSV Encoding

```

with open("../input/kickstarter-projects/ks-projects-201801.csv", 'rb') as rawdata:
    result = chardet.detect(rawdata.read(10000))

```

### 8.1.15 MNIST Example

```

import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import svm
digits = datasets.load_digits()
#print(digits.data)
#print(digits.target)
print(digits.images[0])
clf = svm.SVC(gamma=.001, C=100)
x, y = digits.data[:-10], digits.target[:-10]
clf.fit(x,y)
print("Prediction: ", clf.predict(digits.data[-2].reshape(1, -1)))
plt.imshow(digits.images[-2], cmap=plt.cm.gray_r, interpolation="nearest")
plt.show()

```

## 8.2 R

### 8.2.1 General

- ctrl + shift + a: reformat code in rstudio
- library(rethinking) | data(UCBadmit) | d <- UCBadmit
- purrr::p/map\_dfr/c

- `e <- matrix(NA_real_, nrow = 1000, ncol = 8)`
- rowwise in R: `purrr::pmap`
- `str_split + unnest()`
- <https://stat.ethz.ch/R-manual/R-devel/library/base/html/Last.value.html>
- `na_if`: Replace certain values by na
- `purrr::map_df` - map over columns
- studio multiple cursors, snippets, addins
- `ctrl + shift + o`: toggle table of contents
- `ctrl shift enter`: run chunk
- `ctrl pageup/down`: navigate chunks
- `atl shift k`: shortcuts
- `ctrl + shift + R`: label sections
- `ctrl + alt + R`: run all chunks
- `df_print`: kable in yaml rmarkdown header
- `rm(list = ls())`: Remove all objects in the current workspace
- `cut()`: Transform a numeric variable into a categorical variable.
- `car`: recode (good for dummyming)
- Examine the objects in the workspace: `ls.str()`
- This is how you dynamically update a value in Rmarkdown: `x = backtick r x backtick`.
- `df[setdiff(names(df), "z")]`: remove column z
- `purrr::walk`
- `purrr::keep/discard` for `is.factor`
- `choroplethr | choroplethrMaps || d3r | d3treeR | data.tree`

### 8.2.2 Knitting A GHub Doc

output: `github_document`: default `html_notebook`: default

## 8.3 Various

```
tapply(Summary_Variable, Group_Variable, Function)

boxplot(frequency ~ attitude*gender,
col=c("white", "lightgray"), politeness)

#corrplot package in r
cor(Carseats[sapply(Carseats, function(x) !is.factor(x))])

install.packages(c("caret", "pROC"), dependencies = c("Depends", "Imports", "Suggests"))
```

```
library(knitr)
kable(pause_sum, caption = "Summary Statistics Table for Exit Slip Dataset",
      col.names = c("Mean", "Median", "SD", "Min", "Max", "N", "NAs"))

typeof(1) #"double"
typeof(1L) #"integer"

#Prints out the first 4 multiples of 3.
for(i in 1:4*3) {cat(i)}
```

### 8.3.1 Create Package

```
#https://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch/

library(devtools)
library(roxygen2)

# Navigate to parent folder
setwd('../Desktop/me')

# Name and create package
create("sansor")

# Add file with function
setwd('sansor/R')
file.create('test.R') # test <- function(){print('This is a test')}

# Back to package dir
setwd("..")

# Generate auto docs
document()

# Install package for use
setwd('..')
install()
```

### 8.3.2 Create Operator

```
library(stringr)
`%[]%` <- function(string, subset){
  return(str_sub(string, subset[1], subset[2]))
}
'test'%[]%c(1,3)
```

### 8.3.3 Useful Date Manipulation

```
year = as.integer(format(as.Date(TestEndTime, format="%Y-%m-%d"), "%Y"))
date <- seq(as.Date("2017/09/01"), as.Date("2017/09/25"), by="day")
```

### 8.3.4 R for Everything Talk

- `dir.exists` | `dir.create` | `download.file`
- `untar` | `unlink` | `file.info`
- `dir` | `file.rename` | `file.copy`
- `count.fields` | `system`
- dplyr five main actions: `select`, `filter`, `arrange`, `mutate`, `summarize`.

### 8.3.5 Multi Character To Numeric Casting

```
char_to_num_cols <- df %>% select(starts_with('MMR')) %>% names
df[, char_to_num_cols] <- sapply(df[, char_to_num_cols], as.numeric)
```

### 8.3.6 Grid Plot

```
gridExtra::grid.arrange(
  plots[[1]],plots[[2]],plots[[3]],
  plots[[4]],plots[[5]],plots[[6]], nrow=3)
```

### 8.3.7 Classification Target Comparison

```
chisq.test(table(cut(iris$Sepal.Length, 3), iris$Species))
```

### 8.3.8 Pairwise Scatterplot

```
GGally::ggpairs(data, mapping = aes(colour = category))
```

### 8.3.9 Deploy Shiny App

```
library(shinyapps)
library(shiny)

working_dir <- ''
setwd(working_dir)

shinyapps::setAccountInfo(name="<ACCOUNT>",
                           token="<TOKEN>",
                           secret="<SECRET>")

runApp()
deployApp()
```

### 8.3.10 Tidy Evaluation Example

```
# tidy evaluation
df %>%
  filter(
    !is.na (!!as.symbol("col1"))),
    (!!as.symbol("col2")) == 1
  ) %>%
  select(c(1, 2, 4))
```

### 8.3.11 Shut Up, R

```
#chunk options: echo = F, warning=F, error=F, message=F, results='hide'
```

### 8.3.12 Plotting

Stuff I did while testing Bundesliga xG data.

```
library(openxlsx)
library(tidyr)
library(ggplot2)
library(dplyr)
library(ggvis)
library(rCharts)

bund_xg <- read.xlsx("data/bundesliga_xg_2015_2016.xlsx")

bund_xg.long <- gather(bund_xg, Status, Team, Home:Away)
bund_xg.long$xG <- ifelse(bund_xg.long$Status == 'Home',
                          bund_xg.long$xG_H,
                          bund_xg.long$xG_A)
bund_xg.long.team <- group_by(bund_xg.long, Team)

xg.plot <- ggplot(data = bund_xg.long,
                  aes(x = GW, y = xG, group = Status, color = Status)) +
  geom_point() +
  geom_line() +
  labs(title = 'Bundesliga 15/16', x = 'Game Week', y = 'Expected Goals For') +
  facet_wrap(~Team)

ggsave(plot = xg.plot, filename = 'bund_xg_plot.png')

bund_xg.long[bund_xg.long$Team == 'Bayern',] %>%
  ggvis(~GW, ~xG) %>%
  layer_points() %>% layer_lines()
#add_tooltip(function(df) df$GW)

p = rPlot(xG ~ GW,
          data = bund_xg.long[bund_xg.long$Team == 'Bayern',],
          color = 'Status',
          type = 'line')
```

```
p$set(title = 'Bayern Munich (Bundesliga 15/16)')
p$guides(x = list(title = "Game Week"))
p$guides(y = list(title = "Expected Goals"))
```

### 8.3.13 Nested Tree

Stuff I did while testing tree viz for ODSC DS tools.

```
library(googleVis)
test.data <- read.csv('C:/Users/Gordon/Downloads/test.csv', header = T)

Org <- gvisOrgChart(test.data,
                    options=list(width=600, height=250,
                                size='large', allowCollapse=TRUE))

Org$html$footer <- NULL
Org$html$caption <- NULL
plot(Org)

test.data$Val <- seq(1, nrow(test.data), 1)
test.data$Fac <- seq(1, nrow(test.data), 1)

Tree <- gvisTreeMap(test.data,
                    "Node", "Parent",
                    #"Val", "Fac",
                    options=list(fontSize=16))
plot(Tree)
```

### 8.3.14 Setup Project

```
library(knitr)

#Create top level folder
dir.create('my-data-project')
setwd('./my-data-project/')

#Create child folders
dir.create('code')
dir.create('visualizations')
dir.create('reports')
dir.create('data')
dir.create('deploy_maintain')

#Create trackers
file.create('README.md')
file.create('CHANGELOG.md')

#Create grandchildren folders for data
setwd('./data')
dir.create('raw')
dir.create('refined')
```



```

#Create grandchildren folders for visualizations
setwd('../visualizations')
dir.create('exploratory')
dir.create('communication')

#Initialize git repo
git2r::init('.')

# CHANGELOG.md Example

## Logs

## Excluded processes should be assumed to be the same as version 0.1.

## Current Best Model:
## Current Best Score / Metric:

## 0.1

## 1_import_tidy.Rmd

## 2_transform_visualize.Rmd

## 3_modeling.ipynb

# Layout

## libs, functions, imports, preprocessing, feature engineering, feature selection, model building

```

### 8.3.15 MAP Model

```

library(tidyverse)

model <- function(df) {
  lm(mpg ~ ., data = df)
}

df <- mtcars
df %>% group_by(cyl) %>% split(.$cyl) %>% purrr::map(~ lm(mpg ~ ., data = .))
df %>% group_by(cyl) %>% nest() %>% mutate(model = map(data, model))

chickweight %>% group_by(Diet) %>% nest() %>%
mutate(mod = data %>% map(~ lm(weight ~ Time, data=)))

```

### 8.3.16 DT Options

```

datatable(
  df_version_check,
  options = list(filter = 'top',
                 searching = T,

```

```

        pageLength = 10, lengthChange = FALSE, ordering = FALSE,
        scrollCollapse = T,
        scrollY = '1500',
        paging = T
      )
    )
  )
)

```

### 8.3.17 Plotly MAP

```

ggplotly(
  ggplot(df_master2 %>% filter(skl_group==x),
    aes(SkillId, correct_rate, label = Master, color = SkillId)) +
  geom_jitter(width = .2),
  tooltip = c('y', 'label')
)

```

### 8.3.18 Pipe Anonymous Function

```

data %>%
  (function(input.object) {
    complex
    logic
    goes
    here
    output.object
  })

```

### 8.3.19 Binarize Predictions

```

clf_lr_final <- clf_lr$finalModel
model_tidy <- broom::tidy(clf_lr_final)
df2$pass_predicted <- as.factor(
  ifelse(clf_lr_final$fitted.values >= .5, 'pass', 'fail')
)
metrics(df2, pass, pass_predicted)

var_imp <- varImp(clf_lr_final) %>% mutate(cols = row.names(.)) %>% arrange(-Overall) %>% select(cols, )
var_imp

```

### 8.3.20 Make Formula

```

make_formula <- function(target, additions, subtractions = NULL, powers = NULL){

  add_sub_sep = ifelse(is.null(subtractions), '', '-')
  add <- paste0(additions, collapse = ' + ')
  subtract <- paste0(subtractions, collapse = ' - ')
  add_subtract <- paste(add, subtract, sep = add_sub_sep)
}

```

```

return(as.formula(paste(target, add_subtract, sep = '~')))
}

```

### 8.3.21 Shiny Reactive

```

df_labeled = eventReactive({c(input$good, input$bad)}, {
  df %>% filter(value == input$value)
})

```

### 8.3.22 Custom Theme

```

theme(
  axis.text.y = element_text(angle = 65, vjust = 0.6),
  axis.title.x = element_blank(),
  panel.background = element_blank()
)

```

### 8.3.23 Create Factor

```

mons = factor(
  c("August", "September", "October", "November",
    "December", "January", "February", "March",
    "April", "May", "June", "July"),
  levels = c("August", "September", "October", "November",
    "December", "January", "February", "March",
    "April", "May", "June", "July"),
  ordered = TRUE)

```

### 8.3.24 Extract Date

```

StartTime %>% as.POSIXct() %>% strftime(format="%Y-%m-%d")
YearMonth = Day %>% strftime(format="%Y-%m")

```

### 8.3.25 Geom\_Path

```

ggplot(ilo_data) +
  geom_path(
    aes(x = working_hours, y = country),
    arrow = arrow(length = unit(1.5, "mm"), type = "closed")
  )

```

### 8.3.26 Theme Template

```
theme_ilo <- function() {
  theme_minimal() +
  theme(
    text = element_text(family = "Bookman", color = "gray25"),
    plot.subtitle = element_text(size = 12),
    plot.caption = element_text(color = "gray30"),
    plot.background = element_rect(fill = "gray95"),
    plot.margin = unit(c(5, 10, 5, 10), units = "mm")
  )
}
```

### 8.3.27 Heatmap

```
library(corrplot)
df_filtd <- df %>% select_if(is.numeric) %>%
cor() %>% corrplot(method = "circle", is.corr = FALSE)
```

### 8.3.28 Calculate Accuracy

```
calc_accuracy <- function(labels, preds){

  results <- table(labels, preds) %>%
    data.frame %>%
    mutate(correct = ifelse(labels==preds, 1, 0)) %>%
    group_by(correct) %>%
    summarise(freq = sum(Freq)) %>%
    ungroup() %>%
    mutate(prop = freq/sum(freq)) %>%
    filter(correct == 1)

  return(results)
}
```

### 8.3.29 Replace NA

```
df %>% replace_na(list(x = 0, y = "unknown"))
df2 %>% replace(., is.na(.), "")
```

### 8.3.30 Filter By Row

```
df %>%
  filter(
    !grepl('Schedules', issues)
  )
```

### 8.3.31 DB Functions

```
# DB Functions

data <- dbGetQuery(con, 'select * from iris')
dbListTables(con) %>% sort
dbExistsTable(con, "iris")
dbRemoveTable(con, "iris")
```

### 8.3.32 Knitr

```
knitr::include_graphics(rep("./assets/ml_trading/portfolio.png", 3))
```

## 8.4 Bash

script to do joining of dummied csvs

```
paste -d',' file1.csv file2.csv file3.csv > file_final.csv
```



## Chapter 9

# Time Series

### 9.1 Notes

A stationary time series is one whose properties do not depend on the time at which the series is observed.

### 9.2 Packages

<https://robjhyndman.com/seminars/user-fable/>    <https://github.com/robjhyndman/forecast>    [https://facebook.github.io/prophet/docs/quick\\_start.html](https://facebook.github.io/prophet/docs/quick_start.html)    <https://keras.rstudio.com/articles/examples/index.html>  
<http://www.business-science.io/code-tools/2017/05/02/timekit-0-2-0.html>    <https://www.rdocumentation.org/packages/bsts/versions/0.8.0>

### 9.3 Data Conversion

<https://robjhyndman.com/hyndsight/seasonal-periods/>

```
dt = as.Date(StockData$Date, format="%m/%d/%Y") > Stockdataz = zoo(x=cbind(StockData$Volume, StockData$Adj),
ts: frequency = 5 means that data is at daily level business days
ts(completed_example_data$Wind, start = c(1973, 5, 1), frequency = 365.25)

xts(df[, -1], order.by = as.Date(df[, 1], "%m/%d/%Y"))
```

### 9.4 Models

Exponential smoothing and ARIMA models are the two most widely used approaches to time series forecasting, and provide complementary approaches to the problem. While exponential smoothing models are based on a description of the trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data.

Benchmark models: Mean, naive, seasonal naive

0. Mean

1. Naive: Set all forecasts to be the value of the last observation. This method works remarkably well for many economic and financial time series. Because a naive forecast is optimal when data follow a random walk, these are also called random walk forecasts. `naive()`
2. Seasonal Naive: Assumes magnitude of the seasonal pattern is constant. `snaive()`

#### Linear

10. regression: `tslm()`: trend and season are not objects in the R workspace; they are created automatically by `tslm()` when specified in this way: `tslm(beer2 ~ trend + season)`. It is not recommended that quadratic or higher order trends be used in forecasting. When they are extrapolated, the resulting forecasts are often unrealistic.
11. harmonic regression: An alternative to using seasonal dummy variables, especially for long seasonal periods, is to use Fourier terms.  $i$ th Fourier terms, we often need fewer predictors than with dummy variables, especially when  $m$  is large. This makes them useful for weekly data, for example, where  $m = 52$ . For short seasonal periods (e.g., quarterly data), there is little advantage in using Fourier terms over seasonal dummy variables. `fourier()` - `tslm(beer2 ~ trend + fourier(beer2, K=2))`
12. Dynamic Regression

The R function `Arima()` will fit a regression model with ARIMA errors if the argument `xreg` is used. The order argument specifies the order of the ARIMA error model. If differencing is specified, then the differencing is applied to all variables in the regression model before the model is estimated.

```
Arima(y, xreg=x, order=c(1,1,0)) auto.arima(uschange[, "Consumption"], xreg=uschange[, "Income"])
```

There are two different ways of modelling a linear trend: deterministic and stochastic.

```
deterministic: fit1 <- auto.arima(austa, d=0, xreg=trend)
```

```
stochastic: fit2 <- auto.arima(austa, d=1)
```

There is an implicit assumption with deterministic trends that the slope of the trend is not going to change over time. On the other hand, stochastic trends can change, and the estimated growth is only assumed to be the average growth over the historical period, not necessarily the rate of growth that will be observed into the future. Consequently, it is safer to forecast with stochastic trends, especially for longer forecast horizons, as the prediction intervals allow for greater uncertainty in future growth.

When there are long seasonal periods, a dynamic regression with Fourier terms is often better than other models we have considered in this book.

Seasonal differencing of high order does not make a lot of sense — for daily data it involves comparing what happened today with what happened exactly a year ago and there is no constraint that the seasonal pattern is smooth.

So for such time series, we prefer a harmonic regression approach where the seasonal pattern is modelled using Fourier terms with short-term time series dynamics handled by an ARMA error.

```
fit <- auto.arima(caffe04, xreg = fourier(caffe04, K = i), seasonal = FALSE, lambda = 0)
```

#### ETS (Error-Trend-Seasonality) Models

This label can also be thought of as Exponential Smoothing. The possibilities for each component are: Error = {a,m}, Trend = {n,a,a\_d}, and Seasonal = {n,a,m}. Therefore, for each component in the ETS system, we can assign None, Multiplicative, or Additive (or N, M, A) for each of the three components in our time series.

Three of the combinations of (Error, Trend, Seasonal) can lead to numerical difficulties. Specifically, the models that can cause such instabilities are ETS(A,N,M), ETS(A,A,M), and ETS(A,A d ,M), due to division by values potentially close to zero in the state equations. We normally do not consider these particular combinations when selecting a model.



Models with multiplicative errors are useful when the data are strictly positive, but are not numerically stable when the data contain zeros or negative values. Therefore, multiplicative error models will not be considered if the time series is not strictly positive. In that case, only the six fully additive models will be applied.

The `ets()` function does not produce forecasts. Rather, it estimates the model parameters and returns information about the fitted model. By default it uses the AICc to select an appropriate model

ETS point forecasts are equal to the medians of the forecast distributions. For models with only additive components, the forecast distributions are normal, so the medians and means are equal. For ETS models with multiplicative errors, or with multiplicative seasonality, the point forecasts will not be equal to the means of the forecast distributions.

1. Simple Exponential Smoothing Method: Time series does not have a trend line and does not have seasonality component. We would use a Simple Exponential Smoothing model.

ETS(A,N,N): simple exponential smoothing with additive errors | ETS(M,N,N): simple exponential smoothing with multiplicative errors

2. Holt's Linear Trend Method: Damped Holt's method is best whether you compare MAE or MSE values. So we will proceed with using the damped Holt's method and apply it to the whole data set to get forecasts for future years. Forecasts account for trend & continue at the same trend indefinitely into the future: `holt(damped = T)`. A damped trend makes it level off after time: `holt(damped = F)` ( $\phi = 1$  is Holt's method).

`holt()` | ETS(A,A,N): Holt's linear method with additive errors | ETS(M,A,N): Holt's linear method with multiplicative errors | (a,a) Additive Holt-Winters' method | (a,m) Multiplicative Holt-Winters' method | (a\_d,m) Holt-Winters' damped method | (a,n) Holt's linear method | (a\_d,n) Additive damped trend method

8. Holt-Winters Seasonal Method: `hw(aust,seasonal="additive")`. Accounts for trend and seasonality. In the additive version the seasonality averages to 0 but this changes to 1 in the multiplicative version

ARIMA(p,d,q): non-seasonal: White noise: ARIMA(0,0,0) Random walk: ARIMA(0,1,0) with no constant Random walk with drift: ARIMA(0,1,0) with a constant Autoregression: ARIMA(p,0,0) Moving average: ARIMA(0,0,q)

Seasonal ARIMA(p,d,q)(P,D,Q)\_m

`auto.arima()` does an auto fit.

When fitting an ARIMA model to a set of (non-seasonal) time series data, the following procedure provides a useful general approach.

Plot the data and identify any unusual observations.

If necessary, transform the data (using a Box-Cox transformation) to stabilize the variance.

If the data are non-stationary, take first differences of the data until the data are stationary.

Examine the ACF/PACF: Is an ARIMA(p,d,0) or ARIMA(0,d,q) model appropriate?

Try your chosen model(s), and use the AICc to search for a better model.

Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model.

Once the residuals look like white noise, calculate forecasts.

`auto.arima` only does steps 3-5.

`arima()`

`ggAcf()` | `ggPacf()`

If the data are from an  $\text{ARIMA}(p,d,0)$  or  $\text{ARIMA}(0,d,q)$  model, then the ACF and PACF plots can be helpful in determining the value of  $p/q$ . If  $p$  and  $q$  are both positive, then the plots do not help in finding suitable values of  $p$  and  $q$ .

The data may follow an  $\text{ARIMA}(p,d,0)$  model if the ACF and PACF plots of the differenced data show the following patterns:

the ACF is exponentially decaying or sinusoidal; there is a significant spike at lag  $p$  in the PACF, but none beyond lag  $p$ .

The data may follow an  $\text{ARIMA}(0,d,q)$  model if the ACF and PACF plots of the differenced data show the following patterns:

the PACF is exponentially decaying or sinusoidal; there is a significant spike at lag  $q$  in the ACF, but none beyond lag  $q$ .

seasonal arima:  $\text{arima}(p,d,q) (P,Q,D)_m$ ,  $m = \text{num of observations per year}$

The seasonal part of an AR or MA model will be seen in the seasonal lags of the PACF and ACF. For example, an  $\text{ARIMA}(0,0,0)(0,0,1)_{12}$  model will show:

a spike at lag 12 in the ACF but no other significant spikes; exponential decay in the seasonal lags of the PACF (i.e., at lags 12, 24, 36, ...).

`Arima(euroretail, order=c(0,1,3), seasonal=c(0,1,1))`

`auto.arima(euroretail, stepwise=FALSE, approximation=FALSE)`

comparing information criteria is only valid for ARIMA models of the same orders of differencing.

Other

1. `ma`
2. `wma`
3. X11 method: `seasonal::seas(x11 = ")`. from this output `seasonal()` will extract the seasonal component, `trendcycle()` will extract the trend-cycle component, `remainder()` will extract the remainder component, and `seasadj()` will compute the seasonally adjusted time series.
4. SEATS: `seasonal::seas()`
5. STL: best so far but only uses additive decomposition. `stl(t.window=13, s.window="periodic", robust=TRUE)`. automate picking of first 2 params with `mstl()`. A short-cut approach is to use the `stlf()` function. The following code will decompose the time series using STL, forecast the seasonally adjusted series, and return reseasonalized the forecasts: `stlf(elecequip, method='naive')`

`msts | mstsl | tbats | bld.mbb.bootstrap`

Auto Model:

`forecast()`: You can use this directly if you have no idea which model to use, or use it to produce forecasts after fitting a model.

Hierarchical:

`gts() | hts() | aggts()`

- Feature Engineering

<https://github.com/robjhyndman/tsfeatures>

There are several useful predictors that occur frequently when using regression for time series data:

- 1) A trend variable can be specified in the `tslm()` function using the trend predictor.

- 2) dummy variables for time like day of week. The `tslm()` function will automatically handle this situation if you specify the predictor season.
- 3) interventions, eg. competitor activity
- 4) trading days: `bizdays()`
- 5) distributed lags
- 6) `easter()`

Use lagged values of predictors

A log-log functional form is specified as  $\log(y) = \beta_0 + \beta_1 \log(x) + \epsilon$

scaling by `monthday()` by population or by inflation for money

Logarithms are useful because they are interpretable: changes in a log value are relative (or percentage) changes on the original scale. So if log base 10 is used, then an increase of 1 on the log scale corresponds to a multiplication of 10 on the original scale. Another useful feature of log transformations is that they constrain the forecasts to stay positive on the original scale. A log transformation can address heteroskedasticity.

`boxCox()/boxCox.lambda()`, power transforms,

bias adjustments: mean of forecast distribution instead of median for back-transformed forecast. Bias adjustment is not done by default in the forecast package. If you want your forecasts to be means rather than medians, use the argument `biasadj=TRUE` when you select your Box-Cox transformation parameter.

- Model Evaluation

train-test split: `window()` / `subset()`

`accuracy()`

`tsCV()`

A great advantage of the ETS statistical framework is that information criteria can be used for model selection - AIC, AIC<sub>c</sub>, and BIC

leave-one-out cross-validation statistic : `CV(fit.consMR)` for model selection

we recommend that one of the AIC<sub>c</sub>, AIC, or CV statistics be used, each of which has forecasting as their objective.

When comparing forecast methods applied to a single time series, or to several time series with the same units, the MAE is popular as it is easy to both understand and compute. A forecast method that minimizes the MAE will lead to forecasts of the median, while minimizing the RMSE will lead to forecasts of the mean. Consequently, the RMSE is also widely used, despite being more difficult to interpret. Percentage errors have the advantage of being unit-free, and so are frequently used to compare forecast performances between data sets. The most commonly used measure is Mean Absolute Percentage Error (MAPE) or symmetric MAPE. forecast errors are different from residuals in two ways. First, residuals are calculated on the training set while forecast errors are calculated on the test set. Second, residuals are based on one-step forecasts while forecast errors can involve multi-step forecasts.

A good forecasting method will yield residuals with the following properties:

- 1) The residuals are uncorrelated. If there are correlations between residuals, then there is information left in the residuals which should be used in computing forecasts.
- 2) The residuals have zero mean. If the residuals have a mean other than zero, then the forecasts are biased. (Adjusting for bias is easy: if the residuals have mean  $m$  then add  $m$  to all forecasts and the bias problem is solved.)
- 3) It is useful (but not necessary) for the residuals to also have the following two properties: the residuals have constant variance and are normally distributed.

use the Box-Ljung test to test for autocorrelations: `h_0`: no autocorrelation, `h_a`: autocorrelation.

`checkresiduals()` which will produce a time plot, ACF plot and histogram of the residuals (with an overlaid normal distribution for comparison), and do a Ljung-Box test with the correct degrees of freedom.

#### -Notes

Fourier terms in `lm` good for intraday data.

Methods with multiplicative trend produce poor forecasts.

For stock market prices and indexes, the best forecasting method is often the naïve method. Use also dynamic regression.

Create time series with `ts(mydata[-1], start = c(1981, 1), frequency = 4)`

Time series that show no auto-correlation are called white noise. For white noise series, we expect each auto-correlation to be close to zero. For a white noise series, we expect 95% of the spikes in the ACF to lie within  $\pm 2 * \sqrt{T}$  where  $T$  is the length of the time series.

The possible time series (TS) scenarios can be recognized by asking the following questions: 1) TS has a trend? If yes, is the trend increasing linearly or exponentially? 2) TS has seasonality? If yes, do the seasonal components increase in magnitude over time?

Additive model for linear trend and multiplicative model for exponential trend. Additive for trend and Multiplicative and Additive for seasonal components. For trends that are exponential, we would need to use a multiplicative model. For increasing seasonality components, we would need to use a multiplicative model as well.

Therefore we can generalize all of these models using a naming system for ETS:

Error is the error line we saw in the time series decomposition part earlier in the course. If the error is increasing similar to an increasing seasonal components, we would need to consider a multiplicative design for the exponential model.

A time series model that has a constant error, linear trend, and increasing seasonal components means we would need to use an ETS model of `ETS(N,A,M)`

A time series model that has increasing error, exponential trend, and no seasonality means we would need to use an ETS model of:

non-stationary: This plot shows an upward trend and seasonality.

stationary: This plot revolves around a constant mean of 0 and shows contained variance.

#### -Metrics

Percentage errors, like MAPE, are useful because they are scale independent, so they can be used to compare forecasts between different data series, unlike scale dependent errors. The disadvantage is that it cannot be used if the series has zero values.

Mean Absolute Percentage Error (MAPE) is also often useful for purposes of reporting, because it is expressed in generic percentage terms it will make sense even to someone who has no idea what constitutes a “big” error in terms of dollars spent or widgets sold.

Mean Absolute Scaled Error (MASE) is another relative measure of error that is applicable only to time series data. It is defined as the mean absolute error of the model divided by the the mean absolute value of the first difference of the series. Thus, it measures the relative reduction in error compared to a naïve model. Ideally its value will be significantly less than 1 but is relative to comparison across other models for the same series. Since this error measurement is relative and can be applied across models, it is accepted as one of the best metrics for error measurement.

`AIC_c` for model selection.

**9.4.1 Intro****9.4.2 Assumptions****9.4.3 Characteristics****9.4.4 Evaluation****9.4.5 Pros/Cons****Pros****Cons**

- Seasonal Decomposition; Use additive model when the magnitude of the seasonal fluctuations surrounding the general trend doesn't vary over time. Use the multiplicative model when the magnitude of the seasonal fluctuations surrounding the general trend appear to change in a proportional manner over time. Go from additive to multiplicative with a log transformation.
- White noise: Describes the assumption that each element in the time series is a random draw from  $N(0, \text{constant variance})$ . Also called a stationary series. Time series with trends or seasonality are not stationary.

**ARIMA(p,d,q) (Auto-Regressive Integrated Moving Average) Models**

- AR(p): auto-regressive component for lags on the stationary series. The AR models say that the value of a variable at a specific time is related to the value of the variable at previous times.
- I(d): Integrated component for a series that needs to be differenced.
- MA(q): Moving average component for lag of the forecast errors. In a moving average model of order q, each value in a time series is predicted from the linear combination of the previous q errors. The value of a variable at a specific time is related to the residuals of prediction at previous times.
- In an auto-regressive model of order p, each value in a time series is predicted from a linear combination of the previous p values.
- Procedure: Make stationary if necessary by differencing. Determine possible values of p and q. Assess model fit and try other values of p and q to overfit. Make forecasts with the final model.
- Augmented Dicky-Fuller Test: This tests whether or not a time series is stationary.  $H_0$ : series is not stationary,  $H_a$ : the series is stationary.
- Determine p and q: Look at autocorrelation (AC) and partial correlation (PAC) functions. AC measures the way observations relate to each other. PAC measure the way observations relate to each other after accounting for all other intervening observations. Plot of the autocorrelation function (ACF) displays correlation of the series with itself at different lags. A plot of the PAC displays the amount of autocorrelation not explained by lower order correlations. Spikes in the PACF will choose for AR(p). Spikes in the ACF will choose MA(q).

**Assess Model Fit**

- Appropriate model should resemble white noise. Check scatterplot of residuals vs fit to check constant variance and qqplot to check normality. Autocorrelations should be zero to check for violation of independent errors.
- Box-Ljung Test: Check if all autocorrelations are zero, i.e the series is of white noise.  $H_0$ : autocorrelations are all 0.  $H_a$ : At least one is nonzero.
- Overfit model with extra AR/MA terms and compare using AIC/BIC.

- Interpretation: AR coefficient closer to 1 means series returns to mean slowly, vice versa for closer to 0.
- MA(1) coefficient indicates how much the shock of the previous time period is retained in the current time period. MA(2) refers to the previous two time periods.

# Chapter 10

## Stats

### 10.0.1 General

Another common measure of effect size is  $d$ , sometimes known as Cohen's  $d$  (as you might have guessed by now, Cohen was quite influential in the field of effect sizes). This can be used when comparing two means, as when you might do a  $t$ -test, and is simply the difference in the two groups' means divided by the average of their standard deviations\*. This means that if we see a  $d$  of 1, we know that the two groups' means differ by one standard deviation; a  $d$  of .5 tells us that the two groups' means differ by half a standard deviation; and so on. Cohen suggested that  $d=0.2$  be considered a 'small' effect size, 0.5 represents a 'medium' effect size and 0.8 a 'large' effect size. This means that if two groups' means don't differ by 0.2 standard deviations or more, the difference is trivial, even if it is statistically significant.

Beta Distribution: The larger the alpha parameter is the closer the resulting distribution is to 1.0 and the larger the beta the closer it is to 0.

Fisher's Exact Test: Test for proportions

Bayes Rule:  $\text{Posterior} = \text{Likelihood} * \text{Prior} / \text{Marginal Likelihood}$

<https://www.rdocumentation.org/packages/effsize/versions/0.7.1/topics/cohen.d>

Likelihood  $P(\text{Data}|\theta)$ : Probability of the data could be generated by a model with the given parameter(s)

Prior  $P(\theta)$ : Probability of parameter value

Posterior  $P(\theta|D)$ : Credibility of the parameter value with the data taken into account.

Marginal Likelihood (Evidence): Probability of evidence

Variance: Spread of the data | Distance from the mean

Standard Deviation: Square root of the variance | Same units as the data

Mean: Average/Expected Value

Covariance: Measure of how two random values change together. Negative - variables show opposite behavior ( As A increases B decreases). Positive - variables show similar behavior (As A increases B increases)

Pearson Correlation: Normalized version of covariance

Median: Number at the halfway point of the dataset, found through sorting data into ascending order and selecting middle point

Mode: Most frequent value in dataset

Note about Populations and Samples: Populations have parameters. Samples have statistics.

Distributions:

Normal/Gaussian: Parameters are mean and standard deviation. Bell shaped. z-score: indicates how many standard deviations an element is from the mean

Student t: used to estimate the mean of a normally distributed population where the sample size is small and the standard deviation is unknown

A t test (t.test in R) is a special case of an ANOVA and a paired t test (parameter paired = T) is a special case of a repeated measures ANOVA.

Sample size: A good formula is: if your sample is picked uniformly at random and is of size at least:

$$\frac{-\log(\frac{d}{2})}{2e^2}$$

then with probability at least  $1-d$  your sample measured proportion  $q$  is no further away than  $e$  from the unknown true population proportion  $p$  (what you are trying to estimate). Need to estimate with 99% certainty the unknown true population rate to  $\pm 10\%$  precision? That is  $d=0.01$ ,  $e=0.1$  and a sample of size 265 should be enough. A stricter precision of  $\pm 0.05$  causes the estimated sample size to increase to 1060, but increasing the certainty requirement to 99.5% only increases the estimated sample size to 300.

- Type I Error: Saying there is an effect when there isn't one. Type II error: concluding there is no effect when, in fact, there is one.
- The MWW RankSum test is a useful test to determine if two distributions are significantly different or not. Unlike the t-test, the RankSum test does not assume that the data are normally distributed, potentially providing a more accurate assessment of the data sets.
- Probably the most useful types of statistics for skewed probability distributions are quantiles.
- Hypothesis testing requires an inferential-statistical approach. Crucial are meaningful distributions of test statistics, on which p-values for hypothesis tests can be based. It is not trivial to construct such "meaningful distributions" for complete network data!
- Random sampling & random assignment: generalizable & causal
- Get 95% ci around p-value
- There is no one rule about when a number is not accurate enough to use, but as a rule of thumb, you should be cautious about using any number with a MOE (Margin-of-error) over 10%.
- Probability is calibrated if it is empirically correct, i.e. the empirical probability matches the theoretical one.
- Effect size
- `jarque.bera.test`: Tests the null of normality for the series using the Jarque-Bera test statistics
- `box.test`: Compute the Box-Pierce or Ljung-Box test statistics for examining the null of independence in a given series
- `shapiro.test`: Test for normality
- Sum of normally distributed random variables: sum of means & sum of variances
- Zero correlation doesn't mean independent variables. Covariance only determines linear relationships.
- test heteroscedaticity: `breusch-pagan` or `ncv` test
- Test Normality: Shapiro-Wilk test is a test of normality
- Bypass inferential stats if features at least 1/10 of data points.
- The probability (p-value) of observing results at least as extreme as what is present in your data sample. P-value less than .05 retain `h_0` else reject `h_0`.



- One Sample T-test: To examine the average difference between a sample and the known value of the population mean. Assumes the population from which the sample is drawn is normally distributed and the sample observations are randomly drawn and independent.
- Two Sample T-test: To examine the average difference between two samples drawn from two different populations. Assumes the populations from which the samples are drawn are normally dist, the standard deviations of the two populations are equal, and sample observations are randomly drawn and independent.
- One-Way ANOVA: To assess the equality of means of two or more groups. Assumes the populations from which the samples are drawn are normally dist, the standard deviations of the populations are equal, and sample observations are randomly drawn and independent.
- Chi-Squared Test of Independence: To test whether two categorical variables are independent. Assumes the sample observations are randomly drawn and independent.
- F-Test: To assess whether the variances of two different populations are equal. Assumes the population from which the sample is drawn is normally distributed and the sample observations are randomly drawn and independent.
- Barlett Test: F-Test for more than two populations.
- If we take a larger and larger sample from a population, its distribution will tend to become normal no matter what it is initially. It won't. The Central Limit Theorem, the misreading of which is the cause of this mistake, refers to the distribution of standardized sums of random variables as their number grow, not to the distribution of a collection of random variables.
- Frequentists: Probability is a measure of the the frequency of repeated events. The parameters are fixed (but unknown) and data are random.
- Bayesians: Probability is a measure of the degree of certainty about values, so the interpretation is that rameters are random and data are fixed.
- Independent rvs are uncorrelated. That is  $\text{Cor}(X,Y)=0$
- Don't sum sd's, sum variances.
- One Sample T-Test? To examine the average difference between a sample and the known value of the population mean. Assumptions: The population from which the sample is drawn is normally distributed. Sample observations are randomly drawn and independent.
- When do we use the Two Sample T-Test? To examine the average difference between two samples drawn from two different populations. Assumptions: The populations from which the samples are drawn are normally dist. The standard deviations of the two populations are equal. Sample observations are randomly drawn and independent.
- When do we use the F-Test? To assess whether the variances of two different populations are equal. Assumptions: The populations from which the samples are drawn are normally dist. Sample observations are randomly drawn and independent.
- When do we use One-Way ANOVA? To assess the equality of means of two or more groups. NB: When there are exactly two groups, this is equivalent to a Two Sample T-Test. Assumptions: The populations from which the samples are drawn are normally dist. The standard deviations of the populations are equal. Sample observations are randomly drawn and independent.
- When do we use the chisq2 Test of Independence? To test whether two categorical variables are independent. Assumptions: Sample observations are randomly drawn and independent.
- To perform the one-way ANOVA, we can use the `f_oneway()` function of the SciPy package.
- The degrees of freedom is the number of data rows minus the number of coefficients fit.

- Hypothesis tests aim to describe the plausibility of a parameter taking on a specific value. Confidence intervals aim to describe a range of plausible values a parameter can take on. If the value of the parameter specified by  $H_0$  is contained within the 95% confidence interval, then  $H_0$  cannot be rejected at the 0.05 p-value threshold. If the value of the parameter specified by  $H_0$  is not contained within the 95% confidence interval, then  $H_0$  can be rejected at the 0.05 p-value threshold.

## Introduction to Data

There is no random sampling since the subjects of the study were volunteers, so the results cannot be generalized to all people. However, due to random assignment, we are able to infer a causal link between the belief information is stored and the ability to recall that same information.

Explanatory variables are conditions you can impose on the experimental units, while blocking variables are characteristics that the experimental units come with that you would like to control for.

In random sampling, you use stratifying to control for a variable. In random assignment, you use blocking to achieve the same goal.

Control: compare treatment of interest to a control group

Randomize: randomly assign subjects to treatments

Replicate: collect a sufficiently large sample within a study, or replicate the entire study

Block: account for the potential effect of confounding variables

Group subjects into blocks based on these variations, and randomize within each block to treatment groups

Random Sampling + Random Assignment: Causal + generalizable

No Random Sampling + Random Assignment: Causal + not generalizable

Random Sampling + No Random Assignment: Not causal + generalizable

No Random Sampling + No Random Assignment: Not causal + not generalizable

### 10.0.2 Statistical Tests

The appropriate sample size depends on many things, chiefly the complexity of the analysis and the expected effect size. The “30” rule comes from one of the simplest cases: Whether to use a z-test or t-test for comparing two means.

- 1) for large  $n$  (sample size), many distributions can be approximated to normal. [ courtesy : Central Limit Theorems ]
- 2) The approximation becomes better with larger  $n$ , given other things remaining the same.

However if you are wondering why 30 ? why not 25 or 40 or something else ? Note that actual answer will depend:

- 1) how much error are you going to tolerate
- 2) size of the effect
- 3) exact distribution family (like t vs chi-square vs gamma)

The power of any test of statistical significance is defined as the probability that it will reject a false null hypothesis. Statistical power is inversely related to beta or the probability of making a Type II error. In short,  $\text{power} = 1 - \beta$ . In plain English, statistical power is the likelihood that a study will detect an effect when there is an effect there to be detected. If statistical power is high, the probability of making a Type II error, or concluding there is no effect when, in fact, there is one, goes down. Statistical power is affected

chiefly by the size of the effect and the size of the sample used to detect it. Bigger effects are easier to detect than smaller effects, while large samples offer greater test sensitivity than small samples.

The following four quantities have an intimate relationship:

- sample size
- effect size
- significance level =  $P(\text{Type I error})$  = probability of finding an effect that is not there
- power =  $1 - P(\text{Type II error})$  = probability of finding an effect that is there

Given any three, we can determine the fourth.

#### Other

- The power of any test of statistical significance is defined as the probability that it will reject a false null hypothesis. Statistical power is inversely related to beta or the probability of making a Type II error. In short, power =  $1 - \beta$ .
- For quantitative explanatory variables, effect sizes always have the unit of the response variable divided by the unit of the explanatory variable.
- Power Analysis Calculator

### 10.0.3 Stats For Hackers

- Methods: Simluation, Boostapping (with replacement, not good with ranks), Shuffling (Works when assumption is that the data are the same, representative samples, non-longitudinal data), & Cross Validation.
- Assumes iid.

### 10.0.4 Think Stats

- Is it possible that the apparent effect is due to selection bias or some other error in the experimental setup? If so, then we might conclude that the effect is an artifact; that is, something we created (by accident) rather than found.
- The sampling distribution is the distribution of the samples mean.
- The CDF is the function that maps values to their percentile rank in a distribution.
- I'll start with the exponential distribution because it is easy to work with. In the real world, exponential distributions come up when we look at a series of events and measure the times between events, which are called interarrival times. If the events are equally likely to occur at any time, the distribution of inter-arrival times tends to look like an exponential distribution.
- CLT: More specifically, if the distribution of the values has mean and standard deviation and , the distribution of the sum is approximately  $N(n, n^2)$ .
- Another option is to report just the Bayes Factor or likelihood ratio,  $P(E | H_A) / P(E | H_0)$ , rather than the posterior probability.
- Statistical power is the probability that the test will be positive if the null hypothesis is false. In general, the power of a test depends on the sample size, the magnitude of the effect, and the threshold .
- If there are no outliers, the sample mean minimizes the mean squared error

- An estimator is unbiased if the expected total (or mean) error, after many iterations of the estimation game, is 0.
- A challenge in measuring correlation is that the variables we want to compare might not be expressed in the same units. For example, height might be in centimeters and weight in kilograms. And even if they are in the same units, they come from different distributions. There are two common solutions to these problems:
  1. Transform all values to standard scores. This leads to the Pearson coefficient of correlation.
  2. Transform all values to their percentile ranks. This leads to the Spearman coefficient.

### 10.0.5 AB Testing Overview

Guide: Choose metric (CTR), review stats (Hypothesis Testing), design, analyse

Need to have clear control and thing to test. Also need time. Frequency of customer interaction is important. shopping and searching not independent events.

Point estimate of CTR (users who clicked)/users. Need margin of error for estimate. Approximate binomial by normal if  $N \cdot p$  &  $N(1-p) > 5$ .

margin of error =  $Z \cdot (\text{standard error})$

$se = \sqrt{p(1-p)/n}$

Z for 95% is 1.96 and 2.58 for 99%.

for Two samples calculate a pooled CTR point estimate and standard error.

Need to be substantive (a worthy difference) in addition to being statistically significant. Some changes might not be useful due to the resources one might want to allocate.

Results are repeatable (stat sig) bar should be lower than business interesting (practically sig)

Stat Power: How many page views necessary to get a stat sig result.

Size v Power: The smaller the effect or increased confidence you want to detect/have (greater power of experiment), the larger the sample you need to use.

$\alpha = P(\text{reject null when true})$   $\beta = P(\text{accept null when false})$

small sample: low  $\alpha$ , high  $\beta$  larger sample: low  $\alpha$ , low  $\beta$

$1 - \alpha$  = confidence level  $1 - \beta$  = sensitivity (often 80%)

practical significance: 2%,  $\alpha = 0.05$ ,  $\beta = 0.2$

sample size  $dp\_to\_ctr$  (increase in se), confidence level, sensitivity sample size  $invp\_to\_practical\_sig$  (larger changes easier to detect)

above is design of experiment. below is analysis:

estimate diff: experimental prop - control prop margin of error =  $z \cdot se$

conf int: estimated diff  $\pm$  margin of error

### Policy & Ethics

In the study, what risk is the participant undertaking? The main threshold is whether the risk exceeds that of “minimal risk”.

Next, what benefits might result from the study? Even if the risk is minimal, how might the results help?

Third, what other choices do participants have? For example, if you are testing out changes to a search engine, participants always have the choice to use another search engine.

Finally, what data is being collected, and what is the expectation of privacy and confidentiality? This last question is quite nuanced, encompassing numerous questions: Do participants understand what data is being collected about them? What harm would befall them should that data be made public? Would they expect that data to be considered private and confidential?

Our recommendation is that there should be internal reviews of all proposed studies by experts regarding the questions:

Are participants facing more than minimal risk? Do participants understand what data is being gathered? Is that data identifiable? How is the data handled?

Internal process recommendations Finally, regarding internal process of data handling, we recommend that:

Every employee who might be involved in A/B test be educated about the ethics and the protection of the participants. Clearly there are other areas of ethics beyond what we've covered that discuss integrity, competence, and responsibility, but those generally are broader than protecting participants of A/B tests (cite ACM code of ethics).

All data, identified or not, be stored securely, with access limited to those who need it to complete their job. Access should be time limited. There should be clear policies of what data usages are acceptable and not acceptable. Moreover, all usage of the data should be logged and audited regularly for violations. You create a clear escalation path for how to handle cases where there is even possibly more than minimal risk or data sensitivity issues.

rate better for usability test than prob

## **Choosing & Characterizing Metrics**

### **Define**

Make sure to granularize funnel

### **Intuition**

NA

### **Characterize**

Focus groups, UER (user experience study), survey, experiments, retrospective analysis, external data

temporal effects across hours, days, weeks are important

different browsers screw with the CTR depending on how they deal with JS

Can use cookies to track users

Metric definitions

Def #1 (Cookie probability): For each , number of cookies that click divided by number of cookies

Def #2 (Pageview probability): Number of pageviews with a click within divided by number of pageviews

Def #3 (Rate): Number of clicks divided by number of pageviews

CTR vs CTP: The first can be  $>1$ .

Take into consideration mean and median. Median is robust but might not be useful so look at percentiles.

Measuring sensitivity and robustness: 1) Running experiments to see if metrics move as predicted. A vs A experiments: Measure people who saw the same thing to see if there is a difference between them according to the metric or its too sensitive. Look at previous experiments. 2) Retrospective analysis of logs to see how the metrics responded in the past.

Metric For Loading Vid: Look at distribution of load times for vid. Want a robust metric that doesn't really change for comparable vids. But want a metric that is sensitive to a change that you care about like resolution.

How to compare experiment vs control: difference or relative change. Relative change allows you to keep same significance boundary.

Check that practical sig level is good for metric. Need to have handle on variability for confidence interval.

sign test: Good for looking to implement a change, but doesn't give effect size.

empirical vs analytic: underlying distribution could be weird so do empirical. Analytical for simple metrics. If empirical and analytical don't agree then run AvA test.

A vs A test: Std proportional to square root of number of samples. Diminishing returns for many tests. Can use bootstrap if don't have enough traffic for a big A v A test.

## Designing An Experiment

### Choose Subject

Unit of Diversion: How to identify a person. Cookie (could clear so change), userid, deviceid, and address (changes all the time).

Intra-user vs inter-user experiment.

### Choose Population

Size

Duration

Cohort: People who enter the experiment at the same time.

Cohort > Population: Learning effects, user retention

Can reduce sample if just targeting english speaking traffic

When going to run experiment: holiday or not? How many people to put through experiment and control?

## Analyzing Results

Sanity Checks:

Pass all before move on

Good invariants: # of events, video load time (user has no control over it). Want these to be about the same for experiment and control.

Single Metric:

sign test

Multiple Metrics:

More metrics increase false positive chance. Use Bonferroni assumption: makes no independence assumption and is conservative.

margin of error smaller than observed diff so the confidence interval won't include 0.

Different strategies: family-wise error rate, false discovery rate,

Gotchas:

Simpson's Paradox

Usually A/B testing works for testing changes in elements in the web page. A/B testing framework is following sequence:

Design a research question.

Choose test statistics method or metrics to evaluate experiment.

Designing the control group and experiment group.

Analyzing results, and draw valid conclusions.





# Chapter 11

## Deep Learning

- resnet, attention > RNN / LSTM
- “Training with large minibatches is bad for your health. More importantly, it’s bad for your test error. Friends dont let friends use minibatches larger than 32.” - Yann Le Cun
- DL Cheatsheet
- By reducing learning rate (for example, to 0.001), Test loss drops to a value much closer to Training loss. In most runs, increasing Batch size does not influence Training loss or Test loss significantly. However, in a small percentage of runs, increasing Batch size to 20 or greater causes Test loss to drop slightly below Training loss.
- Reducing the ratio of training to test data from 50% to 10% dramatically lowers the number of data points in the training set. With so little data, high batch size and high learning rate cause the training model to jump around chaotically (jumping repeatedly over the minimum point).
- There are several activation functions you may encounter in practice:
  - Sigmoid: Takes a real-valued input and squashes it to range between 0 and 1 ( $\sigma(x) = 1/(1+\exp(-x))$ )
  - Softmax: Same end result as sigmoid, but different function.
  - tanh: Takes a real-valued input and squashes it to the range [-1, 1] ( $\tanh(x) = 2\sigma(2x) - 1$ )
- ReLU: The rectified linear activation function (called ReLU) has been shown to lead to very high-performance networks. This function takes a single number as an input, returning 0 if the input is negative, and the input if the input is positive.  $f(x) = \max(0, x)$
- softmax doesn’t like multi-label classification
- Embeddings vs One-Hot Encoding: Embeddings are better than One-Hot Encodings because it allows for relationships to be shown between days.
- Advised to scale features
- Rule of 30: A change that affects at least 30 data points in your validation set is usually significant and not just noise.
- Gradient descent takes about 3 times longer than the loss function. Stochastic gradient descent works off an estimate of the loss function from a sample of the training set. Scales better than normal gradient descent.
- Momentum and learning rate decay are good for knowing which way to go in gradient descent.
- Always try to lower your learning rate for improvement.

- ADAGRAD is another optimization option that takes care of some of the hyperparameters for you.
- $n$  inputs and  $k$  outputs gives you  $(n + 1) * k$  parameters.
- Back propogation takes about twice the memory as forward propogation.
- Stop model from overoptimizing on training set: early termination (stop as soon as performance on validation set drops) and regularization which applies artifical constraints to reduce the number of free parameters while making it difficult to optimize. Uses l2 regularization or dropout. Dropout works by randomly setting activations from one layer to the next to 0 repeatly. Forces the network to learn redundant representations, and takes average consensus for final prediction. Makes things more robust and prevents overfitting. Scale the non-zero activates by 2 to get the right average. If it doesn't work, go deeper.
- Two ways are to average the yellow, blue, and green channels or to use YUV representation. If position on the screen doesn't matter then use translation invariance. Use weight sharing if this occurs in text.
- Things that don't chnage across time, space, etc are called statistical invariants.
- RNN's use shared parameters over time to extract patterns. Uses a recurrent connection to provide a summary of the past and pass this info to the classifier.
- Backpropagation occurs throw time to the beginning. All derivative applied to same parameters, so very correlated. Bad for stochastic gradient descent and makes math very unstable. Gradients are either zero (doesn't remember the past well) or infinity. The latter is fixed by gradient clipping and the former by LSTM (long-short term memory). LSTM replaced the rectified linear units by continuous functions. You can use dropout or L2 regularization with an LSTM.
- Generally dealing with images, Convolutional Neural Network is used mostly because of its better accuracy results.
- Model capacity: Same as underfitting and overfitting in bias-variance. Less nodes and hidden layers corresponds to simpler model and vice versa.
- The perceptron is the simplest neural network. The perceptron is an iterative classification method. The perceptron starts with a random hyperplane then adjust its weights to separate the data.
- BackProp Algorithm: Initially all the edge weights are randomly assigned. For every input in the training dataset, the ANN is activated and its output is observed. This output is compared with the desired output that we already know, and the error is "propagated" back to the previous layer. This error is noted and the weights are "adjusted" accordingly. This process is repeated until the output error is below a predetermined threshold.
- The last layer of a neural network captures the most complex interactions.
- When plotting the mean-squared error loss function against predictions, the slope is  $2x(y - xb)$ , or  $2input_{data}error$ .
- $weights\_updated = weights - slope * learning\_rate$
- This is exactly what's happening in the vanishing gradient problem – the gradients of the network's output with respect to the parameters in the early layers become extremely small. That's a fancy way of saying that even a large change in the value of parameters for the early layers doesn't have a big effect on the output.
- Batch: Subset of the data used to calculate slopes during back propagation. Different batches are used to calculate different updates.
- Epoch: One full pass through all the batches in the training data.
- Stochastic gradient descent calculates slopes one batch at a time.

- This network again uses the ReLU activation function, so the slope of the activation function is 1 for any node receiving a positive value as input.
- Few people use kfold cv in deep learning because of the large datasets in play.
- Dying neuron + vanishing gradient: Change activation function



## Chapter 12

# Probabilistic Programming

<https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7>

Bayes Rule:

$\text{old\_prob} [P(h)/p(\sim h)] * \text{strength\_of\_evidence} [p(e|h)/p(e|\sim h)] = \text{new\_prob} [p(h|e)/p(\sim h|e)]$

- Models have a lot of parameters classical methods that use simple point estimates of the parameters don't adequately capture uncertainty so we turn to Bayesian methods because Bayesian inference is one way of finding a large model with metadata.
- The second reason we use Bayesian inference is for combining information you might have experimental data on the drug under one condition and aggregate data under another condition we can combine polls and election forecast of public opinion data from multiple poles and environmental statistics we have measurements of different quality Bayesian methods are particularly adapted to combining information.
- The third appeal of Bayesian methods and the sort of applications that I work on is that the inferences can map directly to decisions based on inferences express not its estimates and standard errors but rather as probability distributions we can take these probability distributions and pipe them directly into decision analysis for these reasons.
- An advantage of recognizing that the prior distribution is a testable part of a Bayesian model is that it clarifies the role of the prior inference, and where it comes from.
- Approximate Bayesian Computation
- Laplace approximation: Approximate the posterior of a non-conjugate model
- When one uses likelihood to get point estimates of model parameters, it's called maximum-likelihood estimation, or MLE. If one also takes the prior into account, then it's maximum a posteriori estimation (MAP). MLE and MAP are the same if the prior is uniform.
- Beta conjugacy: Beta for prior & binomial for likelihood implies beta for posterior. Update:  $\text{beta}(\alpha + \text{successes}, \beta + \text{failures})$ . Mean:  $\alpha / (\alpha + \beta)$
- Small alpha and beta means the prior is less informative
- Empirical Bayes is an approximation to more exact Bayesian methods. With a lot of data it is a very good approximation.

```
from numpy.random import beta as beta_dist
import numpy as np
N_samp = 10000 #number of samples to draw
clicks_A = 450
views_A = 56000
```

```
clicks_B = 345
views_B = 49000
alpha = 1.1
beta = 14.2
A_samples = beta_dist(clicks_A + alpha, views_A - clicks_A + beta, N_samp)
B_samples = beta_dist(clicks_B + alpha, views_B - clicks_B + beta, N_samp)
```

Posterior prob that CTR\_A > CTR\_B given data is `np.mean(A_samples > B_samples)`. Prob that lift of A relative to B is  $\geq 3\%$ : `np.mean(100*(A_samples-B_samples)/B_samples > 3)`

# Chapter 13

## NLP

- **TF-IDF:** A feature vectorization method widely used in text mining to reflect the importance of a term to a document in the corpus. Denote a term by  $t$ , a document by  $d$ , and the corpus by  $D$ .  $TF(t,d)$  is the number of times the term  $t$  appears in document  $d$  while document frequency  $DF(t,D)$  is the number of documents which contain the term  $t$ . If a term appears very often across the corpus it doesn't carry any special info about the document. Inverse document frequency is a numerical measure of how much info a term provides. If a term appears in all documents its IDF value is 0. TF-IDF is the product of TF and IDF. HashingTF uses a hash function to transform input into text.

### Basic Py NLP Pipeline

- Converting text to numerical data is vectorization.
- **Tokenize:** A token is an individual word (or group of words) extracted from a document, using whitespace or punctuation as separators.
- **Create bag of words:** Count tokens within a document and normalised to de-emphasise tokens that appear frequently within a document.
- **Vectorization:** Corpus represented as a large matrix, each row of which represents one of the documents and each column represents token occurrence within that document. CountVectorizer or TF-IDF.
- tf-idf & cosine similarity

### 13.0.1 Topic Modeling

- Pointwise mutual information is any way to evaluate topic models. Uses joint probabilities to evaluate how much a word tells you a label.
- Mutual information generalized pointwise mutual information averaged over all events.
- If you're just using the probability vectors that come out of a topic model you are selecting a probability metric. If you want to do post-processing you something like mutual information or point was mutual information you're selecting another metric. Metrics always have a point of view.
- They're just like human writers. Probability says I like really common words, pointwise mutual information as I like really really really discerning words and I don't care how common they are. Mutual information is sort of a balance between the two.
- In topic modeling using latent Dirichlet distribution, how are "topics" represented? As distributions over words.





## Chapter 14

# Hierarchical Models

“Across counties, as the average mother’s age increases the birth rate decreases while correcting for state-level impacts”  $\text{BirthRate} \sim \text{AverageAgeofMother} + (1 \mid \text{State})$

`lmerTest`

Analysis of Variance, or ANOVA, can be used to compare two different lmer models and test if one model explains more variability than the other model. We can use this to compare two lmer models. For example, if we wanted to see if Year is important for predicting Crime in Maryland, we can build a null model with only County as a random-effect and a year model that includes Year.

- One far out suggestion in one of the links is to do PCA and then regress the original variables against the new axis.
- Account for longitudinal data: random selection of one event from each student, partition cross validation by dates, pca finds major trends hidden within the data