Data Science Cribsheet

Contents

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

Another instantiation via Hadley Wickham is: import -> tidy -> understand [transform <-> visualize <-> model] -> communicate. (ittvmcmd as an abbreviation.) And the two combined: preparation -> import -> tidy -> understand [transform <-> visualize <-> model] -> [communicate + deploy + maintain]. And this is an attempt to fit this into the principles in "The Checklist Manifesto".

1.2 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

1.2.2 Pre-Modeling

- [ ] See what the data looks like by shape, outliers, center, and spread, the types of variables, first and last few observations, and check for missing and blank values.
- [ ] Plot scatterplots, overlayed density plots, trendlines, and histograms.
- [ ] Dimensionality reduction or hierarchical clustering to get a feel for high dimensional structure.

1.2.3 Model

- [ ] Pre-processing - Feature Selection & Engineering
- [ ] Check relationships between features and target: scatterplots and chi-squared test
- [ ] Check model assumptions
- [ ] Model ensembles
- [ ] Model tuning
- [ ] Look at confidence intervals
- [ ] Model explanability and variable importance
- [ ] Look at learning curves

1.3 Resources

Auto Data Science: Visualize Feature_Engineering TPOT Google AutoML AutoML

https://distill.pub/

https://electronjs.org/apps/netron

https://pair-code.github.io/what-if-tool/

https://nextjournal.com/

https://causal.app/

1.4 Pipelines

Sample Pipeline

Source

IO: odbc readxl httr

EDA: DataExplorer

Prep: tidyverse

Sampling: rsample modelr

Feature Engineering: recipes

Modeling: glmnet h2o FFTrees

Evaluation: broom yardstick

Deployment: sqlrutils AzureML opencpu

Monitoring: flexdashboard

Docs: rmarkdown

My Pipeline

Import: odbc dbi readr data.table

Tidy: skimr dplyr tidyr visdat data.table

Transform: dplyr data.table

Visualization: ggplot2

Modeling: fklearn sklearn mlxtend shap yellowbrick sspark, toolbox: classic & bayesian ml, rl, ga & gp: symbolic regression, es, gams, statsmodels

Communicate: rmarkdown

Monitor:

Deploy: shiny

R Pipeline

Import: odbc dbi readr data.table

Tidy: skimr dplyr tidyr visdat data.table

Transform: dplyr data.table

Visualization: ggplot2

Modeling: rsample modelr broom yardstick caret sparklyr

Communicate: rmarkdown

Monitor:

Deploy: shiny

Python Pipeline

Import: pyodbc pandas-modin

Tidy: pandas-modin

Transform: pandas-modin

Visualization: altair plotnine

Modeling: sklearn mlxtend shap yellowbrick pyspark chainlearn

Communicate: jupyter

Monitor:

Deploy: flask

Chapter 2

Pre-Modeling

2.1 Import

2.1.1 General

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.

Make codebooks

Sample from data if its too large

Documentation: 1 | 2 | 3 | 4 | 5 | 6 | 7

CLI: 1 | 2 | 3

pdf table extraction

2.1.2 SQL

Maintain a query library

cross apply = inner join & outer apply = full join

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.

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

2.2 Tidy & Transform

2.2.1 General

In tidy data: Each variable forms a column. Each observation forms a row. Each type of observational unit forms a table. There are three interrelated rules which make a dataset tidy: 1) Each variable must have its own column, 2) Each observation must have its own row, 3) Each value must have its own cell.

Gower distance for categorical dissimilarity.

Use binned statistics.

Character Encoding Precendence: utf-8, iso-8859-1, utf-16

Outlier Detection: interquartile range, kernel density estimation, bonferroni test

Imputation. Options include the Mean, Mode, KNN, Random, and Regression

outliers: 1) dropping is an option but a bad one, 2) create a new columns to mark outliers, 3) rescale with the log to reduce the effect of outliers

tomek link: points of different classes which are neighbors of each other. Removing them often improves model performance

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

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 +- 10% precision? That is d=0.01, e=0.1 and a sample of size 265 should be enough. A stricter precision of +-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: 1) limit number of categories through feature selection, 2) https://stats.stackexchange.com/questions/95212/improve-classification-with-many-categorical-variables, 3) bucket groups by number of samples

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.

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 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- 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.

Dixon and Chi-squared tests for outlier detection plus LOF Algorithm.

2.2.2 Missingness & Imputation

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

Types of imputation: 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

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.

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.

2.3 Visualize

plots: swarmplot | marimekko charts | radviz | Marey Chart

http://pyviz.org/ http://dataviz.tools/ https://www.datawrapper.de/ https://www.data-to-viz.com https://dominikkoch.github.io/Bump-Chart/ https://www.r-graph-gallery.com/ http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html#top

Stats Viz

Seeing Theory

Python Plotting Guide

Py Graph Gallery

ggplot: 1 2 3 Extensions

Sankey Diagram Creator

Waffle Charts

High Dimensional

Links

Tufte In R

R Graph Compendium

Data Viz Catalogue

Text Visualization

2.4 Pre-processing

2.4.1 Feature Engineering

Cyclical Features: 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.

Encode missingness as a feature

Use a robust scaler such as the median & quantiles

On Unbalanced Classes

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

2.4.2 Feature Selection

Use binning + chisquare / mutual information to see correlation between feature and target. Variance Threshold, Univariate Feature Selection, Multivariate Feature Selection. Standardization, dimensionality reduction, dummying. sklearn.preprocessing.binarize, pandas.factorize, and featuretools for automation in Python.

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.

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.

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).

Chapter 3

Model

3.1 General

xgb default hyperparameters good

Random forests are easier to tune and harder to overfit than gradient boosted machines.

KNN memorizes the training data and compares test observations with the nearest training observation. Thus is it a lazy learner.

Pipelines & feature Unions for model deployment

Outlier Detection: interquartile range, kernel density estimation

Dimensionality reduction (PCA, Kernel PCA, TSNE, Isomap)

mixed effect regression

Class imbalance: Gradient boosting is one option.

Use binning + chisquare / mutual information to see correlation between feature and target

Stick to regression models for prescriptive analysis. Validate assumptions and tune appropriately. If using Python leverage statsmodels.

hamming distance for categorical data. Available in sklearn DBSCAN.

precision: how good were you at something being true when you said it was true. recall: how many of the true values did you catch. f1: harmonic mean of the two

Brier score: Don't use for ordinal predictions. 0.25 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.

Algorithms for Rec Systems: Linear/Logistic/Elastic Net Regression, Restricted Boltzmann Machines, Singular Value Decomposition, Makov 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.

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. In general, raising the classification threshold reduces false positives, thus raising precision.

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.

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.

A variable that is completely useless by itself can provide a significant performance improvement when taken with others.

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

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.

SOTA Results

"No Free Lunch" theorem: no single classifier works best across all possible scenarios

Always set the seed

natural log goes from unit change to percentage change

hierarchical clustering: ward

hamming loss: multilabel and multiclass classification

Sequence mining Algorithms: spade, gsp, freespan; hmmlearn

Network models: graphical lasso, ising model, granger causality test, network hypothesis testing, bayesian networks

sequential data: rnn, lstm, hidden markov models & conditional random fields

fowles-mallows score for clustering when ground truth is known

Clustering: Comparison Rules

train & dev set error: low-high = high variance, high-high = high bias, high-high = high bias and variance, low-low = low bias and variance

Opposite the other algorithms, increasing n in KNN means less overfitting.

KNN doesn't easily extrapolate as linear regression.

Linear Regression better than KNN: space to store model, time to query

KNN better than LinReg: compute time to train, ease to add new data

As m increases, AdaBoost tries to assign more and more specific data points to subsequent learners, trying to model all the difficult examples. Thus, compared to simple bagging, it may result in more over-fitting.

SNAP Network Library

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. 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.

"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)."

With large data we can to the train x dev x test split of 98 x 1 x 1.

Bayes optimal error: Aim for this. The difference b/w training error and BOE is the avoidable bias.

new data degrades your model. what should you do? Use the data you have to define a new evaluation metric (using a new dev/test set) taking into account the new species, and use that to drive further progress for your team.

because even though you have higher overall accuracy, you have more false negatives (failing to raise an alarm when a bird is in the air). What should you do? Rethink the appropriate metric for this task, and ask your team to tune to the new metric.

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).

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

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.semi_supervised

Semi-supervised: active learning Auto Labeling https://machinelearningmastery.com/

Active Learning: Finding the optimal data to manually label

Python Distributed

Sklearn Dsitributed

Spark Hub

Interactive ML: 1 2

Online Learning

ML APIs

Auto Optimization

Google ML Glossary

Apache System ML

Learning Curves

Gaussian Processes

Debugging ML

http://gael-varoquaux.info/interpreting_ml_tuto/index.html

LM Model Diagnostics

Gain Curve

Metrics

Choosing A Classifier

LOOCV Perils

Model Evaluations

Model Evaluation

CV Mistakes: 1 2 3

Partial Dependence Plots

https://pair-code.github.io/what-if-tool/

https://pbiecek.github.io/breakDown/ Break Down Plots http://clus.sourceforge.net/doku.php

https://scikit-optimize.github.io/notebooks/sklearn-gridsearchcv-replacement.html https://scikit-optimize.github.io/notebooks/hyperparameter-optimization.html

sklearn metrics Algorithm Comparison Regression Effects Lib

anomalies: skl R

lightgbm: tuning ex

Tree Analysis Viz

unbalanced classes: R Py

Interactive ML

Ensembles: General caret skl 1 2

Multilevel regression and post stratification Mixed Models / multilevel / hierarchical / mixed-effects models Multiple Factor Analysis: FactoMineR Functional PCA

LMM: 1 2

Sklearn Model Selection: 1 2 3

Annotation

Mixed Data Clustering

Dimensionality Reduction Techniques

Unsupervised Learning Sklearn

Info Maximization

K-Means On MNIST

tsne: 1 2 3

3.2 Model Selection/Evaluation

https://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html

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 unalike the data are.

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.

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

Warning PolynomialFeatures(degree=d) transforms an array containing n features into an array containing features, where n! is the factorial of n, equal to $1 \times 2 \times 3 \times \quad \times$ n. Beware of the combinatorial explosion of the number of features!

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.

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.

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 offon 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 dominated 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 intepret 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-parametic, 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.

Sometimes it may be more useful to report the coefficient of determination (), 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:Here, 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 sklean.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:

Naive Bayes is a linear classifier, while KNN is not. The curse of dimensionality and large feature sets are a problem for KNN, while Naive Bayes performs well. KNN requires no training (just load in the dataset), whereas Naive Bayes does.

The McNemar test, introduced by Quinn McNemar in 1947 (McNemar 1947), is a non-parametric statistical test for paired comparisons that can be applied to compare the performance of two machine learning classifiers. Cochran's Q Test or F Test for >2.

Model Comparison

Use SHAP for Model Explanability

Adversarial validation: Makes it easier to check distribution changes over time for train and validation/test data. Combine train and valid/test data into one dataframe, assign labels 1 and 0 to different data and build a binary classification model. If it has high AUC, then there are variables, which separate train and test data - this means that their distributions are quite difference. So you can take top features by feature importance (or other metric) and drop them (at first look at them of course) to make models more stable.

summary and plot for r models

optuna: https://towardsdatascience.com/how-to-make-your-model-awesome-with-optuna-b56d490368a
https://github.com/PiotrekGa/optuna_article/blob/master/Example.ipynb

3.3 Supervised

3.3.1 GLM

3.3.1.1 Linear Regression

radial basis to attach time awareness linear regression

add interaction terms and polynomial features

https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.
GaussianProcessRegressor.html#sklearn.gaussian_process.GaussianProcessRegressor

Discuss Ridge, Lasso, and Elastic Net here.

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.

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 variance=meanvariance=mean.

Ridge/Lasso:

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 shrunken, 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 forcessome 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.

Quantile Regression models the median while Linear Regression models the mean

Linear Regression - BLUE: Best Linear Unbiased (average amount wrong by is error) Estimator

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 => 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 asses 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.

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: $y = b_0 + b_1 x + \epsilon$, ϵ is i.i.d from N(0, σ). 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 en = betan-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-colinearity: 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 ti 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 facotrs influence hte uncertainity of coefficient estimates. >=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.

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-testr for simple and multiple linear regression.

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.

- Forward and Backward Stepwise Selection.
- For simple linear regression, the principal hypothesis test is as follows: H_0 : ? 1 = 0, H_A: ? 1 0. If H_0 is true then the population mean of Y would be ? 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 h0 fail to reject than 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 stringest 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 is 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.)

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

https://www.rdocumentation.org/packages/GGally/versions/1.4.0/topics/ggcoef

https://cran.r-project.org/web/packages/jtools/vignettes/summ.html

- https://twitter.com/alexpghayes/status/1033824995905077250
- Quantile regression is an alternative for quantiles.
- Type 1 Error: Accept ha when h0 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 – RSS/TSS.
- Increasing x1 by 1 means you are increasing the underlying X1 by 1 standard deviation x1+1=(X1+sig-mu)/sig. Therefore you can say that b1 is the effect on y of increasing X1 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 r is a greedy way to find the best predictors by cycling through all the combinations.

The importance of a feature in a linear regression model can be measured by the absolute value of its t-statistic. The t-statistic is the estimated weight scaled with its standard error.

effect of linear regression is w_ix_i. A feature effect plot shows the distribution of effects throughout the data.

3.3.1.2 Logistic Regression

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 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 incorpotate prior knowledge, number fo 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 l2 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.

- 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 (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 Multicolinearity

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

Linear With The Logit Of Predictions

- Wald Test: h_0: b = 0 and h_a: b =/= 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 h0 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

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

with bayesian log reg you can have a probability distribution of the probability estimate so you can attach a ci to the prediction

3.3.2 Generalized Additive Models

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.

Pros

Cons

3.3.3 KNN

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.

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.

3.3.4 SVM

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 trade-off 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 intial 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 kernelis 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 obervations. Take a penalty by potentially misclassifying soe 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 efficientbecause 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.

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

3.3.5 Decision Tree

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 stochastic6 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 meant of the reponse variables in each region for the training observations in that space. Top down and greedy approach called recursive binary spltting to find ebst 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. Maximial 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 crossvalidationprocess, 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 biasbecause the trees are much simpler.

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

3.3.6 Bagging

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 (Xi, Yi), where i=1 …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 treesto 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

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

3.3.7 Random Forest

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.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 = √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 consideredin 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 scoing 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.

Pros

Decorrelates trees (relative to boosted trees)

important when dealing with mulitple 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

3.3.8 Boosting

Unlike boosting 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 dependentupon one another. Boosting approach tends to slowly learn our data. Given a current decision tree model, we fit a new decision tree to the residualsof 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 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

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

3.3.9 Naïve Bayes

Laplace smoothing/estimator in Naive Bayes: To see why consider the worst case where none of the words in the training sample appear in the test sentence. In this case, under your model we would conclude that the sentence is impossible but it clearly exists creating a contradiction.

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.

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 probabilityfor 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.

3.3.10 LDA & QDA

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 -dimensional dataset (is the number of features).

For each class, compute the -dimensional mean vector.

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

Compute the eigenvectors and corresponding eigenvalues of the matrix .

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

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

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.

3.3.11 GAMs

Wiggleness affected by smoothing factor (sp) and number of basis functions (k)

gam(y ~ s(x, k = 3), data = df, sp = .1) gam(y ~ s(x, k = 3), data = df, method = "REML")

Categorical features must be included as factors.

gam(hw.mpg ~ s(weight) + fuel, data = mpg, method = "REML")

A factor smooth interaction fits different models for different categorical vars.

gam(hw.mpg ~ s(weight, by = fuel) + fuel, data = mpg, method = "REML")

Summary:

edf=1 is a straight line

Visualize:

plot(mod, select = 1, seWithMean = TRUE, shift = coef(mod)[1])

Model Evaluation:

gam.check: Bad if model hasn't converged. Likely due to too many features. Small p-values indicate residuals not randomly distributed which is often due to not enough basis functions.

first do concurvity(mod, full = T) then do concurvity(mod, full = F). check co-linearity. worst case is bad over 0.8.

interaction b/w x & y: mod2d <- gam(cadmium ~ s(x, y), data = meuse, method = "REML")

vis.gam for visualization

vis.gam(mod2d, view = c("x", "y"), plot.type = "persp", se = 2, theta = 135)

vis.gam(mod2d, view = c("x", "y"), plot.type = "contour", too.far = 0.05)

vis.gam(mod2d, view = c("x", "y"), plot.type = "contour", too.far = .1)

Make plot with 25% extrapolation vis.gam(mod2d, view = c("x", "y"), plot.type = "contour", too.far = .25)

Overlay data points(meuse)

Nonlinear Modeling in R with GAMs:

Fit a model with a factor-smooth interaction: mod_fs <- gam(copper ~ s(dist, landuse, bs = "fs"), data = meuse, method = "REML")

Plot both models making 3D perspective plots: vis.gam(mod_sep, view = c("dist", "landuse"), plot.type = "persp")

2-way interactions: 2d interaction smooths, categorical-continuous interaction, tensor smooths: allow to model on different scales (space & time)

tensor_mod <- gam(cadmium ~ te(x, y, elev), data = meuse, method = "REML")

Single-variable smooths use s(). Smooths in which multiple variables interact on the same scale also use s(). Smooths that have only interactions, but have multiple lambda terms or scales, use ti().

te() must be used along. if you include s() switch to ti()

classification:

outputs on the log odds scale

gams work with broom and caret.

3.3.12 Hierarchical Models

"Across counties, as the average mother's age increases the birth rate decreases while correcting for state-level impacts" BirthRate ~ AverageAgeofMother + (1 | State)

https://www.jstatsoft.org/article/view/v067i01/0

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

3.4 Unsupervised

3.4.1 K-Means Clustering

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 centes 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.

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.

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.

3.4.2 Hierarchical Clustering

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 dissimialirities. Evaluate pairwise intercluster dissimiliarities 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 intercluster dissimilarity, single linkage: minimal inter-cluster dissimilarity, avergae 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 effcient run-time wise.

3.4.3 Dimensionality Reduction

PCA, Kernel PCA, TSNE, Isomap

3.4.3.1 PCA

Center and scale data. The eigenvectors yield orthogonal directions of greatest variability(principal components). The eigenvalues correspond to the magnitude of variancealong 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.

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 comunalities; 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 1×1 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 this probability is greater than 99.999999%. Most points in a high-dimensional hypercube are very close to the border.

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.

3.4.4 Multiple Correspondence Analysis

PCA for categorical data

3.5 Semi Supervised

TBD

3.6 Reinforcement Learning

TBD

3.7 Other ML

3.7.1 Association Rule Mining

- 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.

- Tutorial: http://pbpython.com/market-basket-analysis.html
- 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. Support(item) = count(item)/N. Higher support is better.
- nfidence: 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.
- ft: the ratio by which the confidence of a rule exceeds the expected outcome. When lift > 1, the presence of X seems to have increasedthe probability of Y occurring in the transaction. When lift < 1, the presence of X seems to have decreasedthe probability of Y occurring in the transaction. When 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.

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!

3.7.2 NLP

https://gking.harvard.edu/readme https://web.stanford.edu/~jurafsky/slp3/ https://finetune.indico.io http://textable.io/ https://pypi.python.org/pypi/tmtoolkit http://www.structuraltopicmodel.com/ https://blog.insightdatascience.com/how-to-solve-90-of-nlp-problems-a-step-by-step-guide-fda605278e4e

Kaggle Guide Chat Bots Doc Similarity Vector Embeddings

Topic Modeling: 1 2 3

qdap Web Devs Bot Makers NLP Intro

3.7.3 Geospatial

https://geocompr.robinlovelace.net/adv-map.html#interactive-maps https://www.jessesadler.com/post/gis-with-r-intro/ https://uber.github.io/kepler.gl/#/ http://gisquick.org/ https://carto.com/blog/inside/CARTOframes-python-interface-CARTO/

R Spatial

tilegrams

Spatial DA R

3.7.4 AI

https://www.stateoftheart.ai/ http://airesources.org/ https://alleninstitute.org http://allenai.org/index.html http://geometry.allenai.org/ https://competitions.codalab.org/competitions/20013 http://www.hbcse.tifr.res.in/jrmcont/notespart1/node27.html

AI Music Generation AI Acheivements Create With AI Playing Games AI 4 Py

3.7.5 CV

https://www.pyimagesearch.com/

3.7.6 Online Learning

Incremental PCA

3.7.7 Sequences

Hidden Markov Model

3.7.8 Federated ML

federated learning vs split learning

https://blog.openmined.org/upgrade-to-federated-learning-in-10-lines/ https://onnx.ai/onnx-r/

Chapter 4

Communicate, Deploy, Maintain

4.1 Communicate

4.1.1 Outline

Reference this blog.

Summary

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

Results

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.

Conclusion

Discuss recommendations, outstanding issues, and possible future work.

4.1.2 Other

Write appendices and port mortems

Interactive Rmarkdown reports without shiny: plotly, dt, crosstalk

https://metabase.com/ http://koaning.io/make-docs.html Mode Studio

4.2 Maintain

TDD: 1 | 2 | 3

Tests: 1 | 2 | 3 | 4 | R | pytest/hypothesis

Validation: 1 | 2 | 3

dvc https://matrixds.com/ | https://www.crestle.com | CoCalc

design by contract: contracts, unit tests: pytest Hypothesis (property based testing), static typing: mypy

packrat & conda

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.

4.3 Deploy

DE: aaS data build tool https://www.stitchdata.com/ https://www.prefect.io/ https://mlflow.org/ Apache Arrow

https://www.meltano.com/ http://pachyderm.io https://datacoral.com/ Algorithmia https://cloud.google.com/ai-platform/ https://ropensci.org/technotes/2019/03/18/drake-700/ Weights & Biases Databricks Ansible https://aws.amazon.com/sagemaker/ https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

Pipelines Workflow System Pipeline Automation Streaming Data Spark

batch job execution framework Sklearn Serialization Auto Deploy Model Deployment

use makefile to run cmd commands pipelines and feature unions Sig Opt fabric-python & http://www.pyinvoke.org/

Chapter 5

Stats

5.1 General

Studies with low statistical power, like those with small samples, are more likely to show false negative and false positive results.

Cohen's Alphabet (H D F^2) Multiple comparisons Problem Importance sampling Bayes Error Rate Pearson's Coefficient of Skewness Bayes factor Correction methods for non representative samples Mann–Whitney U test

A mixture distribution is a distribution whose density is a linear combination of normal distribution densities (components).

Probability Cheat Sheet

Prob Dists In R

Effect Size

Zero correlation doesn't imply independence

lm vs t.test vs anova library(EMT): multinomial test

ecdf: 1 2

Mixed Categorical & Numeric Distance 2 Outliers in small samples

Another common measure of effect size is d, sometimes known as Cohen's d. 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.

The Pearson Correlation is normalized version of covariance

Populations have parameters. Samples have statistics.

Probably the most useful types of statistics for skewed probability distributions are quantiles.

Random sampling & random assignment: generalizable & causal

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.

Sum of normally distributed random variables: sum of means & sum of variances

Zero correlation doesn't mean independent variables. Covariance only determines linear relationships.

Bypass inferential stats if features at least 1/10 of data points.

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 Cor (X,Y)=0

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.

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.

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

Random Sampling + Random Assignment: Causal + generalizable

5.2 Sample Size

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 +- 10% precision? That is d=0.01, e=0.1 and a sample of size 265 should be enough. A stricter precision of +-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.

5.3 Hypothesis Testing

There is only one test.

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 H0 is contained within the 95% confidence interval, then H0 cannot be rejectedat the 0.05 p-value threshold. If the value of the parameter specified by H0 is not contained within the 95% confidence interval, then H0 can be rejectedat the 0.05 p-value threshold.

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.

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.

jarque.bera.test: Tests the null of normaility for the series using the Jarque-Bera test statistics

test heteroscedaticity: breusch-pagan or ncv test

box.test: Compute the Box-Pierce or Ljung-Box test statistics for examining the null of independence in a given series
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. 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.

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.

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:

Most Tests Are Linear Regression: lm() with a single categorical feature is equivalent to a t test or anova test.

Fisher's Exact Test: Test for proportions

Hypothesis testing requires an inferential-statistical approach. Crucial are meaningful distributions of test statistics, on which p-values for hypothesis tests can be based.

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.

5.4 AB Testing

Mostly notes from the Udacity class.

Power-Of-Test: https://rpsychologist.com/d3/NHST/ https://juliasilge.shinyapps.io/power-app/ https://www.statmethods.net/stats/power.html https://www.graphpad.com/quickcalcs/binomial1.cfm https://stat.ethz.ch/R-manual/R-devel/library/stats/html/power.prop.test.html https://stat.ethz.ch/R-manual/R-devel/library/stats/html/prop.test.html Power Analysis AB Testing

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, power = 1 − . 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(Type I error) = probability of finding an effect that is not there
- power = 1 - P(Type II error) = probability of finding an effect that is there

Given any three, we can determine the fourth.

- 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

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 interaqction 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*p & N(1-p) > 5.

marin of error = Z*(stndad error)

se = sqrt(p(1-p)/n)

Z for 95% s 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.

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

Stat Power: How many page view 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(reject null when true) beta = P(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*se

conf int: estimated diff +/- margin of error

rate better for usability test than prob

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 sensitiviyt 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 wnat 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 praticial 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. Analyticcal for simple metrics. If empirical and anlytical 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.

Unit of Diversion: How to indentify 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.

5.5 Experimental Design

Notes from DataCamp

Randomized Complete Block Design (RCBD) experiment

Power usually @ 80%. power effect size & sample size all inter-related (with 2 can calculate 3rd).

power: prob that test correctly rejects null hypo when alt hypo is true

effect size: standardized measure of the difference you're trying to detect

sample size: how many experimental units needed to survey to detect the desired diff at the desired power

pwr::pwr.anova.test() [exactly one of n, d, power, and sig.level must be NULL]

aov = lm + anova | aov + tukey hsd

Another assumption of ANOVA and linear modeling is homogeneity of variance. Homogeneity means "same", and here that would mean that the variance of int_rate is the same for each level of grade. We can test for homogeneity of variances using bartlett.test(), which takes a formula and a dataset as inputs.

One non-parametric alternative to ANOVA is the Kruskal-Wallis rank sum test. For those with some statistics knowledge, it is an extension of the Mann-Whitney U test for when there are more than two groups, such as with our grade variable.

Use the correct function from pwr to find the sample size (d is the effect size) pwr::pwr.t.test(n = NULL, d = .2, power = .8, sig.level =.05, alternative = "two.sided")

multivariable experiment:

lendingclub_multi <- lm(loan_amnt ~ Group + grade + verification_status, lendingclub_ab) tidy(lendingclub_multi)

The sampling package's cluster() creates cluster samples. It takes in a dataset name, the variable in the set to be used as the cluster variable, passed as a vector with the name as a string, e.g. c("variable"), a number of clusters to select, and a method

Randomized Complete Block Designs

the purpose of blocking an experiment is to make the experimental groups more like one another.

A rule of thumb in experimental design is often "block what you can, randomize what you cannot", which means you should aim to block the effects you can control for (e.g. sex) randomize on those you cannot (e.g. smoking status). Variability inside a block is expected to be fairly small, but variability between blocks will be larger.

Chapter 6

Sequences

6.1 Time Series

6.1.1 Notes

ARIMA models don't work well on financial data.

Distance Measures: Dynamic Time Warping & Frechet Distance

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

Another time series with multiple seasonal periods is calls, which contains 20 consecutive days of 5-minute call volume data for a large North American bank. There are 169 5-minute periods in a working day, and so the weekly seasonal frequency is 5 x 169 = 845. The weekly seasonality is relatively weak, so here you will just model daily seasonality. The residuals in this case still fail the white noise tests, but their autocorrelations are tiny, even though they are significant. This is because the series is so long. It is often unrealistic to have residuals that pass the tests for such long series. The effect of the remaining correlations on the forecasts will be negligible.

Box-Cox transformation parameter

Annual data can't be seasonal. Seasonal has fixed lengths, cyclical doesn't. When data are either seasonal or cyclic, the ACF will peak around the seasonal lags or at the average cycle length.

White noise is iid. 95% of autocorrelations should be within blue lines of acf plot.

There is a well-known result in economics called the "Efficient Market Hypothesis" that states that asset prices reflect all available information. A consequence of this is that the daily changes in stock prices should behave like white noise

(ignoring dividends, interest rates and transaction costs). The consequence for forecasters is that the best forecast of the future price is the current price.

What happens when you want to create training and test sets for data that is more frequent than yearly? If needed, you can use a vector in form c(year, period) for the start and/or end keywords in the window() function. You must also ensure that you're using the appropriate values of h in forecasting functions. Recall that h should be equal to the length of the data that makes up your test

multi-step time series forecasting: Predicting multiple time steps into the future. There are four main strategies that you can use for multi-step forecasting.

one-step forecast: Time series forecasting describes predicting the observation at the next time step.

Methods with multiplicative trend produce poor forecasts.

Differencing is a way of making a time series stationary; this means that you remove any systematic patterns such as trend and seasonality from the data. A white noise series is considered a special case of a stationary time series. With non-seasonal data, you use lag-1 differences to model changes between observations rather than the observations directly. You have done this before by using the diff() function.

With seasonal data, differences are often taken between observations in the same season of consecutive years, rather than in consecutive periods. For example, with quarterly data, one would take the difference between Q1 in one year and Q1 in the previous year. This is called seasonal differencing. Sometimes you need to apply both seasonal differences and lag-1 differences to the same series, thus, calculating the differences in the differences.

Time series that show no autocorrelation are called white noise. For white noise series, we expect each autocorrelation to be close to zero. Of course, they will not be exactly equal to zero as there is some random variation. For a white noise series, we expect 95% of the spikes in the ACF to lie within 2 ks where s is the square root of the length of the time series.

For stock market prices and indexes, the best forecasting method is often the naïve method.

More often than not, time series data are "non-stationary"; that is, the values of the time series do not fluctuate around a constant mean or with a constant variance.

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)

interventions, eg. competitor activity | trading days: bizdays() | distributed lags | easter()

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

We think of a time series as comprising three components: a trend-cycle component, a seasonal component, and a remainder component (containing anything else in the time series).

The additive decomposition is the most appropriate if the magnitude of the seasonal fluctuations, or the variation around the trend-cycle, does not vary with the level of the time series. When the variation in the seasonal pattern, or the variation around the trend-cycle, appears to be proportional to the level of the time series, then a multiplicative decomposition is more appropriate. Multiplicative decompositions are common with economic time series.

An alternative to using a multiplicative decomposition is to first transform the data until the variation in the series appears to be stable over time, then use an additive decomposition. When a log transformation has been used, this is equivalent to using a multiplicative decomposition

Remove seasonality to make data seasonally adjusted. Remove when not needed.

A time series decomposition can be used to measure the strength of trend and seasonality in a time series. A series with seasonal strength close to 0 exhibits almost no seasonality, while a series with strong seasonality will have close to 1

While decomposition is primarily useful for studying time series data, and exploring historical changes over time, it can also be used in forecasting.

To forecast a decomposed time series, we forecast the seasonal component and the seasonally adjusted component separately. It is usually assumed that the seasonal component is unchanging, or changing extremely slowly, so it is forecast by simply taking the last year of the estimated component. In other words, a seasonal naïve method is used for the seasonal component.

To forecast the seasonally adjusted component, any non-seasonal forecasting method may be used. For example, a random walk with drift model, or Holt's method (discussed in the next chapter), or a non-seasonal ARIMA model.

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 reseasonalize the forecasts: stlf(elecequip, method='naive')

- 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, constant variance). Also called a stationary series. Time series with trends or seasonality are not stationary.

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.

An alternative to estimating the parameters by minimizing the sum of squared errors is to maximize the "likelihood". The likelihood is the probability of the data arising from the specified model. Thus, a large likelihood is associated with a good model. For an additive error model, maximizing the likelihood gives the same results as minimizing the sum of squared errors. However, different results will be obtained for multiplicative error models.

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

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

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.

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.

A stationary time series is one whose properties do not depend on the time at which the series is observed.14 Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times. On the other hand, a white noise series is stationary — it does not matter when you observe it, it should look much the same at any point in time. Some cases can be confusing — a time series with cyclic

behaviour (but with no trend or seasonality) is stationary. This is because the cycles are not of a fixed length, so before we observe the series we cannot be sure where the peaks and troughs of the cycles will be. In general, a stationary time series will have no predictable patterns in the long-term. Time plots will show the series to be roughly horizontal (although some cyclic behavior is possible), with constant variance.

Most time series models do not work well for very long time series. The problem is that real data do not come from the models we use. When the number of observations is not large (say up to about 200) the models often work well as an approximation to whatever process generated the data. But eventually we will have enough data that the difference between the true process and the model starts to become more obvious. An additional problem is that the optimization of the parameters becomes more time consuming because of the number of observations involved.

dicky fuller test

6.1.2 Workflow

Let's review the process:

First, import and load your data. Determine how much of your data you want to allocate to training, and how much to testing; the sets should not overlap.

Subset the data to create a training set, which you will use as an argument in your forecasting function(s). Optionally, you can also create a test set to use later.

Compute forecasts of the training set using whichever forecasting function(s) you choose, and set h equal to the number of values you want to forecast, which is also the length of the test set.

To view the results, use the accuracy() function with the forecast as the first argument and original data (or test set) as the second.

Pick a measure in the output, such as RMSE or MAE, to evaluate the forecast(s); a smaller error indicates higher accuracy.

Ex-ante forecasts are those that are made using only the information that is available in advance. Ex-post forecasts are those that are made using later information on the predictors. These are not genuine forecasts, but are useful for studying the behavior of forecasting models.

6.1.3 Preprocessing

6.1.3.1 General

So far, we have considered relatively simple seasonal patterns such as quarterly and monthly data. However, higher frequency time series often exhibit more

complicated seasonal patterns. For example, daily data may have a weekly pattern as well as an annual pattern. Hourly data usually has three types of seasonality: a daily pattern, a weekly pattern, and an annual pattern

The best way to deal with moving holiday effects is to use dummy variables. However, neither STL, ETS nor TBATS models allow for covariates. Amongst the models discussed in this book (and implemented in the forecast package for R), the only choice is a dynamic regression model, where the predictors include any dummy holiday effects (and possibly also the seasonality using Fourier terms).

To impose a positivity constraint, simply work on the log scale, by specifying the Box-Cox parameter lambda = 0.

we can transform the data using a scaled logit transform which maps 9a,b) to the whole real line: y = log((x-a)/(b-x))

One way to make a non-stationary time series stationary — compute the differences between consecutive observations. This is known as differencing. The ACF of the differenced Google stock price looks just like that of a white noise series. There are no autocorrelations lying outside the 95% limits, and the Ljung-Box statistic has a p-value of 0.355 (for h=10). This suggests that the daily change in the Google stock price is essentially a random amount which is uncorrelated with that of previous days. When the differenced series is white noise, the model for the original series can be written as y_t - y_{t-1} = e_t (white noise)

6.1.3.2 Differencing

This process of using a sequence of KPSS tests to determine the appropriate number of first differences is carried out by the function ndiffs()

A similar function for determining whether seasonal differencing is required is nsdiffs(), which uses the measure of seasonal strength

In practice, it is almost never necessary to go beyond second-order differences.

A seasonal difference is the difference between an observation and the previous observation from the same season.

If seasonally differenced data appear to be white noise, then an appropriate model for the original data is y_t = y_{t-m} + e_t. Forecasts from this model are equal to the last observation from the relevant season. That is, this model gives seasonal naïve forecasts.

Sometimes it is necessary to take both a seasonal difference and a first difference to obtain stationary data

When both seasonal and first differences are applied, it makes no difference which is done first—the result will be the same. However, if the data have a strong seasonal pattern, we recommend that seasonal differencing be done first,

because the resulting series will sometimes be stationary and there will be no need for a further first difference. If first differencing is done first, there will still be seasonality present.

One way to determine more objectively whether differencing is required is to use a unit root test. These are statistical hypothesis tests of stationarity that are designed for determining whether differencing is required.

A number of unit root tests are available, which are based on different assumptions and may lead to conflicting answers. In our analysis, we use the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test. In this test, the null hypothesis is that the data are stationary, and we look for evidence that the null hypothesis is false. Consequently, small p-values (e.g., less than 0.05) suggest that differencing is required. The test can be computed using the ur.kpss() function from the urca package.

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.

6.1.4 Feature Engineering

Various lags of predictors

Rolling mean, min, max, etc statistics

Bollinger bands

Rolling entropy or majority for categorical features

Rolling text statistics for text features

Log-transformation (for exponential trends and multiplicative models)

Periodic differencing (to make an integrated model with stationary targets)

Simple naive difference (using the most recent FDW row)

Extract data from timestamp (year, month, day)

Rolling windows: summary stats of all previous data points

transforms: square root, cube root

inverse (increasing strength of transformation from left to right), powers. Equal to lambda = 1/2, L = 1/3, L = 0, L = -1 for Box-Cox. (L=0 is no transformation.) boxCox()/boxCox.lambda. Not common to use box-cox with ets since ets can handle seasons & trends. Use box-cox with arima models when there is increasing variation

scaling: by monthday, by population, inflation for money, 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.

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.

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

- A trend variable can be specified in the tslm() function using the trend predictor.
- dummy variables for time like day of week. The tslm() function will automatically handle this situation if you specify the predictor season.

An alternative to using seasonal dummy variables, especially for long seasonal periods, is to use Fourier terms. ith 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))

6.1.5 Packages / Functions

diff() for daily changes

statsmodels (python) & forecast (R) & prophet (both)

lubridate for dates

rsample for cross validation

subset.ts() | meanf | mape allows for cross model comparison

library(forecast) | ts | autplot / autolayer | ggseasonplot | ggsubseriesplot | gglagplot | window | meanf | s/naive | residuals | gghistogram | ggAcf | ounterfactual plot

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

accuracy()

tsCV()

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.

checkresiduals()

tslm()

splinef(lambda=0)

wma

arima()

ggAcf() | ggPacf()

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()

msts | mstsl | tbats | bld.mbb.bootstrap

Model Evaluation: train-test split: window() / subset(), accuracy(), tsCV()

forecast: ets_fit %>% forecast(h=8)

gts() | hts() | aggts() | stlf | hts::

ggAcf() | ggPacf()

6.1.6 Models

6.1.6.1 General

In this model, the slope can be interpreted as an elasticity, and is the average percentage change in y resulting from a 1% increase in x. Other useful forms can also be specified. The log-linear form is specified by only transforming the forecast variable and the linear-log form is obtained by transforming the predictor.

It is not recommended that quadratic or higher order trends be used in forecasting. When they are extrapolated, the resulting forecasts are often unrealistic.

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

Each method is labelled by a pair of letters (T,S) defining the type of 'Trend' and 'Seasonal' components. For example, (A,M) is the method with an additive trend and multiplicative seasonality; (A_d, N) is the method with damped trend and no seasonality, and so on.

short_hand	method
(n,n)	Simple exponential smoothing
(a,n)	Holt's linear method
(a_d,n)	Additive damped trend method

short_hand	method
(a,a)	Additive Holt-Winters' method
(a,m)	Multiplicative Holt-Winters' method
(a_d,m)	Holt-Winters' damped method

Each model consists of a measurement equation that describes the observed data, and some state equations that describe how the unobserved components or states (level, trend, seasonal) change over time. Hence, these are referred to as state space models. For each method there exist two models: one with additive errors and one with multiplicative errors. The point forecasts produced by the models are identical if they use the same smoothing parameter values. They will, however, generate different prediction intervals.

To distinguish between a model with additive errors and one with multiplicative errors (and also to distinguish the models from the methods), we add a third letter to the classification of the table above. We label each state space model as ETS (Error, Trend, Seasonal). 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}.

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

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

ETS(A,A,N): Holt's linear method with additive errors

ETS(M,A,N): Holt's linear method with multiplicative errors

Random walk models are widely used for non-stationary data, particularly financial and economic data. Random walks typically have: long periods of apparent trends up or down, sudden and unpredictable changes in direction.

In a multiple regression model, we forecast the variable of interest using a linear combination of predictors. In an auto-regression model, we forecast the variable of interest using a linear combination of past values of the variable. We refer to this as an AR(p) model, an autoregressive model of order p.

Rather than using past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression-like model. We refer to this as an MA(q) model, a moving average model of order q.

If we combine differencing with autoregression and a moving average model, we obtain a non-seasonal ARIMA model. ARIMA is an acronym for AutoRegressive Integrated Moving Average

6.1.6.2 Benchmark/Baseline Models

0. Mean: The mean

1. Naive (forecast::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. This method works remarkably well for many economic and financial time series. Because a naïve forecast is optimal when data follow a random walk, these are also called random walk forecasts.
2. Seasonal Naive (forecast::snaive) : Assumes magnitude of the seasonal pattern is constant
3. statsmodels.tsa.holtwinters & forecast::ets

6.1.6.3 Innovation State Space Models

Given:

trends = {n, a, a_d} {none, additive, damped}

seasons = {n, a, m} {none, additive, multiplicative}

error = {a,m}

There are 9 possible exponential smoothing methods and 18 possible state space models. Known as ets models. Estimated using likelihood. Choose model by minimizing AIC_c (biased correct AIC). Roughly the same as time series CV but is much faster: ets()

6.1.6.4 ARIMA

arima: linear

auto.arima(). To make auto.arima() work harder to find a good model, add the optional argument stepwise = FALSE to look at a much larger collection of models.

seasonal arima: arima (p,d,q) (P,D,Q)_m. Coefficients are hard to interpret for original data.

austa %>% Arima(order = c(2,1,3), include.constant = T) %>% forecast() %>% autoplot()

The Arima() function can be used to select a specific ARIMA model. Its first argument, order, is set to a vector that specifies the values of p, d and q. The second argument, include.constant, is a boolean that determines if the constant c, or drift, should be included.

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.

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

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.

If the data are from an ARIMA(p,d,0) or 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 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 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.

seaosnal arima: arima(p,d,q) (P,Q,D)_m, m = 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 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(euretail, order=c(0,1,3), seasonal=c(0,1,1))

auto.arima(euretail, stepwise=FALSE, approximation=FALSE)

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

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

- AR(p): auto-regressive component for lags on the stationary series. The AR models sayd 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 valeu 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. h0: series is not stationary, ha: 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).

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

1. Plot the data and identify any unusual observations.
2. If necessary, transform the data (using a Box-Cox transformation) to stabilize the variance.
3. If the data are non-stationary, take first differences of the data until the data are stationary.
4. Examine the ACF/PACF: Is an ARIMA(p,d,0) or ARIMA(0,d,q) model appropriate?
5. Try your chosen model(s), and use the AICc to search for a better model.
6. 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.
7. Once the residuals look like white noise, calculate forecasts.

auto.arima only does steps 3-5.

The same stationarity and invertibility conditions that are used for autoregressive and moving average models also apply to an ARIMA model.

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)

It is sometimes possible to use the ACF plot, and the closely related PACF plot, to determine appropriate values for p & q.

If the data are from an ARIMA(p,d,0) or 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 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 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.

The inclusion of a constant in a non-stationary ARIMA model is equivalent to inducing a polynomial trend of order d in the forecast function.

The seasonal part of an AR or MA model will be seen in the seasonal lags of the PACF and ACF. For example, an 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, …).

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.

6.1.6.5 Dynamic Harmonic Regression

forecast::tslm()

A regression model containing Fourier terms is often called a harmonic regression because the successive Fourier terms represent harmonics of the first two Fourier terms.

In dynamic regression the error term is an arima process.

dynamic harmonic regression uses Fourier series: e_t is non-seasonal since Fourier handles that. Assumes seasonality doesn't change. Only need to select k for compliexty. Start from k=1 and choose model with lowest aic_c. k <= m/2 where m is the seasonal period. Good when the seasonality is very large, weekly, daily, sub-daily, etc. The higher the order (K), the more "wiggly" the seasonal pattern is allowed to be. With K=1, it is a simple sine curve.

Harmonic regressions are also useful when time series have multiple seasonal patterns

harmonic regression: An alternative to using seasonal dummy variables, especially for long seasonal periods, is to use Fourier terms. ith 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))

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(cafe04, xreg = fourier(cafe04, K = i), seasonal = FALSE, lambda = 0)

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()

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.

6.1.6.6 TBATS

forecast::tbats()

Automates everything. Good for data with large and multiple seasonal periods.

6.1.6.7 Holt-Winters

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. 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.

Holt's Linear Trend: 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-Winters Method: Accounds for trend and seasonality. In the additive version the seasonality averages to 0 but this changes to 1 in the multiplicative version: hw(df, seasonal = 'additive')

holt linear trend: holt(). 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. holt-winters seasonal method: hw(aust,seasonal="additive"). A method that often provides accurate and robust forecasts for seasonal data is the Holt-Winters method with a damped trend and multiplicative seasonality: hw(y, damped=TRUE, seasonal="multiplicative")

6.1.6.8 Moving Average Methods

ma wma

simple moving average of order m: ma(elecsales, 5). m is usually odd so the sma is symmetric. It is possible to apply a moving average to a moving average. One reason for doing this is to make an even-order moving average symmetric. The notation "2x4-MA" in the last column means a 4-MA followed by a 2-MA. Called a centered MA of order 4. The most common use of centered moving averages is for estimating the trend-cycle from seasonal data. In general, a 2xm-MA is equivalent to a weighted moving average of order m+1 where all observations take the weight 1/m except for the first and last terms which take weights 1/2m.

6.1.6.9 Exponentia Smoothing

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 large alpha puts more weight on recent values and the weights decay quickly. ses(). Has the same value for all forecasts.

SEATS: seasonal::seas()

simple exponential smoothing: For example, it may be sensible to attach larger weights to more recent observations than to observations from the distant past. This is exactly the concept behind simple exponential smoothing. It has the weighted average form, component form,

exponential smoothing has a "flat" forecast function. That is, all forecasts take the same value, equal to the last level component. Remember that these forecasts will only be suitable if the time series has no trend or seasonal component.

The application of every exponential smoothing method requires the smoothing parameters and the initial values to be chosen. THis is a nonlinear optimization.

6.1.6.10 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.

6.1.6.11 Other

additive/multiplicative decomposition (decompose(type="multiplicative")) not used now because there are better methods.

X11: Another popular method for decomposing quarterly and monthly data is the 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. 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.

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 reseasonalize the forecasts: stlf(elecequip, method='naive') 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()

garch: non-linear

Vector autoregression & quantile regression forests

6.1.7 Model Selection

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 overlayed normal distribution for comparison), and do a Ljung-Box test with the correct degrees of freedom.

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 model as well.

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)

AIC_c for model selection.

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 naive 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.

A great advantage of the ETS statistical framework is that information criteria can be used for model selection - AIC, AIC_c, and BIC

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

we recommend that one of the AICc, 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 model forecasts well (so has low RMSE on the test set) and uses all available information in the training data (so has white noise residuals).

The AICc statistic is useful for selecting between models in the same class. For example, you can use it to select an ETS model or to select an ARIMA model. However, you cannot use it to compare ETS and ARIMA models because they are in different model classes. Instead, you can use time series cross-validation to compare an ARIMA model and an ETS model on the austa data. Because tsCV() requires functions that return forecast objects, you will set up some simple functions that fit the models and return the forecasts.

Adjusting for bias is easy: if the residuals have mean m then add m to all forecasts and the bias problem is solved.

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.

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

- 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.
- The residuals have zero mean. If the residuals have a mean other than zero, then the forecasts are biased.

Recall that r_k is the autocorrelation for lag k. When we look at the ACF plot to see whether each spike is within the required limits, we are implicitly carrying out multiple hypothesis tests, each one with a small probability of giving a false positive. When enough of these tests are done, it is likely that at least one will give a false positive, and so we may conclude that the residuals have some remaining autocorrelation, when in fact they do not. In order to overcome this problem, we test whether the first h autocorrelations are significantly different from what would be expected from a white noise process. A test for a group of autocorrelations is called a portmanteau test, from a French word describing a suitcase containing a number of items. Use Box-Pierce/Ljung-Box test.

All of these methods for checking residuals are conveniently packaged into one R function checkresiduals(), which will produce a time plot, ACF plot and histogram of the residuals (with an overlayed normal distribution for comparison), and do a Ljung-Box test with the correct degrees of freedom.

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.

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.

With time series forecasting, one-step forecasts may not be as relevant as multi-step forecasts. In this case, the cross-validation procedure based on a rolling forecasting origin can be modified to allow multi-step errors to be used.

A prediction interval gives an interval within which we expect y_t to lie with a specified probability.

When forecasting one step ahead, the standard deviation of the forecast distribution is almost the same as the standard deviation of the residuals.

There are three general settings in which judgmental forecasting is used: (i) there are no available data, so that statistical methods are not applicable and judgmental forecasting is the only feasible approach; (ii) data are available, statistical forecasts are generated, and these are then adjusted using judgement; and (iii) data are available and statistical and judgmental forecasts are generated independently and then combined.

Another useful test of autocorrelation in the residuals designed to take account for the regression model is the Breusch-Godfrey test, also referred to as the LM (Lagrange Multiplier) test for serial correlation. It is used to test the joint hypothesis that there is no autocorrelation in the residuals up to a certain specified order. A small p-value indicates there is significant autocorrelation remaining in the residuals. The Breusch-Godfrey test is similar to the Ljung-Box test, but it is specifically designed for use with regression models.

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

We recommend that one of the AICc, AIC, or CV statistics be used, each of which has forecasting as their objective.

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. H0: autocorrelations are all 0. ha: 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.

Model Selection: A great advantage of the ETS statistical framework is that information criteria can be used for model selection - AIC, AIC_c, and BIC

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

6.2 DSP

FFT: time x amplitude to frequency x strength of signal domain

y(t) = Amplitude times sin(2pi times freq times time + phase_shift)

The reduction of a continuous time signal to a discrete time signal is known as sampling

wavelets: continuous (time frequency analysis) vs discrete (data compression & noise reduction).

pipeline: signal -> wavelet transform -> pca -> features to classifier

Chapter 7

Deep Learning

7.1 General

Cheatsheet

Glossary

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.

Activation Functions: Sigmoid: Takes a real-valued input and squashes it to range between 0 and 1. Softmax: Same end result as sigmoid, but different function. tanh: Takes a real-valued input and squashes it to the range [-1, 1]. 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.

Embeddings vs One-Hot Encoding: Embeddings are better than One-Hot Encodings because it allows for relationships to be shown between days.

Rule of 30: A change that affects at least 30 data points in your validation set is usually significant and not just noise.

In a single-label, multiclass classification problem, your network should end with a softmax activation so that it will output a probability distribution over the N output classes.

If you're predicting N categories, don't have layers with less than N nodes.

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes.

Few people use kfold cv in deep learning because of the large datasets in play.

n inputs and k outputs gives you $(n + 1) * k$ parameters.

Do random initialization of weights but set bias to zero. Don't intialize to values that are too large. He initialization works well for networks with ReLU activations.

end to end deep learning: an end-to-end approach as it maps directly the input (x) to the output (y). end-to-end learning works better in practice than multi-task learning, but requires a large amount of data.

restnet/attention > RNN / LSTM

With a reduced the training set a high batch size and high learning rate cause the model to jump around chaotically.

In backpropagation all derivatives applied to same parameters, so they're very correlated. This is bad for stochastic gradient descent. Gradients are either zero (don't remember the past well) or infinity. The latter is fixed by gradient clipping and the former by using LSTMs.

7.2 Pre-processing

Normalization makes the cost function faster to optimize

One common preprocessing step in machine learning is to center and standardize your dataset, meaning that you substract the mean of the whole numpy array from each example, and then divide each example by the standard deviation of the whole numpy array. But for picture datasets, it is simpler and more convenient and works almost as well to just divide every row of the dataset by 255 (the maximum value of a pixel channel).

Scale images: train_set_x = train_set_x_flatten/255.

Common steps for pre-processing a new dataset are: 1) Figure out the dimensions and shapes of the problem (m_train, m_test, num_px, …), 2) Reshape the datasets such that each example is now a vector of size (num_px * num_px * 3, 1), 3) Standardize the data

7.3 Defaults

Regression: loss - mean_squared_error, metric - rmse, activation_function - relu

Classification: loss - categorical_crossentropy, metric - accuracy, activation_function - softmax

7.4 RNN/LSTM

stateful vs stateless LSTMs

One major issue with layer_simple_rnn is that although it should theoretically be able to retain at time t information about inputs seen many time steps before, in practice, such long-term dependencies are impossible to learn. This is due to the vanishing gradient problem, an effect that is similar to what is observed with non-recurrent networks (feed forward networks) that are many layers deep: as you keep adding layers to a network, the network eventually becomes un-trainable.

The default activation function for LSTMs is the hyperbolic tangent (tanh), which outputs values between -1 and 1. This is the preferred range for the time series data: MinMaxScaler(feature_range = (-1, 1)).

Keras' LSTM expects the input as well as the target data to be in a specific shape. The input has to be a 3-d array of size num_samples, num_timesteps, num_features.

Wiring the LSTM hidden states to sets of consecutive outputs of the same length. For example, if we want to produce predictions for 12 months, our LSTM should have a hidden state length of 12.

A loss function is used to optimize a machine learning algorithm. An accuracy metric is used to measure the algorithm's performance (accuracy) in an interpretable way. Loss function applies to a single training example while the cost function applies to many. The loss function computes the error for a single training example; the cost function is the average of the loss functions of the entire training set.

A trick when you want to flatten a matrix X of shape (a,b,c,d) to a matrix X_flatten of shape (b $*$ c $*$ d, a) is to use `X_flatten = X.reshape(X.shape[0], -1).T`

The tanh activation usually works better than sigmoid activation function for hidden units because the mean of its output is closer to zero, and so it centers the data better for the next layer.

Sigmoid outputs a value between 0 and 1 which makes it a very good choice for binary classification. You can classify as 0 if the output is less than 0.5 and classify as 1 if the output is more than 0.5. It can be done with tanh as well but it is less convenient as the output is between -1 and 1.

The deeper layers of a neural network are typically computing more complex features of the input than the earlier layers.

data augmentation for images: rotate and crop

If you have 10,000,000 examples, how would you split the train/dev/test set? 98% train . 1% dev . 1% test. The dev and test set should come from the same

distribution

What is weight decay? A regularization technique (such as L2 regularization) that results in gradient descent shrinking the weights on every iteration. As a rule of thumb, the more training examples you have, the weaker this term should be. The more parameters you have the higher this term should be.

Techniques for reducing variance (reducing overfitting)? Dropout, L2 regularization, Data augmentation

Do a loss vs learning rate plot to pick a good one. It's the key thing to set. Learning rate annealing: decrease learning rate as you reach the minimum.

Training for more epochs will lead to overfitting.

Number of params in logistic regression: k multiplied by d where k is the number of classes and d is the size of the dimensional space.

Large model weights can indicate that model is overfitted

Gradient descent extensions: adagrad, rms prop, adam, mini-batch gradient descent

A dense layer applies a linear transformation to its input

Always use ReLU activation because it doesn't saturate and it converges faster. Use He et al initialization which is square root of two divided by square root of number of inputs.

batch normalization

7.5 Tuning

Optimizers: If batch gradient descent in a deep network is taking excessively long to find a value of the parameters that achieves a small value for the cost function: Try mini-batch gradient descent, Try better random initialization for the weights, Try Adam, Try tuning the learning rate. A variant of this is Stochastic Gradient Descent (SGD), which is equivalent to mini-batch gradient descent where each mini-batch has just 1 example. When the training set is large, SGD can be faster. But the parameters will "oscillate" toward the minimum rather than converge smoothly. In practice, you'll often get faster results if you do not use neither the whole training set, nor only one training example, to perform each update. Mini-batch gradient descent uses an intermediate number of examples for each step. With mini-batch gradient descent, you loop over the mini-batches instead of looping over individual training examples. The difference between gradient descent, mini-batch gradient descent and stochastic gradient descent is the number of examples you use to perform one update step. You have to tune a learning rate hyperparameter alpha. With a well-turned mini-batch size, usually it outperforms either gradient descent or stochastic gradient descent (particularly when the training set is large). Shuffling and

Partitioning are the two steps required to build mini-batches. Powers of two are often chosen to be the mini-batch size, e.g., 16, 32, 64, 128. Mini-batch Gradient descent, Mini-batch Gradient descent with momentum, mini-batch with Adam. Adam usually performs best. Some advantages of Adam include: Relatively low memory requirements (though higher than gradient descent and gradient descent with momentum), usually works well even with little tuning of hyperparameters (except alpha).

early stopping/termination: (stop as soon as performance on validation set drops after a certain number of epochs) and regularization which applies artifical constraints to reduce the number of free parameters while making it difficult to optimize. Use L2 regularization or dropout.

Dropout. Slow down learning with regularization methods like dropout on the recurrent LSTM connections. 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. You should use dropout (randomly eliminate nodes) only in training. Apply dropout both during forward and backward propagation. During training time, divide each dropout layer by keep_prob to keep the same expected value for the activations. For example, if keep_prob is 0.5, then we will on average shut down half the nodes, so the output will be scaled by 0.5 since only the remaining half are contributing to the solution. Dividing by 0.5 is equivalent to multiplying by 2. Hence, the output now has the same expected value. You can check that this works even when keep_prob is other values than 0.5. Also consider inverted dropout. Andrew Ng doesn't like dropout bc it couples optimization with regularization

Inverted dropout: At test time you do not apply dropout (do not randomly eliminate units) and do not keep the 1/keep_prob factor in the calculations used in training.

Layers. Explore additional hierarchical learning capacity by adding more layers and varied numbers of neurons in each layer.

Regularization. Explore how weight regularization, such as L1 and L2, can be used to slow down learning and overfitting of the network on some configurations.

Optimization Algorithm. Explore the use of alternate optimization algorithms, such as classical gradient descent, to see if specific configurations to speed up or slow down learning can lead to benefits.

Loss Function. Explore the use of alternative loss functions to see if these can be used to lift performance.

Features and Timesteps. Explore the use of lag observations as input features and input time steps of the feature to see if their presence as input can improve learning and/or predictive capability of the model.

Larger Batch Size. Explore larger batch sizes than 4, perhaps requiring further manipulation of the size of the training and test datasets. Don't use mini-batches larger than 32. Training with large mini-batches is bad for your test error.

Change Learning Rate: Reducing the learning rate can improve results. Use different learning rates for each layer

Model Capacity (Bias-Variance): If your Neural Network model seems to have high variance try to add regularization & get more training data. For high bias, make a bigger and deeper network. In general, the less training data you have, the worse overfitting will be, and using a small network is one way to mitigate overfitting. Less nodes and hidden layers corresponds to simpler model and vice versa.

Try to add batch normalization or dropout, maybe it will converge better. Try to augment your training data.

ADAGRAD is another optimization option that takes care of some of the hyperparameters for you.

7.6 Debugging

https://karpathy.github.io/2019/04/25/recipe/

1

2

gradient checking

Sigmoid neurons can lead to vanshing gradients at its extremes. ReLU is better and Leaky ReLU is the best to avoid non-activation.

dying relu

Gradient checking verifies closeness between the gradients from backpropagation and the numerical approximation of the gradient (computed using forward propagation). Gradient checking is slow, so we don't run it in every iteration of training. You would usually run it only to make sure your code is correct, then turn it off and use backprop for the actual learning process. In batch GD the cost should be monotone decreasing. In mini-batch it doesn't have to be since it's like training on a different training set every time. It should still be trending downwards. For small training set use (< 2000) use batch GD. Use batches of 64, 128, 256, 512 otherwise which should speed up training. Make sure it fits in meemory. MIni-batch size is another hyperparameter.

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. Change the activation function to solve.

Chapter 8

Probabilistic Programming

8.1 Bayes Rule

old_prob [P(h)/p(~h)] * strength_of_evidence [p(e|h)/p(e|~h)] = new_prob [p(h|e)/p(~h|e)]

Posterior = Likelihood * Prior / Marginal Likelihood

Likelihood P(Data|theta): Probability of the date 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

8.2 Other

http://brunaw.com/slides/SER2019/talk/pres.html#1 https://causal.app/ PyMc3 Quick Intro Prior Guide

PGM Viz

Another option is to report just the Bayes Factor or likelihood ratio, rather than the posterior probability.

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 tocombining 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 likelihoood implies beta for posterior. Update: beta(alpha + successes, beta + 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.

8.3 Doing Bayesian DA

In general, Bayesian analysis of data follows these steps:

1. Identify the data relevant to the research questions. What are the measurement scales of the data? Which data variables are to be predicted, and which data variables are supposed to act as predictors?

2. Define a descriptive model for the relevant data. The mathematical form and its parameters should be meaningful and appropriate to the theoretical purposes of the analysis.

3. Specify a prior distribution on the parameters. The prior must pass muster with the audience of the analysis, such as skeptical scientists.

4. Use Bayesian inference to re-allocate credibility across parameter values. Interpret the posterior distribution with respect to theoretically meaningful issues (assuming that the model is a reasonable description of the data; see next step).

5. Check that the posterior predictions mimic the data with reasonable accuracy (i.e., conduct a "posterior predictive check"). If not, then consider a different descriptive model.

8.4 Statistical Rethinking

bayesian: 1) likelihood: binomial 2) parameters of the likelihood 3) prior for each param

Data are measured and known; parameters are unknown and must be estimated from data.

Compared to MCMC, variational inference tends to be faster and easier to scale to large data

To combat under/overfitting use a regularizing prior or a scoring device like information criteria.

As you'll see once you start using AIC and related measures, it is true that predictor variables that do improve prediction are not always statistically significant. It is also possible for variables that are statistically significant to do nothing useful for prediction.

But the prior on is narrower and is meant to regularize. The prior ~ Normal(0,1) says that, before seeing the data, the machine should be very skeptical of values above 2 and below −2, as a Gaussian prior with a standard deviation of 1 assigns only 5% plausibility to values above and below 2 standard deviations. Because the predictor variable x is standardized, you can interpret this as meaning that a change of 1 standard deviation in x is very unlikely to produce 2 units of change in the outcome.

metrics: dic, waic

Sampling from the posterior now and plotting the model's predictions would help immensely with interpretation. And that's the course I want to encourage and provide code for. In general, it's not safe to interpret interactions without plotting them.

bayesian model: generative model (binomial number of success) + prior (eg. sample of probabilities of success from uniform)

bayesian inference: bayesian model + data

bayesian inference is conditioning on the data to learn the parameter values.

The reason that we call it posterior is because it represents the uncertainty after (that is, posterior to) having included the information in the data.

Bayesian Inference Methods: Sampling, Grid Approximation

8.5 Causality

8.5.1 General

Causality & DL Instrumental variables propensity score matching http://causalinference.gitlab.io/icwsm-tutorial/ http://causalinference.gitlab.io/dowhy/do_why_simple_example.html

8.5.2 Causality edX

Draw a complete DAG if your expert knowledge is insufficient to exclude any possible effect. Knowledge is expressed in the form of missing arrows.

Causal Markov Condition: If two vars share a cause this cause is also on the graph.

An association exists b/w A & Y if having info on A on average better allows us to predict Y.

An association for a -> y exists if the proportion of individuals with y is different given having or not having a.

A -> B -> Y: B is a mediator which blocks the association b/w A & Y when conditioning on it even though A is causal for Y.

Systematic bias: Any assoication between A & Y not causes by an effect of A on Y.

Bias due to a common cause is called confounding. We expect an association b/w two items with a commong cause.

a <- t -> y: t is a commong cause which blocks the association from a to y when conditioned upon.

Common effects (a -> y <- b) don't cause an association but common causes (a <- y -> b) do.

For a -> y <- b, y is called a collider. It blocks association between a and b.

Selection bias: Conditioning on a collider introduces an association between its causes.

Bias stems from The existence of a shared cause of treatment and outcome and conditioning on a common effect of treatment and outcome. (Also conditioning on a mediator.)

Associations: 1) Cause & effect, 2) Common causes, 3) Conditioning on a common effect

Bias introduced by association by chance can be removed by increasing the sample size.

A (parent) -> B (descendant)

D(irectionally)-separation rules: Determine whether paths (which don't need to follow arrows) are blocked or open. D-separation rules:

1. If there are no variables being conditioned on, a path is blocked iff two arrows on the path collide at some variable on the path.
2. Any path with a non-collider that has been conditioned on is blocked.
3. A collider that has been conditioned on doesn't block the path.
4. A collider with a descendant that has been conditioned on doesn't block the path.

The summary of these D-separation rules is that a path is blocked if, and only if, it contains a non-collider that has been conditioned on, or a collider that has not been conditioned on and has no descendant that had been conditioned on.

Two variables are d-separated if all paths between then are blocked

And two variables are marginally or unconditionally independent if they are D-separated without conditioning on all the variables. On the other hand, we say that two variables are conditionally independent, even a set variables L, if they are D-separated after conditioning on the variables L.

Faithfulness: Causal effects in opposite directions that cancel out, i.e a certain medication causes diseases in some people and prevents it in others. Rare.

All paths are blocked variables are not associated.

back door path: a path connecting A and Y without using arrows that leave from A. eg. Given L $\rightarrow$ A $\rightarrow$ Y $\leftarrow$ L, the path A - L - Y is a back door path.

back door path criterion: We can identify the causal effect of A on Y if we have enough data to block all back door paths between them

confounder: A variable that, possibly with other variables, can block back door paths between treatment and outcome.

Change in estimate definition of confounder: A var is a confounder of the effect of a on y if adjusting for it alters the association between a and y

Conventional definition of confounder: L is a confounder of the effect of a on y if L meets 3 conditions: L is associated with a, L is associated with y conditional on a, L is not on a causal pathway from a to y.

surrogate confounder: a descendant of a cofounder.

turn on recruiter linkedin

cofounding if absolute but cofounder is relative to other variables.

If you can't randomize then you have to rely on observational data. Methods: 1) Measure enough variables to block all backdoor paths between A & Y, 2) Alternatives to blocking back door paths (instrumental variable estimation) Here's are a few options for 1):

Stratification (adjust for confounding): A way to adjust for a variable by only looking at a subset of it, for example only looking at people who smoke instead of all people. An alternative is to use regression to look at the effects in each subset and then pool the estimates.

Matching (adjust for confounding): Randomly match someone with a & b to ~a & b and a & ~b to ~a & ~b. Then see if there is an associating in the match subset.

G-methods: The following three methods should be used when time-varying confounders are affected by prior treatment.

Inverse Probability Weighting (adjust for confounding): Compute $1/p(a = 1 \mid b)$ and $1/p(a = 0 \mid b)$, and use these as weights in the association computation. Eliminates backdoor.

Standardization (G-formula)

G-estimation

All these confounding adjustment methods require knowledge of the true causal DAG since you need to know which paths to block.

Randomization, when possible, is the preferred approach to eliminating confounding.

Selection bias under the null is called collider stratification bias. One way it occurs is by conditioning on a collider or the child of a collider.

Case-control study: Outcome based selection design

Selection bias can occur due to eligibility criteria and loss to follow-up. The latter can occur from causes that have significance in the causal graph.

Internal validity: We say that a study has internal validity when the estimated association has a causal interpretation in the study population. That is, there is no selection bias, there is no confounding, et cetera.

External validity: We say that a study has external validity when the estimated association has a causal interpretation in another population. That is, we can transport or generalize the estimated effect to other populations.

In randomized trials, self-selection bias does not affect internal validity, but may affect external validity.

Bias under the null due to self-selection at baseline can happen in a follow up study.

Smart selection, adjustment for selection bias

Use inverse probability weighting to control for the four selection bias dags discussed: {insert dags here}

Competing events

Measurement error: True value isn't equal to measured value.

Measurement bias: When the measure of association between a and y (measured treatment & outcome values) differs from that of a and y.

Types of measurement error:

Type 1, independent non-differential. Type 2, dependent non-differential. Type 3, independent differential. Type 4, dependent differential.

Independent non-differential error is the only type of measurement error that does not create bias under the null.

information bias

Statistical models and validation samples can be used to correct measurement error.

We can't correctly choose methods of measurement error correction without considering the structure of the measurement error.

A causal DAG can't eliminate bias due to confounding. A causal DAG can improve precision. A causal DAG can't improve external validity. A causal DAG can represent sources of bias.

Time-varying treatments

treatment-confounder feedback: In its presence most methods are biased. Use G Methods.

The difference method: Run regression with E[M|A,C] which has the term beta_1 times a and E[Y|A,M,C] with the terms theta_1 times a & theta_2 times m. The direct effect is theta_1 and the indirect effect is beta_1 times theta_1.

8.5.3 Causality Coursera

The fundamental problem of causal inference: We can only observe one potential outcome for each person.

Causal effect: E(y^1) - E(y^0)

Chapter 9

Codebook

Common Scripts

```
def objective(trial):

    #joblib.dump(study, 'study.pkl')

    lr = trial.suggest_uniform('learning_rate', 0.01, 1.0)
    nl = trial.suggest_int('num_leaves', 1, 100)
    md = trial.suggest_int('max_depth', 0, 50)
    mcs = trial.suggest_int('min_child_samples', 0, 50)
    mb = trial.suggest_int('max_bin', 100, 1000)
    ss = trial.suggest_uniform('subsample', 0.01, 1.0)
    ssf = trial.suggest_int('subsample_freq', 0, 10)
    csbt = trial.suggest_uniform('subsample', 0.01, 1.0)
    mcw = trial.suggest_int('min_child_weight', 0, 10)
    ssfb = trial.suggest_int('subsample_for_bin', 100000, 500000)
    rl = trial.suggest_loguniform('reg_lambda', 1e-9, 1000)
    ra = trial.suggest_loguniform('reg_alpha', 1e-9, 1.0)
    spw = trial.suggest_loguniform('scale_pos_weight', 1e-6, 500)
    ne = trial.suggest_int('n_estimators', 50, 100)

    grid = {'learning_rate': lr,
            'num_leaves': nl,
            'max_depth': md,
            'min_child_samples': mcs,
            'max_bin': mb,
            'subsample': ss,
            'subsample_freq': ssf,
            'colsample_bytree': csbt,
            'min_child_weight': mcw,
```

```
            'subsample_for_bin': ssfb,
            'reg_lambda': rl,
            'reg_alpha': ra ,
            'scale_pos_weight': spw,
            'n_estimators': ne}

    mdl.set_params(**grid)

    return(np.mean(sk_ms.cross_val_score(mdl, X_train, y_train, cv = 5)))

study = optuna.create_study()
study.optimize(objective, timeout = 10)
```

9.1 Nested CV

```
mdl = sk_lm.LogisticRegression()
grid = None
mdl_tuned = sk_ms.GridSearchCV(estimator = mdl, param_grid = grid, cv = 5) if grid is n
training_scores = cross_val_score(mdl_tuned, X_train, y_train, cv = 5)
training_scores.mean(), training_scores.std()
```

9.2 Simple Baves

```
# Posterior prob that CTR_A > CTR_B given data is np.mean(A_samples > B_samples). Prob

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)
```

9.3 Unbalanced Classes

```
mdl = svm.SVC(kernel='linear', class_weight={1: 10})
svm.SVC(kernel='linear', class_weight = 'balanced')`
```

9.4 Mixed Effect Regression

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

mdl = smf.mixedlm("y ~ x". df, groups = train['group'])
mdl_fit = mdl.fit()
```

9.5 Regression By Groups

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

mtcars %>% group_by(cyl) %>% split(.$cyl) %>% purrr::map(~ lm(mpg ~ ., data = .))
mtcars %>% group_by(cyl) %>% nest() %>%
mutate(model = map(data, function(x) lm(mpg ~ ., data = x) ))
```

9.6 Symbolic Regression

```
from gplearn import SymbolicRegressor

est_gp = SymbolicRegressor(population_size = 100, generations = 100, stopping_criteria = 0.01, p_

est_gp.fit(X_train, y_train)
preds = est_gp.predict(X_train)
np.sqrt(mean_squared_error(y_train, preds))

graph = pydotplus.graphviz.graph_from_dot_data(est_gp._program.export_graphviz())
Image(graph.create_png())
eqn = str(est_gp._program)
mapping_dict = {'X{}'.format(i): X_train.columns[i] for i in range(X_train.shape[1])}

for old_value, new_value in mapping_dict.items():
  eqn = eqn.replace(old_value, new_value)
```

9.7 Stats By Simulation

Notes on the talk by Jake Vanderplas. I left out bootstrap and cross validation below. For all methods look out for selection bias and make sure you have representative samples. More on the bootstrap: These are simulated confidence intervals that work well for n > 20 as a rule of thumb.

```
# Direct Simulation: Works well with an apriori (generative) model of the world, like

n <- 10^6
h_a <- 22
result <- purrr::map(seq(1, n, 1),
           ~ sum(base::sample(c(0,1), 30, replace = T)) >= h_a) %>%
  unlist() %>% sum()

result/n

# Shuffling: Comparing groups

iris_100 <- iris[1:100, c(2,5)]
(t.test(iris_100[1:50, 1], iris_100[51:100, 1],
       alternative = 'greater'))$p.value

ha_shuffle <- function(df){

  result <- df %>%
  mutate(Species = sample(Species, n())) %>%
  group_by(Species) %>%
  summarise_at(1, mean) %>%
  mutate(mu_diff = Sepal.Width - lag(Sepal.Width)) %>%
  filter(!is.na(mu_diff)) %>%
  pull(mu_diff)

  return(result >= observed_diff)

}

observed_diff <- iris_100 %>%
  group_by(Species) %>%
  summarise_at(1, mean) %>%
  mutate(mu_diff = Sepal.Width - lag(Sepal.Width)) %>%
  filter(!is.na(mu_diff)) %>%
  pull(mu_diff)

n <- 10^6
result <- purrr::map(seq(1, n, 1), ~ ha_shuffle(iris_100)) %>%
```

```
  unlist() %>% sum()

result/n

# He discussed cross validation as a third method
```

9.8 TS Harmonic Regression

```
# Set up harmonic regressors of order 13
harmonics <- fourier(gasoline, K = 13)
# Fit regression model with ARIMA errors
fit <- auto.arima(gasoline, xreg = harmonics, seasonal = FALSE)
# Forecasts next 3 years
fc <- forecast(fit, xreg = fourier(gasoline, K = 13, h = 156))
# Plot forecasts fc
autoplot(fc)
```

9.9 Save And Load Models

```
from sklearn.externals import joblib

joblib.dump(grid_search, 'model.joblib')
grid_search = joblib.load('model.joblib')

model = grid_search.best_estimator_
```

```
save(m1, file = "my_model1.rda")
m1 = load("my_model1.rda")
```

9.10 LSTM

```
# Data Shape
data_x = np.array([
    # Datapoint 1: 3 features for timesteps 1 & 2
    [[1, 2, 3], [4, 5, 6]],
    # Datapoint 2
    [[7, 8, 9], [10, 11, 12]]])

# Prediction Example
mdl.predict([[np.array([.5, .6, .6] + [0 for i in range(14)]).reshape((17, 1))]])[0]
```

9.11 Pairwise Scatterplot

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

9.12 Anti-join

```
# https://stackoverflow.com/questions/38516664/anti-join-pandas

# Identify what values are in TableB and not in TableA
key_diff = set(TableB.Key).difference(TableA.Key)
where_diff = TableB.Key.isin(key_diff)

# Slice TableB accordingly and append to TableA
TableA.append(TableB[where_diff], ignore_index=True)
```

9.13 Df Diff

```
# https://stackoverflow.com/a/36893675/6627726

merged = df1.merge(df2, indicator=True, how='outer')
merged[merged['_merge'] == 'right_only']
```

9.14 Read SQL

```
df <- dbGetQuery(con, statement = read_file('query.sql'))
```

9.15 Extract Date

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

9.16 GGplot To Plotly

```
ggplotly(ggplot(df), aes(x, y, label = z, color = w)) +
        geom_jitter(width = .2), tooltip = c('y', 'label'))
```

9.17 Custom Plotting Theme

```
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")
        )
}
```

9.18 Rowwise

```
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)
```

```
# purrlyr::by_row in R

library(dplyr)

mtcars %>%
  rowwise() %>%
  mutate(mymean = mean(c(cyl,mpg))) %>%
  select(cyl, mpg, mymean)

df %>%
  mutate(mu = select(., R1:R16) %>% pmap_dbl(~mean(c(...))))

calc_row_mean <- function(li){

  df <- as.data.frame(li)

  result <- df %>%
    select(contains("Round")) %>%
    mutate(mu = rowMeans(.)) %>%
```

```
    pull(mu)

  return(result)

}
```

9.19 Sampling File

```
pip install subsample
subsample -s 8 -n 100000 test.csv -r > test_sample.csv
```

9.20 Featuretools

```
# https://stackoverflow.com/questions/50145953/how-to-apply-deep-feature-synthesis-to-

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=

# produces 626 features
fm, features = ft.dfs(
    entityset = es,
    target_entity = 'd',
    agg_primitives = ['mean', 'max', 'percent_true', 'last'],
    trans_primitives = ['subtract', 'divide']
)

# produces no new features
_, features2 = ft.dfs(
    entityset = es,
    target_entity = 'd',
    max_depth = 2
)
```

9.21 SHAP

```
import shap
import matplotlib.pyplot as plt

mdl.fit(X_train, y_train)
explainer = shap.TreeExplainer(clf)
shap_values = explainer.shap_values(X_train)

#all records explained (both lines of code)
shap.summary_plot(shap_values, X_train)
shap.summary_plot(shap_values, X_train, plot_type = "bar")

df_shap = pd.DataFrame(list(zip(np.mean(shap_values, axis = 0), X_train.columns)),
                       columns = ['shap_mean', 'feature'])
df_shap = df_shap[['feature', 'shap_mean']]

df_shap['shap_mean_abs'] = np.absolute(df_shap['shap_mean'])
df_shap.sort_values(['shap_mean_abs'], ascending = False, inplace = True)

attribution_data = np.array(train_data[:50])
explainer = shap.DeepExplainer(model, attribution_data)
```

9.22 CSV To DB

```
# alternative: http://bit.ly/294gmmP
sqlite3 db_name.db
.mode csv db_name
.import data.csv db_name
```

9.23 Bayesian CV

```
from scipy.stats import randint as sp_randint
from skopt import BayesSearchCV
from skopt.space import Real, Integer, Categorical

mdl_tuner = BayesSearchCV(estimator = mdl, search_spaces = grid, scoring = 'roc_auc',
                          cv = folds, random_state = 8)

result = mdl_tuner.fit(X_train.values, y_train.values)
```

9.24 Pyspark

```
# Cheatsheet: https://www.qubole.com/resources/pyspark-cheatsheet/

# Spark only handles numeric data.

# selectExpr() takes SQL expressions as a string.

# Spark's assumes the target is called 'label' in ML. Everything else is a feature.

# Estimator classes are for modeling and all implement a .fit() method. eg. StringIndex

# It's important to split the data after all the transformations because operations li

# Import

import findspark
findspark.init()

import pyspark
from pyspark.sql import SparkSession
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 pyspark.sql.functions as F

sc = pyspark.SparkContext()
spark = SparkSession.builder.getOrCreate() # SparkSession.builder.appName('chosenName')

# Approximate Pi

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

num_samples = 100000000
count = sc.parallelize(range(0, num_samples)).filter(inside).count()
pi = 4 * count / num_samples

print(pi)

df = spark.read.csv('data.csv', header = True)
#df = spark.read.format("csv").option("header","true").option("inferSchema","true").lo
```

```
df.show(5)
df.columns

# Transform

## List tables
spark.catalog.listTables()

# Access and display data
flights10 = spark.sql("FROM flights SELECT * LIMIT 10")
flights10.show()

# Cast
df['g'] = df['g'].astype(str)

# Spark df to pandas and vice versa
spark_temp = spark.createDataFrame(pd_temp) # toPandas()

# Add to the catalog
spark_temp.createOrReplaceTempView("temp")

# 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 expres
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 n

# Groupby
flights.filter(flights.origin == "PDX").groupBy().min("distance").show()
flights.filter(flights.carrier=='DL').filter(flights.origin=='SEA').groupBy().avg('air_time').sho
flights.withColumn("duration_hrs", flights.air_time/60).groupBy().sum('duration_hrs').show()

# Spark functions
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. Me

# 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 = ["carrier_fact", "dest_fact", "plane_age"]

## Import & Make Pipeline
flights_pipe = Pipeline(stages=[dest_indexer, dest_encoder, carr_indexer, carr_encoder

## Fit and transform the data
piped_data = flights_pipe.fit(model_data).transform(model_data)

# Preprocessing

# Mutate
df = df.withColumn("label", df["target"].cast('integer'))
#df = df.withColumn("newCol", df.oldCol + 1)
#model_data = model_data.withColumn("plane_age", model_data.year - model_data.plane_ye

feature_cols = ["some list of column names"]

scf_indexer = StringIndexer(inputCol = "some_cat_feature", outputCol = "some_cat_featu
scf_encoder = OneHotEncoder(inputCol = "some_cat_feature_index", outputCol = "some_cat
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

mdl = LogisticRegression()
evaluator = evals.BinaryClassificationEvaluator(metricName = "areaUnderROC")

grid = tune.ParamGridBuilder()
grid = grid.addGrid(mdl.regParam, np.arange(0, .1, .01))
grid = grid.addGrid(mdl.elasticNetParam, [0, 1])
```

```
grid = grid.build()

cv_results = tune.CrossValidator(estimator = mdl, estimatorParamMaps = grid,
                                 evaluator = evaluator, numFolds = 5)

mdl_best = cv_results.fit(training).bestModel
results = mdl_best.transform(training)

print(evaluator.evaluate(results))

sc.stop()
```

9.25 Sparklyr

```
library(sparklyr)

# feature transforms: ft_, ml functions: ml_, spark df functions: sdf_
# spark: src, ft, ml, sdf, new_ml, spark, stream, compute & collect spark

sc <- spark_connect(master="local")
config <- spark_config()
config$spark.executor.cores <- 8
config$spark.executor.memory <- "25G"

flights <- copy_to(sc, flights, "flights")
src_tbls(sc)

spark_apply(iris_tbl,
            function(e) summary(lm(Petal_Length ~ Petal_Width, e))$r.squared,
            names = "r.squared",
            group_by = "Species")

final <- names %>%
  spark_dataframe() %>%
  sqlfunction(sc, "SELECT * FROM test") %>%
  sdf_register("name5")

## 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(sc, query)
```

```
## General transformation structure and example
df <- ft_some_transformation(df, "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 <- sdf_partition(df, training = 0.7, testing = 0.3)

## gradient boosted trees model Example

gradient_boosted_trees_model <- ml_gradient_boosted_trees(df, '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_t

spark_disconnect(sc)
```

9.26 Data Table

```
# Operations done by reference

names(DT) # colnames
dim(DT) # dimensions

DT[i, j, by]  # subset by i calculate by j grouped using by
DT[.N]  # prints last row
```

```
DT[, .(A, B)] # returns two columns
DT[, c(A, B)] # returns a concatenated vector
DT[, .(sum_c = sum(C)] # mutate
DT[, plot(A, C)] # plot?
DT[, A := NULL] # Remove column A
DT[, .(sumB = sum(B)), by = .(Grp = A%%2)] # group_by & summarize
DT[, .N, by = Sepal.Width] # .N is the count of each group
DT[, lapply(.SD, median)] # .SD is a placeholder for all the columns
DT[, lapply(.SD, mean), .SDcols = 2:3] # Find mean of columns 2 & 3
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 = .EACH]

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)

dt1[dt2, roll=-Inf, rollends=FALSE] # rolling join
```

9.27 Keras

The general framework: 1) instantiate, 2) add layers input-hidden-output, 3) compile, 4) fit

```
network <- keras_model_sequential()

network %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28)) %>%
  layer_dense(units = 10, activation = "softmax") %>%
  compile(optimizer = "rmsprop",  loss = "categorical_crossentropy",  metrics = c("accuracy")) %>
  fit(train_images, train_labels, epochs = 5, batch_size = 128)

metrics <- network %>% evaluate(test_images, test_labels)

#import keras
#from keras.layers import Dense
#from keras.models import Sequential

# Regression

# Specify the model
#model = Sequential()
## Input
```

```
#model.add(Dense(50, activation='relu', input_shape = 3))
# 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))

# Classification

# Specify the model: Two hidden layers
#model = Sequential()
## Input
#model.add(Dense(50, activation='relu', input_shape = 3))
# Hidden
#model.add(Dense(32, activation='relu'))
# Output
#model.add(Dense(2, activation = 'softmax'))
# Compile the model
#model.compile(optimizer = 'sgd', loss = 'categorical_crossentropy', metrics = 'accura
# Fit the model
#model.fit(predictors, target)

# Other

# Dummying
#y = 1
#keras.utils.to_categorical(y, num_classes = 2)

#Look at summary
#model.summary()

#Calculate predictions: predictions
#predictions = model.predict(pred_data)

#Calculate predicted probability of survival
#predicted_prob_true = predictions[:, 1]
```

9.28 Parsnip

```
# parsnip
```

```
predict_cv <- function(split, rec, model) {

  test <- bake(rec, assessment(split))

  result <- tibble(actual = test$map_growth_fw,
                   predicted = predict(model, test))


  return(result)

}

split <- initial_split(df_fw, prop = 3/4)
train <- training(split)
test <- testing(split)

rcp <- recipe(map_growth_fw ~ ., train) %>%
pp <- prep(rcp, training = train, retain = TRUE)
train_pp <- juice(pp)
#test_pp <- bake(pp, newdata = test)

mdl <- set_engine(rand_forest(mode = "regression", trees = 200),
                  "randomForest")

# regular
mdl_fit <- fit(object = mdl,
               formula = formula(pp),
               data = train_pp)

results <- train_pp %>%
  mutate(actual = target,
         preds = pull(predict(mdl_fit, train_pp), .pred)) %>%
  select(actual, preds)

metrics(results, truth = actual, estimate = preds)

#cv

folds <- vfold_cv(train, v = 10)

cv <- folds %>%
  mutate(train = map(splits, prepper, recipe = rcp),
         val = map(splits, analysis),
         mdl_fits = map2(train,
                         val,
```

```
                          ~ fit(mdl,
                                formula(.x),
                                data = bake(object = .x, new_data = .y))))

preds <- mutate(cv,
                pred = pmap(list(splits, train, mdl_fits), predict_cv))

evaluation <- preds %>%
  mutate(metrics = map(pred, ~ metrics(.x, truth = actual, estimate = predicted))) %>%
  select(metrics) %>%
  unnest(metrics) %>%
  summarise_at(vars(accuracy), funs(mean, sd))
```