Data science prep for academics (in vaguely descending order of priority):

* Take/review machine learning course(s)
* Thoroughly review best examples of your previous coding and analysis projects (especially ML) so that you can discuss them in detail
* Prepare responses to common data science interview questions (conceptual and coding; see bottom of this document)
* Consider creating a practice project to have a more recent example of your data science skills or to showcase skills not included in your previous projects (can use publicly available data or other data you have access to)
* Build a profile and portfolio of examples of your coding on GitHub that you are prepared to walkthrough, explain, point out what I would do differently now (just include key chunks of code that you can tell a story about)
* Practice coding through practice projects and coding challenge websites (e.g., <https://www.codewars.com/>; take/review Python courses if needed)
* Choose a representative job ad to help you create templates for you resume and cover letter
* Find guides to help you build a strong LinkedIn profile
* Explore contacts that could help connect you with job opportunities you want through friends, current and former colleagues, LinkedIn, alumni networks
* Learn basics of SQL
* Learn basics of AWS cloud computing
* Score yourself in data scientist rubric to identify strengths and weaknesses (see bottom of this document and <https://towardsdatascience.com/a-framework-for-evaluating-data-scientist-competency-89b5f275a6bf>)
* Study scikit-learn algorithm map, one helpful overview of ML approaches (<https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html>)
* Study basics of probability

Bonus:

* Utilize kaggle.com (can start with their free micro-courses)
* Read relevant articles on towardsdatascience.com
* Learn about / practice with Tableau, Hadoop, and more advanced SQL

Important learning areas:

* machine learning
* statistics
* probability
* computer science
* optimization

High-level ML workflow:

1. Data collection
2. Data processing
3. Model selection
4. Model evaluation

**Kaggle examples**

decent overall outline from a data scientist, but a bit messy, includes model tuning

<https://www.kaggle.com/code/kenjee/titanic-project-example>

random forest with feature importances, overall pretty messy

<https://www.kaggle.com/code/muku007/brain-stroke-binaryclassifier-5-models-gsearchcv>

no model tuning but pretty good otherwise

<https://www.kaggle.com/code/surajjha101/regression-models-diamond-price-prediction>

**Data processing**

Samples/observations/records (rows)

Features/variables/attributes (columns)

Target/outcome/label (dependent variable)

Load data

Examine variables – categorical vs. numeric, data type, missing data, range of values, categorical coding

Visualize distributions of the variables

Scaling or standardizing numeric features can be important for some models, especially models that rely on distance between samples such as k-nearest neighbors so that each feature contributes equally to distance

Also helpful for linear models, helps them to converge on solution faster

Not needed for decision trees because decisions based on only one feature at a time

Encoding of categorical variables

ordinal encoding – for categorical variables with ordinal meaning (categories labeled as 0, 1, 2, 3…), also more efficient for tree-based models even if categories are not actually ordinal (but many possible categories per variable)

one-hot encoding – dummy coding, uses sparse matrix by default, use for linear models

Feature engineering, sometimes term used more specifically, sometimes more broadly, but often refers to generating new features to help model more effectively predict outcomes, such as transforming non-linear features such that they can be used in linear models (polynomial feature expansion, MinMaxScaler may work better than StandardScaler before PolynomialFeatures), can become relevant when model underfits, i.e., somethings needs to be done to the data to unlock its explanatory power

Missing data and imputation

MCAR – missing completely at random, missingness is independent of the data, there’s nothing systematic/predictable about why the missing values are missing, e.g., faulty scale resulting in missing weight values, but when the scale is faulty is unrelated to the weight of the participant or any other observed variables, can test for MCAR, dummy code missing vs. non-missing values for variable of interest and use t tests for each other continuous variable and chi-square tests of independence for each categorical variable to test for significant relationship (if significant relationship is observed, then data are not MCAR), related alternative is Little’s MCAR Test but it is mainly suited for small percentages of missing data and low-dimensional data sets, both test approaches have significant limitations, can use deletion or imputation with MCAR, imputation most common in practice with machine learning, deletion is reasonable if the percentage of missing data is very small

MAR – missing at random, missingness depends only on the observed data, there is a systematic cause for the missing values, but it’s unrelated to the values themselves, it’s related to other observed variables, e.g., more missing weight values in women because they are more likely to omit reporting their weight, missingness is related to gender but not to the weight values themselves, can test for MCAR (theoretically) and possibly proceed with MAR assumption (with expert discretion) if MCAR fails, can use model-based imputation (sklearn IterativeImputer, KNNImputer)

MNAR – missing not at random, missingness depends on the unobserved data, there is a systematic cause for the missing values and it’s related to the values themselves, e.g., very low weights result in more scale errors and missing weight values, MNAR cannot be easily tested, avoid using data with MNAR if possible or be very clear about likely bias in results

Structurally missing – value is missing in actuality, as opposed to just being unrecorded in the data set, i.e., there is no value to be recovered, e.g., missing high school GPA for a candidate because their high school did not have a GPA system

Some options for dealing with missing data (listwise deletion methods tend to apply to missing data in target variables, while imputation is used for missing data in features where listwise deletion may not be necessary):

* Listwise deletion of samples with missing data from full data set, can be an elegant, simple solution if number of samples with missing data is very small and/or missing data is from critical variables
* Listwise deletion by models utilizing the variable, can retain more data if the variable is not relevant for all models
* Simple imputation, helpful for filling in missing values with a reasonable constant (e.g., zero, mean, etc.)
* Advanced imputation methods, predict missing values from the remaining features, regression-based and k-nearest neighbors approaches, one comparison has shown a slight advantage for k-nearest neighbors if data set is not too large

<https://scikit-learn.org/stable/modules/impute.html>

<https://www.ksk-anl.com/blog/dealing-with-missing-data/>

Need to take care to avoid data leakage (model performs better in training and testing than it would once deployed in practice):

Target leakage – model uses features in training and testing that would not be available when the model is deployed

Train-test contamination – model performance on test data is optimistically biased by fitting certain preprocessing steps on the full train+test data set rather than on the training data alone, for example, imputation fitting should be done on the training data only and then that transform applied to both the training and testing data, including appropriate data preprocessing steps in sklearn pipelines can help to avoid these problems as they are meant to appropriately apply fit and transform steps

**Machine learning models**

Machine learning – extracting rules from existing data that will generalize to predict new data

Supervised learning – uses examples where features are paired with outcomes, labeled training data, used to predict outcomes, often used with very large and categorical data sets

Regression - continuous dependent/outcome variable

**Linear regression** (in practice, use ridge regression [employs regularization] and tune alpha, ridge regression also helps to avoid strange results from multicollinearity in features, can also use lasso regression if you have many predictors and only expect a small proportion to be relevant)

**Multiple linear**

**Polynomial**

**Decision tree** – series of decisions (nodes) based on one feature at a time to minimize prediction error, non-parametric (no assumption about data distribution), yields piecewise constant value predictions, simple decision trees are almost never used in practice (piecewise constant predictions too coarse) but they are the building blocks of gradient-boosting decision tree and random forest models (ensemble models), which are often the best performing algorithms for tabular data sets, decision/split nodes (top decision node called root node, best predictor) and leaf/terminal nodes (predictions based on decisions, do not split), decision trees usually very poor at extrapolating outside training range

**AdaBoost** (adaptive boosting; ensemble) – iteratively combine underfitted, weak learners, re-weighting the training data for prediction errors after each iteration, see description below, HistGradBoost generally recommended over AdaBoost now

**Gradient-boosting decision tree** (ensemble) – add weak learning trees sequentially to minimize some loss function with gradient descent, each tree predicts the negative errors of previous ensemble, when all base predictions are summed then prediction errors cancel out, usually outperforms random forest, trees stop being added when limit reached, loss reaches acceptable level, or new tree no longer improves predictive performance, can overfit, so can benefit from regularization methods, exact GradientBoosting method too slow for samples > 10,000, HistGradBoost is faster and should be default method for larger data sets (numerical features are binned into histograms rather than preserving each individual value), recommended max depth of 3-8, or max leaf nodes of 8-256, tradeoff between # estimators and depth

**Random forest** (ensemble) – ensemble method involve bootstrapping deep decision trees, special modification of bagging with decision trees that also involves using a random subset of features for each decision node of each tree to decorrelate prediction errors across base models (makes aggregating work more accurately)

**Neural network** – can increase capacity (complexity, learning power) by making model wider (add more units to existing layers) or deeper (add more layers, i.e., deep learning), wider networks tend to be better at learning linear relationships, while deeper networks can better learn non-linear relationships, Keras is go-to library for deep learning (part of TensorFlow package), can apply activation functions to the output from layers, common example is rectifier function that enables fitting non-linear relationships (rectified linear unit or ReLU), stochastic gradient descent describes the optimization algorithms used in deep learning (sample training data and run through network to make prediction, measure loss, adjust weights to decrease loss), Adam is good standard choice for SGD algorithm because it is adaptive (self-tuning), can apply dropout to make it harder for network to learn noise, percentage of layer units dropped each step of optimization, spurious patterns in training data more likely to drop out, like an ensemble approach, batch normalization layers can also stabilize model training and reduce time to optimization

Classification – discrete dependent/outcome variable

**Logistic regression** – regularized by default in scikit-learn (C hyperparameter)

**K-nearest neighbors** – determine the majority target label of the k closest samples from the training data to a given sample and classify the given sample with that majority target label (intuitive algorithm but rarely useful in practice)

**Support vector machine** – find hyperplane to classify, in simple form can function similarly to logistic regression but uses different method for optimization, can be made more flexible through a kernel that makes it non-linear, weights assigned to samples (as opposed to just features in standard linear models), kernel methods such as SVC are very efficient for small-to-medium data sets (<10,000 samples), polynomial feature expansion etc. more computationally efficient for larger data sets

**Naïve Bayes** – based on conditional probability, suited for complex inputs and very large data sets

**Decision tree**

**Gradient-boosting trees** (ensemble)

**AdaBoost** (ensemble)

**Random forest** (ensemble)

**Neural network**

Note accuracy cannot be used as the loss function for optimizing classification because it is discrete rather than continuous, instead cross-entropy is used (based on classification probabilities)

Linear models provide a good baseline, they are generally fast to train and easy to understand, but often helpful to then check whether more complex models (such as ensemble of decision trees, gradient-boosting trees, kernel machines) can improve performance (if linear models not giving adequate performance), linear models tend to work very well when the number of features is large, may underfit with a very small number of features relative to samples

Inductive bias – the choice of model family/algorithm will favor certain model shapes

Regularization, changing parameters of the model to favor a simpler model function (reducing risk of overfitting), most popular way to introduce regularization to simple linear regression is ridge regression, pulls coefficients towards zero (controlled by alpha), logistic regression is regularized by default in scikit-learn

* L1, also known as Lasso in regression, shrinks some parameters (weights/coefficients) to zero. This functions as a form of feature selection, as features with weights near or equal to zero won’t contribute to prediction. L1 solution can be referred to as sparse, given that only a selection of features will have non-zero weights. For highly correlated features, L1 will tend to just select one of them, warranting caution in interpretation. L1 regularization is more robust to outliers.
* L2, also known as Ridge in regression, shrinks parameters, but not all the way to zero. L2 gives non-sparse solution. L2 is sensitive (not robust) to outliers.

Bagging, bootstrap aggregating, aggregating many different overfitted base models on bootstrapped samples, reduces overfitting, bagging is a general strategy that can be used with any base machine learning algorithm

Bootstrapping, repeated random sampling of data set with replacement

Adaptive boosting (basic boosting), underfit numerous weak learners on full training data set, base models are fit sequentially and each updates the ensemble prediction before the next weak learner is fit, larger prediction errors from the ensemble are given more weight for the next weak learner so that it will focus on refining problem areas for the ensemble, AdaBoost can be used with any machine learning base algorithms that support sample weighting

Gradient boosting is a faster and more powerful application of boosting when the base model algorithm is a decision tree, mechanism not as intuitive as AdaBoost but histogram gradient boosting almost always recommended over AdaBoost now

Out of ensemble methods, gradient boosting tends to perform slightly better and is more computationally efficient than bagging and random forest methods

Unsupervised learning – find patterns of input data without reference to labeled outcomes, unlabeled training data, used to find structure in data, often used with cluttered data

Clustering

**K-means**

**Hierarchical**

**Mean shift**

**Density-based**

Dimensionality reduction

**Principal component analysis**

Reinforcement learning – learning from mistakes and experiences, no starting training data, system programmed to perform actions, which will be punished or rewarded by environment, system learns from experience to make more informed decisions/actions (generally maximizing reward as operationalized in some way)

Semi-supervised learning – partially labeled training data

**Model tuning**

Can use grid search or randomized search to search for optimal values for hyperparameters. Randomized search is typically more efficient relative to grid search when optimizing 3 or more hyperparameters. In practice, would use many iterations for randomized search (maybe 500+ for 5+ hyperparameters). Important to use nested cross-validation for tuning hyperparameters. Inner cross-validation determines hyperparameter values; outer cross-validation tests generalization performance of best hyperparameter values.

**Evaluation**

Splitting data set into training and testing sets allows for evaluation of generalization performance, or predictive performance on unseen data

sklearn.model\_selection.train\_test\_split – single split, tell it what percentage to use, default is last 25% test without shuffling

Cross-validation – repeating training and testing with different splits of the data set to better evaluate predictive performance and evaluate its variability, sometimes thought about as estimating the predictive performance if model were trained on the full data set, also used for hyperparameter tuning, but can’t use the same cross-validation for both or might underestimate overfitting from hyperparameter tuning, instead needs to be nested

sklearn.model\_selection.KFold – specify K splits, default 5 without shuffling, every sample used in a test set exactly once

sklearn.model\_selection.ShuffleSplit – shuffles data before each split, specify number of splits (default = 10) and size of split (default = 10%)

Validation curve (or model complexity curve) – training and testing performance of a model across a set of values for a hyperparameter, useful for evaluating model fit (look for minimum error in testing performance)

Learning curve – training and testing performance of model for varying numbers of training samples, useful for evaluating the capacity for a model’s performance to improve with additional training samples

Helpful to compare performance to a baseline model, such as sklearn.dummy.DummyClassifier (strategy = “most\_frequent”) or sklearn.dummy.DummyRegressor (strategy = “mean”; ignores features and just predicts target mean from training set, can also use median if extreme outliers)

Linear models can also serve as a more intelligent baseline to more complex models

**Evaluation metrics**

For regression:

default score is R^2

sklearn.metrics.mean\_squared\_error – what linear models optimize

sklearn.metrics.mean\_absolute\_error – more intuitive than mean squared error because gives mean error in units of target variable, also a median option to reduce the influence of outlier errors

When it makes more sense to get error that scales with the data, can use mean absolute percentage error

For classification:

sklearn.metrics.ConfusionMatrixDisplay – gives true +, true -, false +, false –

precision (or positive predictive value) = TP / (TP + FP)

sensitivity (or recall, hit rate, true positive rate) = TP / (TP + FN)

specificity (or selectivity, true negative rate) = TN (TN + FP)

accuracy = (TP + TN) / total, accuracy generally not as meaningful of a metric with imbalanced classes, can use balanced accuracy, which is the average sensitivity/recall across classes, or use precision and sensitivity/recall

sklearn.metrics.PrecisionRecallDisplay – displays precision-recall curve across classification decision thresholds (e.g., default 0.5 decision threshold probability for binary classification, but alternative threshold can lead to better performance, particularly in case of imbalanced classes?), AUC can be used as metric focusing on performance with positive class (range 0(?)-1)

sklearn.metrics.RocCurveDisplay – can use ROC curve with sensitivity and specificity to evaluate compromise of accurately predicting positive and negative classes (AUC range 0.5-1)

Bayes error rate or irreducible error – the error of the best model trained on unlimited data, essentially error due to true variance in signal (not noise) that is not captured by the features in the model

ML Explainability:

Feature importance – which features have the largest impact on predictions

Permutation importance – one feature at a time is randomly shuffled and then loss in prediction performance is measured (user can choose performance metric), this is repeated for each feature and then the full process is repeated to stabilize the estimates over random shufflings (default 5 iterations in eli5), giving relative importance of all features, this can be done by shuffling features in the training data (or more simply removing them) and re-training the model each time but this is computationally intensive and reveals more about importance of features in the data set, the preferred solution is to perform this process in the test data with a model that has already been trained (can use permutation\_importance in eli5 library), this is faster and reveals importance of features for a specific model, however even this implementation can get computationally intensive if there is a very large number of features (and large data set), another caution is that this method assumes independent features, feature importance could be misleadingly small for highly correlated features, also sensitive to outliers, i.e., a medium permutation importance could reflect a large effect for a few predictions but little effect for most or a medium effect for all predictions

Shapley additive explanation (SHAP) values – at its core, this method generates feature importances for an individual case prediction, but it can be scaled up to aggregate across many or all test cases, computes difference in prediction for each feature between true feature value and baseline (e.g., mean) feature value (does this in a more sophisticated way than PDP to reduce problem of improbable/impossible comparison cases), the sum of all SHAP values equals the prediction relative to the baseline prediction (average prediction of the model), can then average absolute values of SHAP values for all test cases to examine overall feature importance, shap library has different implementations for different model families, can visualize results with a force plot, can summarize results for many cases with a SHAP summary plot (but graph can’t handle too many cases; spread of points can be a reasonable indicator of feature importance), can also summarize with stacked force plot but this is much more difficult to read, computing many SHAP values is intensive and can be slow, although there are optimizations to speed up computations for xgboost models, can also focus on specific features with SHAP dependence (contribution) plots, similar to PDP but contain more information, again can only display a limited number of cases before graph breaks, with SHAP value aggregations caution is warranted to not overinterpret outliers, especially if only a random subset of test cases is shown

Partial dependence plots – again starts with an assumption of feature independence, shows how model prediction changes on average over value range of a given feature, 2D PDP also possible to examine interaction of 2 features (library pdpbox), need to interpret carefully as the plot can show results based on improbably or impossible data (combinations of feature values that don’t make sense in reality)

**Questions**

How many years of applied research experience do you have?

* person-specific

What is your quantitative analysis experience?

* person-specific

What statistical software package(s) are you familiar with?

* person-specific

What is your experience working with large data sets, or combining data from multiple sources? Can you provide an example?

* person-specific

What training have you completed in CS, coding, and stats?

* person-specific

What is overfitting? How should you deal with it?

* Learning power of model is too high, in other words, the number of samples in the training data is too small compared to the flexibility/complexity of the model, such that the model is learning too much from the noise in the data, model fits training data well but performs badly on new data, testing error is much larger than training error, can also be thought of in terms of high variance, with different training data sets the models may average near the true signal but each is sensitive to the specific training set and so will make unique (test) errors, unstable
* Tune model hyperparameters to reduce learning power, add regularization, increase size of training data set, feature selection to reduce features

What is underfitting? How should you deal with it?

* Happens when the model is too simple to capture the shape/structure of the training data, the model does not have enough flexibility to accurately capture the structure of the training data, training and testing error will both be large (but this can also happen due to very noisy data), can also be thought of in terms of bias, even with different training data sets the models will systematically make similar prediction errors
* Tune model hyperparameters or select a different model to increase learning power/flexibility/complexity

What is an imbalanced data set? How should you deal with it?

* The target class is unequally distributed across the training data set. So, if you’re trying to predict disease risk level, and your data set contains 4x as many low-risk samples vs. high-risk samples, you have an imbalanced data set. This can pose a challenge for predictive modeling as many models implicitly assume balanced classes.
* It may be important to use a stratified sampling procedure for cross-validation to avoid overly optimistic estimates of model performance simply due to bias toward the majority class. Imbalanced data sets can also call for careful consideration of which evaluation metrics to use based on relative importance of true +, true -, false +, and false – for the specific application. Decision trees or penalized models can also be used to better address minority classes. Also, target variable can be reweighted to reduce problems related to imbalance.

Describe the random forest classifier? What are its pros and cons?

* The random forest classifier uses an ensemble of decision tree learners. Each individual decision tree is designed to overfit the training data, but when the predictions of all the decision trees are aggregated together, the overfitting of each individual tree will be negated and the prediction of the whole ensemble will generalize much better to new data. Random forest classifiers are defined by two different randomization processes. First, bootstrapping is used to generate a random variation of the training data set for each decision tree to learn. Second, for each decision node in each tree, only a random subset of features is used to make the decision.
* Random forest classifiers are powerful models that tend to have very good generalization performance. They are generally resistant to underfitting and overfitting, and because they are based on decision trees, they don’t require as much data preprocessing. They are computationally expensive relative to some other top-performing algorithms, but each base decision tree is fit independently, so parallel computing can be used to speed up the overall model estimation.

List 3 evaluation metrics and describe the advantages and disadvantages of each.

* mean absolute error, for regression models, can evaluate average error in the same units as the target variable, so its very intuitive relative to other metrics, but this metric doesn’t scale with the target variable
* mean absolute percentage error, if you want a measure of error that will scale with the target variable, this metric will do that, but it may be a little less intuitive on its own because it is not in the original units of the target variable, also a little further from the loss function that a linear model, for example, is actually optimized on, which is generally the mean squared error
* accuracy, a very intuitive metric for evaluating classification performance, what percentage of samples does the model classify correctly, but it doesn’t include information on the types of successes and errors (TP, FP, TN, FN)

When should you use L1 vs. L2 regularization?

* L1, also known as Lasso in regression, shrinks some parameters (weights/coefficients) to zero, functioning as a form of feature selection, reducing the need for separate feature selection steps, but when you have more features than samples/observations, L1 might not be able to retain all meaningful features. When you have highly correlated features, L1 will tend to just select one of them, warranting caution in interpretation. But L1 regularization is more robust to outliers.
* L2, also known as Ridge in regression, shrinks parameters, but not all the way to zero. L2 regularization is sensitive (not robust) to outliers, but will maintain all features and is more computationally efficient.

What are hyperparameters and how do you tune model hyperparameters?

* Hyperparameters are configuration variables external to a model that control how it learns. These are generally specified by the user. Hyperparameters can be distinguished from internal model parameters that are learned from the data through model estimation.
* Hyperparameters can be tuned through semi-automated grid searches or randomized searches for optimal combinations of hyperparameters for a given data set/problem. Optimal hyperparameters vary by data set/problem, so they are not known in advance and must be determined for each application. Hyperparameter tuning often occurs in an inner cross-validation level within a nested cross-validation pipeline, so that the data used to validate the model hyperparameters is separate from the data used to evaluate model generalization performance using those hyperparameters.

What are boosting and gradient boosting models?

* Boosting refers to an ensemble model method, in which many weak, underfitted models are fitted to a training data set sequentially, such that each new model focuses on correcting prediction errors from the existing ensemble. Aggregating the predictions from all the individual underfitted models leads to an ensemble prediction that is no longer underfitted.
* Gradient boosting is a variation on the boosting method in which instead of each subsequent model being focused on correcting prediction errors by weighting poorly classified data points more heavily, subsequent models actually predict the residuals of the previous prediction. Generally a little better than simple boosting models. Histogram gradient boosting is much more efficient still for data sets with >10,000 samples.

What is big O notation?

* It’s a way to express the relative time complexity of different algorithms. It expresses how the computational load of an algorithm scales with the size of the input to that algorithm. Examples, O(1) is constant time, algorithm requires same number of computational steps regardless of input size; O(n) is linear time, linear relationship between input size and computational steps; O(n^2) is big O squared or quadratic time, often arising from nested loops. Earlier solutions or stopping may be possible; big 0 refers to upper bound of computational complexity.

What is precision?

* Precision refers to the proportion of true positives out of all positive predictions from a classification model, in other words, what proportion of a model’s positive predictions are correct.

Experience with A/B testing?

* person-specific

Walk through example project, first at high level then technical details.

* person-specific

Talk about your experience with problem solving.

* person-specific

Talk about your experience drawing conclusions from data to recommend actions.

* person-specific

Be ready to write example code for:

Supervised learning:

Decision tree

Linear regression

Logistic regression

K-nearest neighbors

Unsupervised:

K-means clustering

Framework for thinking about data science competencies/skills from Schaun Wheeler

|  |  |
| --- | --- |
| **Competency** | **Skill** |
| Design analyses | Explicitly plan analysis |
|  | Anticipate and address competing explanations |
|  | Determine the best way to evaluate results |
| Conduct analyses | Explore data appropriately |
|  | Apply appropriate algorithms |
|  | Clearly document findings |
| Incorporate analyses into pipeline | Read/write data to/from any format and location |
|  | Incorporate complex matching and filtering |
|  | Make work compatible with engineering stack |
| Incorporate pipelines into the business | Discover needs of the business |
|  | Navigate business’s organizational structure |
|  | Package technical work for diverse audiences |
| Build the profession | Contribute externally |
|  | Lead the team (mentoring/training team members) |
|  | Draft policy and procedure |