Table of Contents

[EDA 4](#_Toc88140734)

[Categorical 4](#_Toc88140735)

[Continuous 4](#_Toc88140736)

[Preprocessing 5](#_Toc88140737)

[Imputation 5](#_Toc88140738)

[Continuous Features 5](#_Toc88140739)

[Categorical Features 6](#_Toc88140740)

[Text Data 6](#_Toc88140741)

[Dimensionality Reduction 8](#_Toc88140742)

[PCA (Unsupervised) 8](#_Toc88140743)

[Whitening (Unsupervised) 9](#_Toc88140744)

[LDA 9](#_Toc88140745)

[Manifold Learning (t-SNE Unsupervised) 9](#_Toc88140746)

[Feature Extraction 10](#_Toc88140747)

[Polynomial Features 10](#_Toc88140748)

[Clustering (Unsupervised) 10](#_Toc88140749)

[Train-Test Split / Cross Validation / Column Transformers / Pipelines 12](#_Toc88140750)

[Regression Models 14](#_Toc88140751)

[Linear Regression Assumptions: 14](#_Toc88140752)

[Law of Large Numbers 14](#_Toc88140753)

[Central Limit Theorem 14](#_Toc88140754)

[Linear Regression Assumptions 15](#_Toc88140755)

[Linear Regression 15](#_Toc88140756)

[Ridge Regression 15](#_Toc88140757)

[Lasso Regression 16](#_Toc88140758)

[L1 vs L2 Penalties 16](#_Toc88140759)

[LinearSVR / SVR (Support Vector Machines) 18](#_Toc88140760)

[Trees 20](#_Toc88140761)

[Random Forest 20](#_Toc88140762)

[Gradient Boosting 21](#_Toc88140763)

[Classification Models 23](#_Toc88140764)

[Logistic Regression 23](#_Toc88140765)

[LinearSVC/SVC (Support Vector Machines) 24](#_Toc88140766)

[Multiclass Classification 26](#_Toc88140767)

[Trees 26](#_Toc88140768)

[Random Forest 27](#_Toc88140769)

[Gradient Boosting 28](#_Toc88140770)

[Model Evaluation 29](#_Toc88140771)

[Regression 29](#_Toc88140772)

[Binary Classification 30](#_Toc88140773)

[Precision-Recall Curve 32](#_Toc88140774)

[ROC 32](#_Toc88140775)

[Probability Calibration and Imbalanced Data 33](#_Toc88140776)

[Adjusting Probabilities 33](#_Toc88140777)

[Adjusting Classifier 34](#_Toc88140778)

[Model Interpretation & Feature Selection 35](#_Toc88140779)

[Model Interpretations 35](#_Toc88140780)

[Feature Selection 35](#_Toc88140781)

[Parameter Tuning and AutoML 37](#_Toc88140782)

[Grid Search 37](#_Toc88140783)

[Random Search 37](#_Toc88140784)

[Bayesian Optimization (SMBO, state of the art optimization) 37](#_Toc88140785)

[Multi-Fidelity Search 38](#_Toc88140786)

[Successive Halving 38](#_Toc88140787)

[Neural Networks 39](#_Toc88140788)

[Keras Neural Nets 40](#_Toc88140789)

[Convolution neural nets 42](#_Toc88140790)

[Batch Normalization 44](#_Toc88140791)

[Residual Neural Nets 44](#_Toc88140792)

[Recurrent Neural Networks 45](#_Toc88140793)

[Time Series 46](#_Toc88140794)

[Preprocessing 46](#_Toc88140795)

[Stationary Time Series Assumptions: 46](#_Toc88140796)

[Autocorrelation 46](#_Toc88140797)

[Autoregressive (linear) model 47](#_Toc88140798)

[ARIMA 48](#_Toc88140799)

[FBProphet 49](#_Toc88140800)

[Other Libraries 50](#_Toc88140801)

# EDA

## Categorical

* Inspect class frequencies and note data imbalance – sns.countplot()
* If the column has primarily one class it won’t contribute to the model solution, so most likely can drop it
* Bar plot – sns.barplot() calculate mean/sum/etc. with respect to categorical columns
* Box plot – sns.boxplot() with respect to target by each class

## Continuous

1. Box Plot – sns.boxplot should provide insight into distribution and outliers
2. Test for normality – scipy.stats.normaltest runs a test to determine if distribution is normal. P < 0.05 not normal distribution
3. Q-Q plot – statsmodels.api.qqplot(line=’q’, fit=True) will plot theoretical quantiles vs actuals. Straight diagonal line is normal distribution
4. Histogram – pd.Series.hist() or sns.countplot() will plot a histogram of the series
5. Inspect outliers and determine if need to remove them
6. Correlation matrix – pd.DataFrame().corr() will calculate Spearman correlation between continuous columns. Plot with sns.heatmap()
7. Pair plot – scatter between variables and histogram on diagonal sns.pairplot().

# Preprocessing

## Imputation

* Rows with missing target - drop
* A lot of missing data – if majority of the data is missing from feature, might drop it
* Categorical can either fill in with a separate category (‘NA’) or fill in with most frequent category
* Mean / Median – fills in mean and median of column for missing values. Make sure to separate train/test sets, fit on train and transform after fitted on train.
* kNN – model driven imputation KNNImputer
* Regression models – introduces another layer of uncertainty

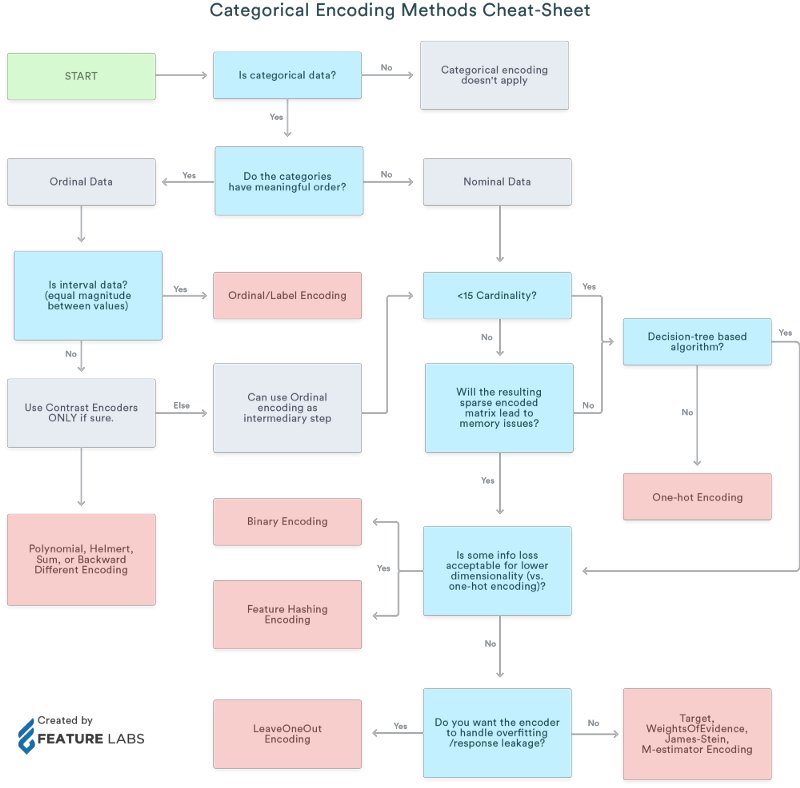
## Continuous Features

Chart, diagram, scatter chart

Description automatically generated

* Standard – most common, normalized data
* MaxAbsScaler – similar to Standard but standardizes by max value instead of mean
* MinMaxScaler – scales [0,1], utilized a lot on SVM
* RobustScaler – uses quartiles and not std dev, more robust to outliers than Standard
* Normalizer – projects to sphere with radius 1, not as common.

## Categorical Features



* Only scale if algorithm is sensitive to distances.
* Target and LeaveOneOut should be scaled for sensitive algorithms
* OneHotEncoder should not be scaled because it’s already binary (1,0)

## Text Data

* CountVectorizer – creates a dictionary of all words in column and places 1 for each word in each cell to create a feature vector. Similar to OHE but for words.

Timeline

Description automatically generated with medium confidence

* TfidfVectorizer – normalized CountVectorizer by frequency of words. Higher freq words are penalized. By default, it’s L2 normalized, divide each row by its length.

Topic Models – combine words from CountVectrozer by topics with feature reduction.

* Matrix Factorization (LSA, Latent Semantic Analysis)

*LSA is basically just the same as PCA. The idea is to reduce the dimensionality of the data to some semantically meaningful components. The thing is, we can't easily do PCA here, because we can’t remove the mean. Remember, this is a sparse dataset and so the zero is sort of meaningful and if we shift the zero, like by trying to subtract the mean, we will get a density dataset that won't have many zeros anymore and so we won’t be able to even store it.*

*Process*

1. Use CountVectorizer or TfidVectorizer to convert to dictionaries
2. Scale – MaxAbsScaler same as STD but doesn’t subtract mean. Or use TfidVectorizer and no need to scale since it normalizes everything to document length.
3. Use TruncatedSVD to perform PCA on sparse matrix of CountVectorizer. *TruncatedSVD.explaned\_variance\_ratio shows how much variance is captured with each additional component.*
4. By reducing to fewer dimensions from dictionaries we ‘group’ by topic

* NMF - Similar to PCA but limits to zeros and positive values
* LDA (Latent Dirichlet Allocation) – most commonly used topic model, probabilistic model.

# Dimensionality Reduction

## PCA (Unsupervised)

*Rotates data around max variance axis and finds another orthogonal max variance axis. Transformation would decorrelate the data and reduce the number of inputs to PCA desired number of variables. Finds linear combination of predictors that maximize variability, it’ll be naturally drawn to features that have more variations.* ***Since PCA relies on distances, need to scale before applying PCA!***

Graphical user interface, text, application

Description automatically generated

A picture containing chart

Description automatically generated

*Can inspect variance captured plot to estimate how many features to derive*

Chart, line chart

Description automatically generated

## Whitening (Unsupervised)

*Compute PCA and then rescale (PCA 🡪 StandardScaler)*

## LDA

*Supervised alternative to PCA Linear Discriminant Analysis. Probabilistic model with Gaussian distribution assumption.*

## Manifold Learning (t-SNE Unsupervised)

*Mainly used for visualization and reduces to 2 dimensions. Resulting axis don’t respond to anything in real world and it’s hard to interpret. T-SNE sometimes used to visualize layers of neural networks.*

# Feature Extraction

## Polynomial Features

*Expands continuous features with polynomial combinations between features. For example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are [1, a, b, a^2, ab, b^2].*

## Clustering (Unsupervised)

*Unsupervised learning algorithms that group the data together. Can be used to extract categorical features by creating a category of cluster number or numerical features that correspond to distances to each group centroid.*

* K-Means – need to scale inputs

*The algorithm proceeds like this: you pick the number of clusters k, then you randomly pick k data points from the data, and declare those as cluster centers. Then, you label each data point according to which cluster center it is closest to. Finally, you recompute the cluster centers as the means of the clusters, and you iterate. The algorithm always converges, which is when the assignment of the points to the clusters doesn't change any more.*

* Agglomerative Clustering – need to scale inputs

*In agglomerative clustering, the idea is that you start out with a bunch of points again. You assign each point its own cluster label, and then you greedily merge clusters that are similar. So here, each of the points is its own cluster, then, in step one, I'm merge the two closest points. Step two, I merged the next closest one, and so on. I can keep merging and merging until one big group is left. Here, I merged until there's three clusters left.*

* DBSCAN – need to scale inputs

*The way this algorithm works is, it picks a point at random. Let's say, we pick point A at random, then it looks whether this is a core point or not, by looking at the number of neighbors with an epsilon, you see three neighbors here with an epsilon and hence it's a core point and so that means I give it a new cluster label, in this case, it’s red. Then I add all of these neighbors to a queue and then I process them all in turn. So I go to this red point, check is it a core sample and not a core sample and again, it has four neighbors, so it's a core point. And I added all the neighbors to the queue. And so you keep doing that.*

Diagram

Description automatically generated

* Mixture Model

*Mixture models are unsupervised models that try to fit per metric density to your data. So you want to model like for p(X), you want to model a distribution of your data. Mixture model does this as a mixture of some based distribution.*

# Train-Test Split / Cross Validation / Column Transformers / Pipelines

Train\_test\_split – Splits data to train and test sets, set stratify = y for categorization.

Cross\_val\_score / cross\_val – By default 5-fold CV for continuous and StratifiedKFold for categorical data. CV methods below:

* K-Fold

*Splits the data into N groups and takes one as test, repeats N times*

Diagram

Description automatically generated

* Repeated K-Fold

*Repeats K-fold n times, producing different splits for each repetition.*

* Shuffle & Split

*Iterator will generate a user defined number of independent train / test dataset splits. Samples are first shuffled and then split into a pair of train and test sets.*

* StratifiedKFold

*Splits categorical data to match class frequencies*

Diagram

Description automatically generated

* Stratified Shuffle Split

*Shuffles and splits based on class frequencies*

* Leave one out

*Picks all but one point, same as K-Fold if N was set to rows – 1*

* Shuffle-split

*Shuffles the data for each fold and splits to N groups*

Diagram, table

Description automatically generated

Column Transformers

*Splits data into different transformers to maintain the workflow*

A diagram of different colored squares

Description automatically generated with low confidence

Pipelines

*Creates a pipeline of steps (impute, encode, scale, model). This maintains data integrity and prevent data leakage.*

Diagram

Description automatically generated

# Regression Models

## Linear Regression Assumptions:

*Linear regression is modeled with the equation above. Beta’s are coefs / weights and epsilon is a random variable from distribution E() = 0. The equation is expanded for multiple input variables x, but is still considered to come from the same distribution, otherwise modelling errors separately for each variable becomes too complex. Since is a random variable Y is a random variable, otherwise the model would be deterministic instead of stochastic. Random variable is synonymous with model residuals.*

## Law of Large Numbers

*As number of samples from the population grows the mean of the sampling distribution will tend to the mean of the population.*

## Central Limit Theorem

*Assuming data is sampled independently, distribution of a sample variable approximates a normal distribution (i.e., a “bell curve”) as the sample size becomes larger, assuming that all samples are identical in size, and regardless of the population's actual distribution shape.*

Explanation

*Consider each data row as a sample from our distribution around true regression line (y = ax + b – deterministic). Residuals correspond to epsilon from the original equation and therefore we are sampling residuals from random error distribution, with each x value having an error distribution around it. The only random variable in the equation is epsilon / residuals, therefore each data row can be considered as a measurement of values y takes on for a given X. Therefore, we are sampling error distribution for each X. Image below displays X6 and corresponding error distribution around true regression line.*

A close-up of a graph

Description automatically generated with low confidence

## Linear Regression Assumptions

1. Observations are independent from each other

*Central limit theorem requires independent samples*

1. X-Y relationship is linear

*Underlying equation is linear y=ax+b*

1. Residuals should be normally distributed

*Since we are taking a lot of independent samples from error distribution CLT states our sample distribution should be normal*

1. Residuals are centered around 0

*We are trying to fit a model y=ax+b with underlying statistical model y = ax+ b + err. For the statistical model to equate to true regression line E(residuals) has to be equal to 0. Law of large numbers states that with enough samples our residuals mean should be the same as underlying distribution 🡪 mean of residuals = 0.*

1. Variance of residuals is the same for all X’s

*Since errors / residuals are samples from the same distribution they should have the same variance and mean (CLT).*

## Linear Regression

|  |  |
| --- | --- |
| Equation (Objective Function) |  |
| Loss Function - OLS |  |
| Scaling | Not Necessary |
| Comments | Model isn’t sensitive to scaling because there is regularization. |

## Ridge Regression

|  |  |
| --- | --- |
| Equation (Objective Function) |  |
| Loss Function - OLS |  |
| Regularization – L2 | Alpha ↑ Complexity ↓ |
| Scaling | Needed penalties are based on weights, need to scale to give each feature same weight |
| Comments | Since penalties are calculated with coefs scaling features |

## Lasso Regression

|  |  |
| --- | --- |
| Equation (Objective Function) | A picture containing watch  Description automatically generated |
| Loss Function - OLS |  |
| Regularization – L1 | Alpha ↑ Complexity ↓ |
| Scaling | Needed penalties are based on weights, need to scale to give each feature same weight |
| Comments | L1 regulization performs feature selection and sets coef = 0 for not important features |

## L1 vs L2 Penalties

*L1 penalizes by adding the sum of absolute values of coefficients to cost function*

*L2 penalizes by adding the sum of squared coefficients to cost function*

Diagram, schematic

Description automatically generated

*Above is the example of loss functions. Gradient descent will iteratively make the weights change in the opposite direction of the gradient with a step size η multiplied with the gradient. This means that a steeper gradient will make us take a larger step, while a flatter gradient will make us take a smaller step.*

*Gradient for L1 gradient is either 1 or -1, except for when w1=0. That means that L1-regularization will move any weight towards 0 with the same step size, regardless the weight's value. In contrast, you can see that the L2 gradient is linearly decreasing towards 0 as the weight goes towards 0. Therefore, L2-regularization will also move any weight towards 0, but it will take smaller and smaller steps as a weight approach 0.*

Cost Functions – cost function is what gets minimized to find the solution to objective function.

* Sum of Squared Errors (SSE / OLS)

*Default by all linear models. Since the residuals get squared outliers have a major impact. Model tries to chase the outliers and correct those residuals.*

* Sum of Absolute Errors

*Summation of residuals’ absolute values. Since differences are squared it’s more resistant to outliers.*

* Huber function

*Mix of the two above. Squared residuals for smaller values and sum absolute value for higher residuals.*

Engineering drawing

Description automatically generated

## LinearSVR / SVR (Support Vector Machines)

*LinearSVR and SVM are similar, but LinearSVR runs a linear model with SVM penalties. SVM utilizes a kernel trick that transforms data to capture more complex relationships. LinearSVR with PolynomialFeatures is similar to SVR with poly kernel. Below is the SVR cost function*

Engineering drawing, polygon

Description automatically generated with medium confidence

*Given a threshold e points with residuals lying inside the threshold don’t contribute to loss and points outside of threshold contribute a linear amount. If we set epsilon margin wide enough to capture all the small residuals, we are only fitting the model on the outliers. The points that have a loss are called support vectors.*

|  |  |
| --- | --- |
| Equation (Objective Function) | A picture containing watch  Description automatically generated |
| Loss Function  Cost = C in sklearn |  |
| Regularization | C ↑ Complexity ↑  LinearSVR: epsilon ↑ Complexity ↓  SVR: gamma ↑ Complexity ↑ |
| Scaling | Needed, uses inner products of distances |
| Comments | Doesn’t scale well on big data sets. |

## Trees

*Trees scan over each feature and each value (threshold) to find the split that will minimize the MSE. Predictions are made as mean in current leaf, therefore MSE is calculated as SSE with each sample and mean of that split and finds an average across all splits. Impurity of each leaf is measured as SSE.*

|  |  |
| --- | --- |
| Equation (Objective Function) |  |
| Loss Function – MSE or MAE |  |
| Regularization – L1 | max\_depth  max\_leaf\_nodes  min\_sample\_split  min\_impurity\_decrease |
| Scaling | Not Needed |
| Comments | Trees tend to get deep and overfit, therefore need to preprune the tree prior to running the algorithm.  Trees can’t forecast on unseen data because predictions made as mean in the final leaf.  Trees are unstable and with minor changes to train data split path will change. |

## Random Forest

*Random forest works by building a lot of shallow trees and averaging the results from them all for final evaluation. It’s been proven that if you decorrelate the errors averaging will improve the results.*

Random forest randomizes the splits for each tree in 2 ways:

1. Bootstrapping of rows

*Sample with replacement to build each tree, can capture some points multiple times and omit some altogether. On average bagging uses 66% of the data, so the rest can be used as a free validation set.*

1. Sampling of features without replacement

*Done at each split, only use some of the features on every split (max\_features).*

|  |  |
| --- | --- |
| Equation (Objective Function)  Average over all tree predictions | A picture containing text  Description automatically generated |
| Loss Function – MSE or MAE | Text  Description automatically generated |
| Regularization  (max\_features, n\_estimators) | max\_features – sqrt(n\_features) for classification  max\_features – n\_features for regression  n\_estimators – trees to build  max\_depth  min\_sample\_split |
| Scaling | Not Needed |
| Comments | The more trees you use the better, start with 100 trees. Use warm\_start to search for best number of trees.  Use rf.oob\_score\_ for a free validation set score |

## Gradient Boosting

*Gradient boosting works by creating weak learners on residuals of the previous model. Therefore, the algorithm initially learns the data and then proceeds to correct residuals. Any algorithm can be used, but trees are the most common because their complexity can be easily adjusted with tree depth. Below is the diagram*

Text, letter

Description automatically generated

*There is a learning rate that discounts the corrections after each stage.*

*By default Sklearn uses randomized trees (similar to random forest) as simple learners.*

|  |  |
| --- | --- |
| Equation (Objective Function) | SEE IMAGE ABOVE |
| Loss Function (OLS) |  |
| Regularization | Pick # of trees and tune learning rate  Pick learning rate and use early stopping  Early stopping  Max\_depth – strong pruning  Number of selected features |
| Scaling | Not Needed because algorithm uses trees |
| Comments | Has oob imporvements, to get a free estimate of improved score on not selected data |
| Libraries | Sklearn, XGBoost, LightGBM, CatBoost |

# Classification Models

## Logistic Regression

*Logistic regression models maximize likelihood of training set by modeling log odds with linear regression line.*

Likelihood – describes the joint probability of the observed data as a function of the parameters of the chosen model. In plain English it’s a function that describes the joint probability of predicting correct class labels.

*Likelihood is a probability and varies between 0 and 1, which can’t be guaranteed by linear regression equation. To overcome that we can model odds instead p / (1 –p) where p is probability of success. Then the odds logged to provide numerical stability. Since log odds can range from -inf to inf there are no concerns regarding the output of the linear regressor.*

A picture containing company name

Description automatically generated

Diagram, text, schematic

Description automatically generatedChart

Description automatically generated

*Log odds are then rearranged to the final equation above. This function is called logistic sigmoid, and it ties linear regression with probability estimates. Plot above displays logistic sigmoid response and it’s evident that it crosses 0.5 probability threshold where decision function = 0 (x axis – decision function wx +b, y axis – probability p(y=1|x). Therefore, if decision function is > 0 we predict positive class, otherwise negative. Logistic regression doesn’t solve a linear equation, however the modeled line separating 2 classes is linear.*

|  |  |
| --- | --- |
| Equation (Objective Function) | SEE IMAGE ABOVE |
| Objective Function | Probability of class 1 |
| Loss Function |  |
| Regularization | C ↑ Complexity ↑  Penalties: L1, L2, elasticnet, none |
| Scaling | Needed to normalized penalties |
| Comments | Probabilistic model, predicts probilities of predicted class  Predicts probability with predict\_proba |

## LinearSVC/SVC (Support Vector Machines)

*Similar to Logistic Regression that it predicts class by sign of output but differs in loss function. Logistic regression uses log loss (sigmoid transformation), where SVC uses hinge loss (similar margin approach to SVR).*

asdfChart, line chart

Description automatically generated

*Main inspiration for this model was classifying 2 perfectly separable classes (see below)*

Chart, scatter chart

Description automatically generated

*There are an infinite number of lines that would perfectly separate the two classes, however no certain metric for which is the best one. SVC picks a line that separates the classes using “margin”. Loosely speaking the margin is the distance between the classification boundary and the closest training set point. As complexity increases the margin gets smaller (see image below).*

Chart, scatter chart

Description automatically generated

*Where in SVR support vectors were all the points that are outside of the margin in classification the model only considers the points between the margin lines and points on the wrong side of the line. All the points outside of the margin and correct side have a weight of 0. Points that contribute to the loss are called support vectors.*

|  |  |
| --- | --- |
| Equation (Objective Function) | SEE IMAGE ABOVE |
| Objective Function |  |
| Loss Function |  |
| Regularization | C ↑ Complexity ↑ Margin ↓  Penalties: L1, L2, elasticnet, none  SVR: gamma ↑ Complexity ↑ |
| Scaling | Needed to normalized penalties |
| Comments | Not probabilistic, but still can predict probabilities with predict\_proba  Not suggested to use probabilities for ROC, use decision\_function |

## Multiclass Classification

Multiclass separated onto 2 approaches

* One vs Rest

*Learn one classifier for each class vs rest of the classes and pick the class with the highest score. For example, with 4 classes we would build a classifier for class 1 vs rest, class 2 vs rest, etc. Then we score the point on all 4 classifiers and pick the one with the highest score as the class prediction.*

* One vs One – only SVC

*Learn one classifier for each combination of classes 1 vs 2, 1 vs 3, 1 vs 4, 2 vs 3, etc. The models are only trained on selected class rows 1vs2 would only train on data with those 2 classes as labels. Then we score a point in all the classifiers and pick the one that was selected most frequently. One vs Rest builds less classifiers, therefore it’s faster.*

## Trees

*Classification trees are similar to regression trees in process of splitting the data. Where regression tree used MSE and mean leaf as prediction, classification trees use either Cross-Entropy or Gini index to calculate decrease in impurity from each split.*

Text, letter

Description automatically generated

*Tree scans all features for the split threshold that minimizes either of the impurity metrics above.*

|  |  |
| --- | --- |
| Equation (Objective Function) | Prediction is made by target leaf and corresponds to majority class in that leaf |
| Loss Function | Minimize Gini or Cross-Entropy |
| Regularization – L1 | max\_depth  max\_leaf\_nodes  min\_sample\_split  min\_impurity\_decrease |
| Scaling | Not Needed |
| Comments | Probabilities of predictions are estimated by target leaf class frequency |

## Random Forest

*Same are regression random forest with a change to impurity metric. To estimate a new point each tree votes on classification and the proportion of votes in each class across the ensemble is the predicted probability vector.*

|  |  |
| --- | --- |
| Equation (Objective Function) | Prediction is made by each tree’s response and class frequencies from those responses |
| Loss Function | Minimize Gini or Cross-Entropy |
| Regularization – L1 | max\_depth  max\_leaf\_nodes  min\_sample\_split  min\_impurity\_decrease |
| Scaling | Not Needed |
| Comments | Probabilities of predictions are estimated by class frequencies from each tree’s prediction |

## Gradient Boosting

*Similar principal to regression gradient boosting, build a simple tree, make predictions with a tree and process continues with residuals. It’s been shown that additive modeling such as boosting can be interpreted as a forward stagewise additive model that minimizes exponential loss.*

|  |  |
| --- | --- |
| Equation (Objective Function) | SAME AS REGRESSION, COMBINATION OF EACH BOOST |
| Loss Function (equation isn’t straight forward) |  |
| Regularization | Pick # of trees and tune learning rate  Pick learning rate and use early stopping  Early stopping  Max\_depth – strong pruning  Number of selected features |
| Scaling | Not Needed because algorithm uses trees |
| Comments | Has oob imporvements, to get a free estimate of improved score on not selected data |
| Libraries | Sklearn, XGBoost, LightGBM, CatBoost |

# Model Evaluation

*Metrics in this section are just for scoring the results, they don’t impact the solution algorithm. Using cross validation and grid search it’s possible to maximize the desired metric and that way the model will be optimized on that scoring technique (later in section).*

## Regression

* R2 – easy to understand and ranges from (-1, 1). Explains how much variability is captured by the model. One downside is that it’s dependent on sample variance, so if one the same model runs on 2 different sets of data and results to the same SSE, dataset with smaller sample variance will have a higher R2 value.

A picture containing chart

Description automatically generated

* MSE – mean squared error is another metric, but it’s susceptible to outliers since residuals are squared.
* MAE – mean absolute error not as vulnerable to outliers and it tell you how much the model is off on average.
* MAPE – mean absolute % error. Normalizes residuals and large points aren’t contributing as much since we calculate % residual.

Graphical user interface

Description automatically generated with medium confidence

* Prediction Plots – plot of predicted vs true values should lie on diagonal line

Chart, scatter chart

Description automatically generated

* Residual Plots – plot true vs residuals (actual – pred). Shows residual variance and distribution. If errors are centered around 0 we don’t have systematic bias.

Chart, scatter chart

Description automatically generated Chart, histogram

Description automatically generated

* Residuals per Feature – plot sns.scatter\_matrix to plot residuals for each feature. X axis is the feature values and y values are residuals.

Calendar

Description automatically generated

## Binary Classification

*Positive and negative classes are arbitrary, usually minority class is positive*

* Confusion Matrix

Table

Description automatically generated

* Accuracy – what percentage of predictions is correct
* Precision (PPV, positive predicted value) – what % of positive predictions are actually positive
* Recall (Sensitivity, true positive rate) – what % of true positives were captured by the model
* False Positive Rate - % of true negatives classified as false positive
* F score – harmonic mean of precision and recall, combines the two metrics
* Macro average – averaging scores equally for all classes
* Weighted average – averaging scores based on class frequency (higher frequency class 🡪 higher portion of the average)

*Classification models make decisions based on either predicted probabilities or the output of the decision function. Changing the threshold will change the predicted class.*

*Text

Description automatically generated with medium confidence*

## Precision-Recall Curve

*Only cares for ordering of data points and not the threshold. The curve shows Precision and Recall for all possible thresholds, therefore the only thing that matters is the order of points with respect to each other. Since SVC doesn’t compute probabilities decision\_function is called instead. Ideal location is top left corner. Area under the curve is called Average Precision and model with the highest AP will be closer to top right corner*

## ROC

*Only cares for ordering of data points and not the threshold. The curve shows False Positive Rate and Recall for all possible thresholds, therefore the only thing that matters is the order of points with respect to each other. Area under the curve is called AUC and the model with higher AUC is usually better. Ideal position is top left corner.*

Multiclass metrics are mainly the same, but it’s preferred to use threshold-based metrics (accuracy, precision, recall, f1).

# Probability Calibration and Imbalanced Data

*Since classification models mainly work on probabilities and predict the % confidence of certain class, underlying modeled probabilities should ideally line out with actual data distribution. To check that calibration curve is utilized. There are 2 reasons for class imbalance:*

1. Business costs – if the cost of misclassifying class 1 is higher than that should increase the weight of class 1.
2. Class frequencies – if classes are imbalanced (almost always in the real world) the underlying probabilities will be imbalanced.

There are also 2 ways to deal with class imbalance and imbalanced costs:

## Adjusting Probabilities

*First method to deal with class imbalance, doesn’t change the classifier itself, just changes the estimated probabilities and decision thresholds*

Calibration Curve

*Take probability estimates from the model and bin the points by their probability of predicting class 1. Then in each bin calculate the % of points that are actually class 1 and compare the two probabilities.*

**Chart, line chart

Description automatically generated**

*Well calibrated model should have probabilities along the diagonal. X axis are binned predicted probabilities and Y axis is the percentage of label 1 in each bin.*

*Chart, line chart, histogram

Description automatically generated*

*Model on the left is too certain because it’s predicted probability @ 100% is lower than actual probability of 90%. The opposite is true in the model to the left, it’s too uncertain in predicting the positive class.*

Brier Score

*Brier score tells us how close the line is to an ideal diagonal. Lower score 🡪 better calibrated probabilities.*

If the probabilities aren’t well calibrated, they can be calibrated with building another model on top of classification model. Fitting a probability calibration model doesn’t change the point ranking, therefore order based metrics (AUC, AP) aren’t impacted by this transformation. **Need to use a holdout set or CV after classifier is fitted.**

There are 2 main methods:

1. Platt Scaling – uses logistic sigmoid to calibrate class frequencies, works well with SVM
2. Isotonic Regression – another alternative to plat scaling

Below is the example code for utilizing CalibratedClassifierCV

Chart, histogram

Description automatically generated

## Adjusting Classifier

*This method changes the classifier itself to account for cost and frequency imbalances.*

1. Random Undersampling – easiest strategy is to undersample the majority class to match the frequency of minority class.
2. Random Oversampling – oversample minority class to match the frequency of majority class.
3. Class – weights – in sklearn classifiers have options for class\_weight, setting to balanced will sample minority class points as many times to match majority class frequency.

# Model Interpretation & Feature Selection

*Model inspection only tell us about the model, nothing can be inferred about the data!!*

## Model Interpretations

1. Coef\_ for linear models – inspect coefficients for each feature. To compare features should scale all the data. Correlation between coefficients will make them unstable.
2. Feature\_imporances – for tree-based models
3. Permutation importances – shuffles each feature and scores the model, output baseline\_score – permutation\_score. No clear guidance on whether to use on train set or validation set. Correlated features might decrease the importance of each feature by splitting it between the two.
4. Partial dependence plots – marginalize other features and calculate average model response on feature of interest, it tells us the average model response on changing feature of interest. Calculations are completed by getting a range of values for feature of interest and for each value in range we set feature of interest to that value and compute the rest of the scores for each row. Then average the response. Result is the plot that shows response to the average model. For example, if number of rooms are plotted on housing data set, it’ll show how the prediction changes with changing number of rooms.

## Feature Selection

*Feature selection reduces the number of features used for the model. Could prevent overfitting, shrink your model and reduce computation time while retaining a reasonable score.*

1. Covariance – check covariance between features and drop the one with the highest sum of covariances. Continuous features will run Pearson’s correlation test and categorical will run Chi2 test. Getting rid of correlated features.
2. Univariate Statistics (ANOVA) – uses linear model as base. Utilize f test, chi2 for classification, output will show F value / Chi 2 value and P score. Below are the null hypothesis for tests

Graphical user interface, text, application

Description automatically generated

1. Model Based
   1. SelectFromModel – fits a simple model to training set and select top k features with highest coef\_
   2. Recursive Feature Elimination – fit the model, drop least important feature, refit the model, drop least important features, etc. At the end will be left with k most important features.
   3. Wrapper Method – drop each feature and CV score the model, drop the feature where excluding it results in the highest score (R2, accuracy). Could work backwards and start with 0 features and add the one with the highest R2 score.

# Parameter Tuning and AutoML

*Parameters for each model need to be tuned, below is the breakdown of different tuning techniques.*

## Grid Search

*sklearn offers a GridSearch function that would search over parameters and pick the set of parameters that yield the best score. Exponential in computation, CV fits model for each fold and each parameter.*

## Random Search

*Random search pick values at random from provided ranges and computes N configuration to pick the best.*

*Text

Description automatically generated*

## Bayesian Optimization (SMBO, state of the art optimization)

*Fits a cheap surrogate probabilistic model to a black box model output. Considers model itself as black box. Surrogate models aren’t complex and cheap to compute, therefore it’s a quick solution. Then acquisition function picks the next test point by estimating uncertainty along surrogate function and picking the point with highest uncertainty.*

*Surrogate functions could be Gaussian Process, Random Forest, Non-parametric, Neural Nets*

*Chart

Description automatically generated with medium confidence*

## Multi-Fidelity Search

*Strongly subset the data or use a more simplistic model to run grid search. Grid search results should narrow down the performance envelope and it’s usually correspondent with results of the full model/data set.*

## Successive Halving

*Picking budget and n configurations the process would try all configurations and split budget across each configuration equally. Keep half of best configurations, double the budget for next iteration, repeat.*

Autosklearn – library for Bayesian optimization and other techniques.

# Neural Networks

*Below is the visualization of logistic regression*

Diagram

Description automatically generated

*Each node represents different feature and w represent the weights. Y is calculated by multiplying the feature by its weight and summing for the result. Neural networks have intermediate steps between the estimations.*

Diagram

Description automatically generated

*In Neural networks the process of calculating a weighted sum is repeated for each hidden cell within hidden unit. Weights are initially randomized, therefore resulting in different hidden layer values. After h values are estimated an activation function is then applied, otherwise a linear combination of linear combinations is still a linear combination.*

*Sample calculation for the network above:*

Text

Description automatically generated

|  |  |
| --- | --- |
| Equation (Objective Function) |  |
| Loss Function  OLS -regression  Cross-entropy - classif |  |
| Activation Functions  tanh – if weights are high the derivative becomes minimal, longer to learn  relu – derivative is always the same, faster to converge |  |
| Scaling | Needed |
| Regularization | Number of parameters – coefficients (layers, cells)  Regularization – L2 regression called weight decay  Drop Out – turns off random neural nodes  Early Stopping – stop when error isn’t decreasing |
| Comments | If the model is too deep or complex it might not fit the train data well, need to decrease model complexity in that case. Deeper the network harder it is to optimize. |

## Keras Neural Nets

1. Create a model

Text

Description automatically generated

*First layer needs the number of features (784) and how many hidden nodes (32), then activation function and finally the final layer with classification classes (10). Last activation is how to pick the class.*

1. Compile

Graphical user interface, text, application, email

Description automatically generated

1. Fit

Graphical user interface, text, application, email

Description automatically generated

*Epochs – number of iterations for convergence*

1. Need to convert categorical targets

Text

Description automatically generated

1. Evaluate – same as score

*Can create a sklearn wrapper to behave same as sklearn models*

Text

Description automatically generated

## Convolution neural nets

*Mainly used on image processing. There are 2 main ideas*

* Translation invariance

*One is if you move your focus of an image, the semantics of the image are the same. So if I look a little bit further right a pixel or so then this shouldn't really change my interpretation of the image. It also implies if I detect something on the right side, it's sort of the same as detecting something in the left side, for many natural image tasks. So if I want to say is there’s a person in the image, it doesn't matter if the person is in the left, right, top or bottom, detecting a person will always be the same thing.*

* Weight sharing

*Basically, you can detect something no matter where it is in an image.*

Convolution – smooths the function

Text

Description automatically generated

*f – our data (image, time series)*

*g – kernel/floater, computes 2nd derivative of f*

*Process – overlaying filter (g) and slide over series. In this case it’s a three number window. Multiply g \* f and add. Pink 3 = -1\*2 + 2 \* 2 + 0\*-1*

Diagram

Description automatically generated

## Batch Normalization

*Models best learn when data is zero mean and unit variance. Otherwise, can take a while to converge if values are too high.*

## Residual Neural Nets

*Computer vision mainly*

*The model is fitting residuals vs the actual y. There is connection to skip layers*

Diagram

Description automatically generated

## Recurrent Neural Networks

*Counter part of feed-forward NN. Works on sequences (words in sentences, time steps). Used in NLP and speech recognition.*

A picture containing text, clock

Description automatically generated

*In RNN, we have an input, hidden layer and an output. Basically, the hidden layer is connected to itself. So at the first time step, you get some input here, compute the hidden layer, and get some output. At the second time step, you get some input again and compute some output, but you can also look back to the last hidden layer state. But the weights are all shared. So the matrix, U, V, and W are constant all the time.*

# Time Series

## Preprocessing

* Data should be evenly spaced. If distances aren’t the same, group to same period length and aggregate. Use pandas.resample()
* Can fill in NA’s for feature vectors with pandas fillna(‘ffill’). If aggregating to time step length, drop missing values.
* Use parse dates in pandas to combine day month year columns and set to index.

## Stationary Time Series Assumptions:

1. Mean independent of time
2. Variance independent of time
3. Covariance of 2 observations k steps apart independent of time

## Autocorrelation

*Autocorrelation basically asks you what each time step tells you about the next time step, how much correlation is there between one-time step and the future (covariance between time steps). Use pandas.autocorr()*

Table

Description automatically generated

*Use the function below to plot autocorrelation for multiple lag times or pandas.tools.plotting*

Chart, line chart

Description automatically generated

*If there is a strong autocorrelation need to de-trend, usually done with differencing between rows (pandas.diff())*

*Can analyze season model with statsmodels module*

Chart

Description automatically generated

## Autoregressive (linear) model

|  |  |
| --- | --- |
| Equation  Using previous time step to forecast |  |
| Loss Function | OLS |
| Module | Stats.models.tsa.AutoReg  Chart  Description automatically generated  Max\_lag – how many previous periods to use for forecasting current point. Needs to be at at least as long as your seasonality to capture pattern. |
| Regularization |  |
| Comments |  |

ARIMA – autoregressive integrated moving average

Implementation with Sklearn

1. Predict trend with linear model (detrend)

*Index (dates) becomes X and split by selecting first N rows for train and rest as test.*

*Predicted variable becomes y, split the same way as X.*

Text

Description automatically generated

Chart, line chart

Description automatically generated

1. Calculate season component

*Find residuals between linear model and actuals 🡪 season component*

1. Extract unit of season (month, quarter) as new feature
2. Use that feature column as continuous but introduces linearity between months. Can use OneHotEncoder on months to exclude linearity between months.

FBProphet – facebook library for time series forecasting

Chart, line chart

Description automatically generated

## Other Libraries

Graphical user interface, text, application

Description automatically generated