CS – 344 Guide 8 – Feature Engineering

* Google’s [Machine Learning Crash Course](https://developers.google.com/machine-learning/crash-course)
  + [Representation](https://developers.google.com/machine-learning/crash-course/representation/video-lecture)
    - Terms
      * *Feature vector*
        + The set of floating-point values comprising the examples in your data set.
      * *One-hot* vs. *multi-hot* encodings
        + Create a binary vector for each categorical feature in our model that represents values as follow:

For values that apply to the example, set corresponding vector elements to 1.

Set all other elements to 0.

Length of vector = # of elements in the vocabulary.

* + - * + *One-hot* – a single value is 1.
        + Multi-hot – multiple values are 1.
      * *Binning*
        + Converting a usually continuous feature into multiple binary features called buckets or bins, typically based on value range.
    - What are the qualities of good features?
      * Avoid rarely used discrete feature values:
        + Good feature values should appear more than 5 or so times in a data set
        + Many examples with same discrete value gives model a chance to see feature in different settings and determine when it is a good predictor for the label.
      * Prefer clear and obvious meanings:
        + Each feature should have a clear and obvious meaning to anyone on the project.
      * Don’t mix “magic” value with actual data:
        + Good floating-point features don’t contain peculiar out-of-range discontinuities or “magic” values.
        + Replace magic values as follows:

For discrete variables, add new value to set and use to signify feature value is missing.

For continuous variables, ensure missing values don’t affect model by using mean value of the feature’s data.

* + - * Account for upstream instability:
        + Definition of a feature shouldn’t change over time.
        + Don’t use a value inferred by another model as it could change.
    - What are the best practices for data *cleansing*?
      * Scaling feature values:
        + Convert floating-point feature values from natural range to a standard range.

Provides benefits if feature set consists of multiple features

Helps gradient descent converge more quickly

Avoids the NaN trap – value exceeds floating-point precision

Helps model learn appropriate weights for each feature

* + - * Handling extreme outliers:
        + One method is to take the logarithm of every value.
        + Another method is to cap or clip the tail of outlier values.

All values greater than the maximum now becomes the maximum

* + - * Binning:
        + Divide a floating-point feature into multiple distinct Boolean features, then unite into a single n-element vector for the n-features.
      * Scrubbing:
        + “Fix” bad examples by removing from the data set.
        + Real-life data sets unreliable due to:

Omitted values

Duplicate examples

Bad labels

Bad feature values

* + - * + Generate aggregate statistics such as:

Min/max

Mean/median

Standard deviation

* + - * Know your data:
        + Keep in mind what you think the data should look like
        + Verify that the data meets these expectations
        + Double-check that the training data agrees with other sources
  + [Feature Crosses](https://developers.google.com/machine-learning/crash-course/feature-crosses/video-lecture)
    - Are the logical functions we discussed in class (i.e., AND, OR, XOR) linear functions?
      * AND – linear function
      * OR – linear function
      * XOR – non-linear function
    - Definition: a synthetic feature formed by multiplying (crossing) two or more features.
    - Compare and contrast *synthetic features* vs. *feature crosses*.
      * Synthetic features:
        + Feature not present among input features, but created from one or more of them.

Bucketing – continuous feature into range bins

Multiplying or dividing one feature value by other feature value(s) or by itself.

Feature crosses

* + - * Feature crosses:
        + Synthetic feature formed by taking the Cartesian product of individual binary features obtained from categorical data or from continuous features via bucketing.
        + Helps represent non-linear relationships
    - How are feature crosses useful?
      * Can provide predictive abilities beyond what those features can provide individually.
      * Allows efficient training on massive-scale data sets by supplementing scaled linear models with feature crosses to represent non-linear relationships.
  + [Regularization for Simplicity`](https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/video-lecture)
    - Terms
      * *Over-fitting*
        + Creating a model that matches the training data so closely that the model fails to make correct predictions on new data.
      * *Lambda*
        + A scalar value, represented as lambda, specifying the relative importance of the regularization function.

If too high, model is simple but run risk of under-fitting the data – model won’t learn enough about training data to make useful predictions

If too low, mode is more complex but run risk of overfitting the data – model learns too much about particularities of training data and can’t generalize to new data.

* + - * + Raising the regularization rate reduces overfitting but may make the model less accurate
      * *Early stopping*
        + A method of regularization that involves ending model training before training loss finishes decreasing.

End model training when loss on validation dataset starts to increase (a.k.a. when generalization performance worsens)

* + - Definition: prevent overfitting by penalizing complex models – minimize loss + complexity
    - Compare and contrast *Loss* vs. *structural risk minimization*.
      * Loss – empirical risk minimization
        + Minimize(Loss(Data | Model))

Minimize loss.

* + - * Loss + complexity – structural risk minimization
        + Minimize(Loss(Data | Model) + complexity(Model))

Minimize loss and strongly regularize.

* + - * Loss term – measure how well the model fits the data
        + The lower the loss, the more predictive the model.
      * Regularization term – measure model complexity
        + The higher the regularization, the less complex the model.
      * Two common ways to think of model complexity:
        + As a function of the weights of all features in the model
        + As a function of the total number of features with non-zero weights
    - Compare and contrast *L0* vs. *L1* vs. *L2* regularization.
      * L0 regularization:
        + Counts the number of 0 weights in the model – discontinuities in the function – can’t take derivative as not continuous function.
      * L1 regularization:
        + Type of regularization that penalizes weights in proportion to the sum of the absolute values of the weights
        + In models relying on sparse features, helps drive weights of irrelevant or barely relevant features to exactly 0, which removes those features from the model.
      * L2 regularization:
        + Type of regularization that penalizes weights in proportion to the sum of the squares of the weights
        + Helps drive outlier weights (high positive or low negative values) closer to 0 but not quite to 0.
        + Always improves generalization in linear models.
* Programming Tools
  + [Keras](https://keras.io/)
    - What is *Keras*?
      * High-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
        + Developed with a focus on enabling fast experimentation.
    - Use if need deep learning library that:
      * Easy and fast prototyping (through user friendliness, modularity, and extensibility)
      * Support for both convolutional and recurrent networks, as combinations of both.
      * Runs seamlessly on CPU and GPU>
    - What are its guiding principles?
      * User friendliness:
        + User experience is key
        + Consistent and simple API’s
        + Minimizes # of user actions required for common use cases.
        + Provides clear and actionable feedback upon user error.
      * Modularity:
        + Model is understood as a sequence or graph of standalone, fully configurable modules combined with minimal restrictions.
      * Easy extensibility:
        + New models simple to add (as new classes/functions)
        + Existing models provide ample examples.
      * Work with python:
        + No separate model configuration files in declarative format.
        + Models described in Python code – compact, easier to debug.
    - Do the “30 seconds to Keras” exercises.
      * Keras seems to be a lot less convoluted than the Guide 7 programming exercise utilizing Tensor flow and linear regressors.