Chapter 16

Feature Design

So far in this course we have been focused on models. We have simply assumed that some input features $\mathbf{x}$ are given and our goal is to learn the best possible model given these features. However, this does not paint an accurate picture of machine learning in the real world. Typically, one of the most important aspects of machine learning is the task of *feature design*, where the goal is to find the best possible feature representation for our data.

In this chapter, we will start with the most basic forms of feature design, and we will assume that the eventual goal is to perform a supervised learning task. The goal of these techniques is to transform an initial input feature representation $\mathbf{x}' \in \mathbb{R}^{m'}$ into the feature representation $\mathbf{x} \in \mathbb{R}^m$ that we actually use in our model. The hope is that by designing better features we can facilitate the learning of more accurate and powerful models. Indeed, it is a common knowledge in machine learning that extracting the right features from data often leads to the biggest performance improvements, e.g., compared to designing more complex models.

All of the techniques that we will discuss in this chapter are essentially heuristic techniques that researchers use to manually define features. In the following chapters, we will begin to introduce more powerful techniques that can *automatically* learn useful feature representations from data. These are generally called *representation learning* approaches, and the most sophisticated of these approaches are *deep neural networks*, which we will cover in the final part of this course.

16.1 Feature Design: An Overview

We begin with an overview of the high-level approach to feature design. Generally, we will assume that we are given some initial feature representations $\mathbf{x}' \in \mathbb{R}^{m'}$, which refer to as the *raw features*. For example, if we were doing image classification, then the raw features would simply be a vectorized version of the pixel values in the image, while in text classification, the raw features

often correspond to word counts.

In general, raw features typically have two issues: first, the raw features can contain redundant or useless information; second, important information is often hidden in the raw features and not easily accessible for a model. For instance, suppose we are trying to classify the sentiment of movie reviews. There will be many words—such as "the" and "and"—whose occurrence simply irrelevant to this task. On the other hand, the occurrence of some words—such as "great" and "terrible"—will have outsize performance on the prediction task. It is thus critical that we determine the right set of features to extract from the raw feature set.

Recall from Chapter 10 that adding more features tends to make models overfit. This is especially true for noisy or useless features. Indeed, features that are completely unrelated to the prediction task will add complexity to your model—i.e., increase the likelihood overfitting—without providing an useful performance improvements on the target prediction.

Generally, the steps in feature design are as follows:

1. Develop a candidate set of features $\tilde{\mathbf{x}}$ for your problem using the raw features $\mathbf{x}'$.
2. Select a subset of the features $\mathbf{x}$ from the candidate set $\tilde{\mathbf{x}}$.
3. Test a model with the selected features on the validation set.
4. Possibly iterate steps 1-3 using different candidate sets of features.

16.2 Generating Features

One of the most basic approaches to feature design is to generate additional features. In Chapter 10, we already covered some approaches to doing this, which we will re-iterate here. We will also discuss some domain-specific approaches to feature generation in the context of natural language processing.

16.2.1 Feature Transformations

The most basic approach to feature generation is to expand the original seed set of features through transformations. Some examples of popular transformations include:

- **Polynomial expansions** take an initial feature vector $\mathbf{x}'$ and generate k-degree polynomials based on the initial set:

$$\mathbf{x} = \left[\mathbf{x}'[0]^k + \mathbf{x}'[0]^{k-1}\mathbf{x}'[1] + \mathbf{x}'[0]^{k-2}\mathbf{x}'[1]\mathbf{x}'[2] + ...\right]^\top \tag{16.1}$$

  In a full k-degree expansion, we include all powers of each raw feature from $\mathbf{x}'[i]$ to $\mathbf{x}'[i]^k$ as well as all possible interaction terms containing products of up to k of the raw features. It is also possible to use only a subset of the

full terms in the polynomial expansion, which includes some of the other common feature expansions below. Polynomial expansions are useful in situations where the researcher believes that underlying function mapping the features to the target is actually non-linear, and typically expansions are limited to $k = 2$ or $k = 3$.

- **Logarithmic transformations** take a raw feature value $\mathbf{x}'[i]$ and create a new feature $\log(\mathbf{x}[i])$. Logarithmic transformations are only applicable for positive-valued features, and they tend to work well for features that have *heavy-tailed distributions.*

- **Interaction terms** are a special case of polynomial features where we add a new feature $\mathbf{x}'[i]\mathbf{x}'[j]$ to our data, which is an interaction (i.e., product) of two of the initial raw features. Interaction features can be useful if you suspect that two raw features are strongly interrelated and combined complementary information.

Many other feature transformations are possible—e.g., taking the square roots of initial features or applying trigonometric functions—but the approaches listed above are by far the most popular.

16.2.2 Domain-Specific Features

In many cases, simply applying uniformed feature transformations is not the best choice, and it is possible to use *domain knowledge* to extract better feature representations from the raw feature data. We will use the task of *natural language processing*—where raw features correspond to word counts—in order to give a few examples of how this is done. In all these examples, we will assume that the initial raw features $\mathbf{x}' \in \mathbb{R}^{m'}$ correspond to word counts over an m'-dimensional vocabulary. In other words, a raw feature value $\mathbf{x}'_i[j]$ corresponds to how many times word j occurs in document i.

- **Lexicon-based features** are a popular approach in natural language processing. Here, instead of counting all words, we use a dictionary or lexicon to group words together. For example, we might have a dictionary (or lexicon) $\mathcal{L}$ of positive sentiment words (e.g., "good" or "great") and we can create a feature—call it l—based on this lexicon by grouping together how often these specific words occur

$$\mathbf{x}[l] = \sum_{j \in \mathcal{L}} \mathbf{x}'_i[j]. \tag{16.2}$$

- **Frequency transformations** are techniques used to transform the raw counts $\mathbf{x}'_i[j]$. Some popular strategies are noted below.

  - **Frequency normalization** where we normalize by the total word count in each training example/document

$$\mathbf{x}_i[j] = \frac{\mathbf{x}'_i[j]}{|\mathbf{x}'_i|}. \tag{16.3}$$

  - **Binary thresholding** where we transform the raw counts to binary values indicating whether the word occurred or not:

$$\mathbf{x}_i[j] = \begin{cases} 1 & \text{if } \mathbf{x}'_i[j] > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{16.4}$$

  - **Log transformations** computed as follows

$$\mathbf{x}_i[j] = \log(1 + \mathbf{x}'_i[j]), \tag{16.5}$$

    where we must add one in order to avoid taking logarithms of zero values.

- **Weighting by inverse document frequency (IDF)** is an extremely useful approach. Here, we actually normalize the word counts according to how many different training examples/documents each word occurs in. The idea is that words—such as "the"—that occur in many training documents are not particularly informative. Typically, we compute an inverse-document frequency (IDF) score $I(j)$ for each word j as follows:

$$I(j) = |\mathbf{x}'_i \in \mathcal{D} : \mathbf{x}'_i[j] > 0|. \tag{16.6}$$

  We then use this score to normalize the features within each document, typically with a logarithmic transformation:

$$\mathbf{x}_i[j] = \frac{\mathbf{x}'_i[j]}{\log(I(j))}. \tag{16.7}$$

  This IDF approach can be combined with any of the frequency transformations noted above.

16.2.3 Feature Normalization

No matter what kind of features are being extracted or transformed, a critical aspect of feature design is properly normalizing features. For example, suppose that we are building a model for spam classification, with one feature representing the word count for the email and another feature representing the number of attachments. Typically, the number of words in an email will be 10-50 times larger than the number of attachments, which means that the word count feature will typically be 10-50 times larger than the attachment feature. This kind of imbalance in feature magnitude can be problematic for many models and especially models that use regularization.

For this reason, it is standard practice to normalize features. There are two common normalization strategies. Note that in this section we continue to use $\mathbf{x}'$ to denote the original features and $\mathbf{x}$ to denote the normalized features. However, it is entirely possible that we are normalizing features that were derived via some of the feature generation techniques introduced previously.

Standard normalization is the most common approach used. In this approach, we normalize each feature to have a mean of zero and a variance of one (according to the empirical distribution of the training data). That is, we compute

$$\mathbf{x}_i[j] = \frac{\mathbf{x}'_i[j] - \mu_j}{\sigma_j}, \tag{16.8}$$

where

$$\mu_j = \frac{\sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} \mathbf{x}'[j]}{|\mathcal{D}_{\text{trn}}|}, \tag{16.9}$$

and

$$\sigma_j = \sqrt{\frac{\sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} (\mathbf{x}'[j] - \mu_j)^2}{|\mathcal{D}_{\text{trn}}| - 1}}. \tag{16.10}$$

Fixed-range normalization is another popular approach. In this approach, we normalize the values to have a fixed range (typically $[-1, 1]$):

$$2 \times \frac{\mathbf{x}'[j] - \min\left(\{\mathbf{x}'[j] : \mathbf{x}' \in \mathcal{D}\}\right)}{\max\left(\{\mathbf{x}'[j] : \mathbf{x}' \in \mathcal{D}\}\right) - \min\left(\{\mathbf{x}'[j] : \mathbf{x}' \in \mathcal{D}\}\right)} - 1 \tag{16.11}$$

Fixed-range normalization can be useful if the researcher requires that the features are bounded, but typically standard normalization is employed.

16.3 Selecting Features

After feature generation and normalization, the final stage of feature design is selection. In this step, one typically prunes away features that might not be helpful and prioritizes only the features that look the most useful. Feature selection can be critical because adding unnecessary features to a model can increase both computational cost as well as the likelihood of overfitting.

16.3.1 Ranking Features

One approach that can often be useful for weeding up useless features is a ranking approach. In this approach, we rank the features based on their correlation with the target value, and we select only the top-ranked features to include in our model. Some popular ranking approaches include the following:

- **Pearson correlation** can be used to compute the linear correlation between each feature and the target according to the formula

$$r(j, y) = \frac{\sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{trn}}} (\mathbf{x}[j] - \mu_j)(y - \mu_y)}{\sqrt{\sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{trn}}} (\mathbf{x}[j] - \mu_j)^2} \sqrt{\sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{trn}}} (y - \mu_y)^2}}, \tag{16.12}$$

  where μ_j and μ_y are the average of $\mathbf{x}[j]$ and y across all training points (as in Equation 16.9). Features with the highest correlation can be kept.

Typically, this approach works best for regression tasks, where $y \in \mathbb{R}$ is a continuous value. The Pearson correlation is not well-defined in cases where y is discrete.

- **The mutual information** between a feature and the target can also be used. For example, in the case of a binary feature $\mathbf{x}[j]$ and a binary classification target y, we can compute

$$I(\mathbf{x}[j], y) = \sum_{k \in \{0,1\}} \sum_{c \in \{0,1\}} P(\mathbf{x}[j] = k, y = c) \log \left(\frac{P(\mathbf{x}[j] = k, y = c)}{P(\mathbf{x}[j] = k) P(y = c)} \right)$$

by estimating the probabilities from the training set as

$$P(\mathbf{x}[j] = k, y = c) = \frac{|(\mathbf{x}, y) \in \mathcal{D}_{\text{trn}} : \mathbf{x}[j] = k, y = c|}{|\mathcal{D}_{\text{trn}}|} \tag{16.13}$$

$$P(\mathbf{x}[j] = k) = \frac{|(\mathbf{x}, y) \in \mathcal{D}_{\text{trn}} : \mathbf{x}[j] = k|}{|\mathcal{D}_{\text{trn}}|} \tag{16.14}$$

$$P(y = c) = \frac{|(\mathbf{x}, y) \in \mathcal{D}_{\text{trn}} : y = c|}{|\mathcal{D}_{\text{trn}}|}. \tag{16.15}$$

The mutual information has a benefit over the Pearson correlation in that it can capture non-linear dependencies between features and the target. However, the mutual information is only tractable to compute for discrete (e.g., binary) features and targets.

- **Validation set improvement** is the final—and most powerful—ranking method. In this approach, we compute the performance on the validation set with—and without—feature j. We can then rank all the features according to how much improvement they provide on the validation set.

16.3.2 Subset Selection

One limitation of ranking-based methods is that they do not consider dependencies between features. For example, two features might only be useful when they are combined together in a model. For this reason, some researchers also consider subset selection methods, where groups of features are tested together. Typically, subset selection approaches test how well different subsets of features perform on the validation set, and the best performing subset is selected. The challenge with subset selection—however—is that there are exponentially many possible subsets to consider. For this reason, researchers often resort to intuition or individual ranking approaches.

16.3.3 Regularization as Feature Selection

Lastly, it is worth noting that regularization methods (as discussed in Chapter 11) implicitly perform feature selection. Indeed, when one applies L1 or L2 regularization, this has a similar impact as removing the least useful features.

In fact, regularization will tend to downweight the features that are least useful. This regularization can be viewed as an effective and elegant strategy for feature selection.