25 Feature Selection and Generation

In the beginning of the book, we discussed the abstract model of learning, in which the prior knowledge utilized by the learner is fully encoded by the choice of the hypothesis class. However, there is another modeling choice, which we have so far ignored: How do we represent the instance space $\mathcal{X}$? For example, in the papayas learning problem, we proposed the hypothesis class of rectangles in the softness-color two dimensional plane. That is, our first modeling choice was to represent a papaya as a two dimensional point corresponding to its softness and color. Only after that did we choose the hypothesis class of rectangles as a class of mappings from the plane into the label set. The transformation from the real world object "papaya" into the scalar representing its softness or its color is called a *feature function* or a feature for short; namely, any measurement of the real world object can be regarded as a feature. If $\mathcal{X}$ is a subset of a vector space, each $x \in \mathcal{X}$ is sometimes referred to as a *feature vector*. It is important to understand that the way we encode real world objects as an instance space $\mathcal{X}$ is by itself prior knowledge about the problem.

Furthermore, even when we already have an instance space $\mathcal{X}$ which is represented as a subset of a vector space, we might still want to change it into a different representation and apply a hypothesis class on top of it. That is, we may define a hypothesis class on $\mathcal{X}$ by composing some class $\mathcal{H}$ on top of a feature function which maps $\mathcal{X}$ into some other vector space $\mathcal{X}'$. We have already encountered examples of such compositions – in Chapter 15 we saw that kernel-based SVM learns a composition of the class of halfspaces over a feature mapping ψ that maps each original instance in $\mathcal{X}$ into some Hilbert space. And, indeed, the choice of ψ is another form of prior knowledge we impose on the problem.

In this chapter we study several methods for constructing a good feature set. We start with the problem of *feature selection*, in which we have a large pool of features and our goal is to select a small number of features that will be used by our predictor. Next, we discuss *feature manipulations and normalization*. These include simple transformations that we apply on our original features. Such transformations may decrease the sample complexity of our learning algorithm, its bias, or its computational complexity. Last, we discuss several approaches for *feature learning*. In these methods, we try to automate the process of feature construction.

We emphasize that while there are some common techniques for feature learning one may want to try, the No-Free-Lunch theorem implies that there is no ultimate feature learner. Any feature learning algorithm might fail on some problem. In other words, the success of each feature learner relies (sometimes implicitly) on some form of prior assumption on the data distribution. Furthermore, the relative quality of features highly depends on the learning algorithm we are later going to apply using these features. This is illustrated in the following example.

Example 25.1 Consider a regression problem in which $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \mathbb{R}$, and the loss function is the squared loss. Suppose that the underlying distribution is such that an example $(\mathbf{x}, y)$ is generated as follows: First, we sample x_1 from the uniform distribution over $[-1, 1]$. Then, we deterministically set $y = {x_1}^2$. Finally, the second feature is set to be $x_2 = y + z$, where z is sampled from the uniform distribution over $[-0.01, 0.01]$. Suppose we would like to choose a single feature. Intuitively, the first feature should be preferred over the second feature as the target can be perfectly predicted based on the first feature alone, while it cannot be perfectly predicted based on the second feature. Indeed, choosing the first feature would be the right choice if we are later going to apply polynomial regression of degree at least 2. However, if the learner is going to be a linear regressor, then we should prefer the *second* feature over the first one, since the optimal linear predictor based on the first feature will have a larger risk than the optimal linear predictor based on the second feature.

25.1 Feature Selection

Throughout this section we assume that $\mathcal{X} = \mathbb{R}^d$. That is, each instance is represented as a vector of d features. Our goal is to learn a predictor that only relies on $k \ll d$ features. Predictors that use only a small subset of features require a smaller memory footprint and can be applied faster. Furthermore, in applications such as medical diagnostics, obtaining each possible "feature" (e.g., test result) can be costly; therefore, a predictor that uses only a small number of features is desirable even at the cost of a small degradation in performance, relative to a predictor that uses more features. Finally, constraining the hypothesis class to use a small subset of features can reduce its estimation error and thus prevent overfitting.

Ideally, we could have tried all subsets of k out of d features and choose the subset which leads to the best performing predictor. However, such an exhaustive search is usually computationally intractable. In the following we describe three computationally feasible approaches for feature selection. While these methods cannot guarantee finding the optimal subset, they often work reasonably well in practice. Some of the methods come with formal guarantees on the quality of the selected subsets under certain assumptions. We do not discuss these guarantees here.

25.1.1 Filters

Maybe the simplest approach for feature selection is the filter method, in which we assess individual features, independently of other features, according to some quality measure. We can then select the k features that achieve the highest score (alternatively, decide also on the number of features to select according to the value of their scores).

Many quality measures for features have been proposed in the literature. Maybe the most straightforward approach is to set the score of a feature according to the error rate of a predictor that is trained solely by that feature.

To illustrate this, consider a linear regression problem with the squared loss. Let $\mathbf{v} = (x_{1,j}, \ldots, x_{m,j}) \in \mathbb{R}^m$ be a vector designating the values of the jth feature on a training set of m examples and let $\mathbf{y} = (y_1, \ldots, y_m) \in \mathbb{R}^m$ be the values of the target on the same m examples. The empirical squared loss of an ERM linear predictor that uses only the jth feature would be

$$\min_{a,b \in \mathbb{R}} \frac{1}{m} \|a\mathbf{v} + b - \mathbf{y}\|^2,$$

where the meaning of adding a scalar b to a vector $\mathbf{v}$ is adding b to all coordinates of $\mathbf{v}$. To solve this problem, let $\bar{v} = \frac{1}{m}\sum_{i=1}^m v_i$ be the averaged value of the feature and let $\bar{y} = \frac{1}{m}\sum_{i=1}^m y_i$ be the averaged value of the target. Clearly (see Exercise 1),

$$\min_{a,b \in \mathbb{R}} \frac{1}{m} \|a\mathbf{v} + b - \mathbf{y}\|^2 = \min_{a,b \in \mathbb{R}} \frac{1}{m} \|a(\mathbf{v} - \bar{v}) + b - (\mathbf{y} - \bar{y})\|^2. \tag{25.1}$$

Taking the derivative of the right-hand side objective with respect to b and comparing it to zero we obtain that $b = 0$. Similarly, solving for a (once we know that $b = 0$) yields $a = \langle \mathbf{v} - \bar{v}, \mathbf{y} - \bar{y} \rangle / \|\mathbf{v} - \bar{v}\|^2$. Plugging this value back into the objective we obtain the value

$$\|\mathbf{y} - \bar{y}\|^2 - \frac{(\langle \mathbf{v} - \bar{v}, \mathbf{y} - \bar{y} \rangle)^2}{\|\mathbf{v} - \bar{v}\|^2}.$$

Ranking the features according to the minimal loss they achieve is equivalent to ranking them according to the absolute value of the following score (where now a higher score yields a better feature):

$$\frac{\langle \mathbf{v} - \bar{v}, \mathbf{y} - \bar{y} \rangle}{\|\mathbf{v} - \bar{v}\| \, \|\mathbf{y} - \bar{y}\|} = \frac{\frac{1}{m} \langle \mathbf{v} - \bar{v}, \mathbf{y} - \bar{y} \rangle}{\sqrt{\frac{1}{m}\|\mathbf{v} - \bar{v}\|^2} \sqrt{\frac{1}{m}\|\mathbf{y} - \bar{y}\|^2}}. \tag{25.2}$$

The preceding expression is known as *Pearson's correlation coefficient*. The numerator is the empirical estimate of the *covariance* of the jth feature and the target value, $\mathbb{E}[(v - \mathbb{E}\, v)(y - \mathbb{E}\, y)]$, while the denominator is the squared root of the empirical estimate for the *variance* of the jth feature, $\mathbb{E}[(v - \mathbb{E}\, v)^2]$, times the variance of the target. Pearson's coefficient ranges from -1 to 1, where if the Pearson's coefficient is either 1 or -1, there is a linear mapping from $\mathbf{v}$ to $\mathbf{y}$ with zero empirical risk.

If Pearson's coefficient equals zero it means that the optimal linear function from $\mathbf{v}$ to $\mathbf{y}$ is the all-zeros function, which means that $\mathbf{v}$ *alone* is useless for predicting $\mathbf{y}$. However, this does not mean that $\mathbf{v}$ is a bad feature, as it might be the case that together with other features $\mathbf{v}$ can perfectly predict $\mathbf{y}$. Indeed, consider a simple example in which the target is generated by the function $y = x_1 + 2x_2$. Assume also that x_1 is generated from the uniform distribution over $\{\pm 1\}$, and $x_2 = -\frac{1}{2}x_1 + \frac{1}{2}z$, where z is also generated i.i.d. from the uniform distribution over $\{\pm 1\}$. Then, $\mathbb{E}[x_1] = \mathbb{E}[x_2] = \mathbb{E}[y] = 0$, and we also have

$$\mathbb{E}[yx_1] = \mathbb{E}[x_1^2] + 2\,\mathbb{E}[x_2x_1] = \mathbb{E}[x_1^2] - \mathbb{E}[x_1^2] + \mathbb{E}[zx_1] = 0.$$

Therefore, for a large enough training set, the first feature is likely to have a Pearson's correlation coefficient that is close to zero, and hence it will most probably not be selected. However, no function can predict the target value well without knowing the first feature.

There are many other score functions that can be used by a filter method. Notable examples are estimators of the mutual information or the area under the receiver operating characteristic (ROC) curve. All of these score functions suffer from similar problems to the one illustrated previously. We refer the reader to Guyon & Elisseeff (2003).

25.1.2 Greedy Selection Approaches

Greedy selection is another popular approach for feature selection. Unlike filter methods, greedy selection approaches are coupled with the underlying learning algorithm. The simplest instance of greedy selection is forward greedy selection. We start with an empty set of features, and then we gradually add one feature at a time to the set of selected features. Given that our current set of selected features is I, we go over all $i \notin I$, and apply the learning algorithm on the set of features $I \cup \{i\}$. Each such application yields a different predictor, and we choose to add the feature that yields the predictor with the smallest risk (on the training set or on a validation set). This process continues until we either select k features, where k is a predefined budget of allowed features, or achieve an accurate enough predictor.

Example 25.2 (**Orthogonal Matching Pursuit**) To illustrate the forward greedy selection approach, we specify it to the problem of linear regression with the squared loss. Let $X \in \mathbb{R}^{m,d}$ be a matrix whose rows are the m training instances. Let $\mathbf{y} \in \mathbb{R}^m$ be the vector of the m labels. For every $i \in [d]$, let X_i be the ith column of X. Given a set $I \subset [d]$ we denote by X_I the matrix whose columns are $\{X_i : i \in I\}$.

The forward greedy selection method starts with $I_0 = \emptyset$. At iteration t, we look for the feature index j_t, which is in

$$\underset{j}{\operatorname{argmin}} \min_{\mathbf{w}\in\mathbb{R}^t} \|X_{I_{t-1}\cup\{j\}}\mathbf{w} - \mathbf{y}\|^2.$$

Then, we update $I_t = I_{t-1} \cup \{j_t\}$.

We now describe a more efficient implementation of the forward greedy selection approach for linear regression which is called *Orthogonal Matching Pursuit (OMP)*. The idea is to keep an orthogonal basis of the features aggregated so far. Let V_t be a matrix whose columns form an orthonormal basis of the columns of X_{I_t}.

Clearly,

$$\min_{\mathbf{w}} \|X_{I_t}\mathbf{w} - \mathbf{y}\|^2 = \min_{\boldsymbol{\theta}\in\mathbb{R}^t} \|V_t\boldsymbol{\theta} - \mathbf{y}\|^2 .$$

We will maintain a vector $\boldsymbol{\theta}_t$ which minimizes the right-hand side of the equation.

Initially, we set $I_0 = \emptyset$, $V_0 = \emptyset$, and $\boldsymbol{\theta}_1$ to be the empty vector. At round t, for every j, we decompose $X_j = \mathbf{v}_j + \mathbf{u}_j$ where $\mathbf{v}_j = V_{t-1}V_{t-1}^\top X_j$ is the projection of X_j onto the subspace spanned by V_{t-1} and $\mathbf{u}_j$ is the part of X_j orthogonal to V_{t-1} (see Appendix C). Then,

$$\begin{aligned}
&\min_{\boldsymbol{\theta},\alpha} \|V_{t-1}\boldsymbol{\theta} + \alpha\mathbf{u}_j - \mathbf{y}\|^2 \\
&\quad = \min_{\boldsymbol{\theta},\alpha} \left[\|V_{t-1}\boldsymbol{\theta} - \mathbf{y}\|^2 + \alpha^2\|\mathbf{u}_j\|^2 + 2\alpha\langle \mathbf{u}_j, V_{t-1}\boldsymbol{\theta} - \mathbf{y}\rangle\right] \\
&\quad = \min_{\boldsymbol{\theta},\alpha} \left[\|V_{t-1}\boldsymbol{\theta} - \mathbf{y}\|^2 + \alpha^2\|\mathbf{u}_j\|^2 + 2\alpha\langle \mathbf{u}_j, -\mathbf{y}\rangle\right] \\
&\quad = \min_{\boldsymbol{\theta}} \left[\|V_{t-1}\boldsymbol{\theta} - \mathbf{y}\|^2\right] + \min_{\alpha}\left[\alpha^2\|\mathbf{u}_j\|^2 - 2\alpha\langle \mathbf{u}_j, \mathbf{y}\rangle\right] \\
&\quad = \left[\|V_{t-1}\boldsymbol{\theta}_{t-1} - \mathbf{y}\|^2\right] + \min_{\alpha}\left[\alpha^2\|\mathbf{u}_j\|^2 - 2\alpha\langle \mathbf{u}_j, \mathbf{y}\rangle\right] \\
&\quad = \|V_{t-1}\boldsymbol{\theta}_{t-1} - \mathbf{y}\|^2 - \frac{(\langle \mathbf{u}_j, \mathbf{y}\rangle)^2}{\|\mathbf{u}_j\|^2} .
\end{aligned}$$

It follows that we should select the feature

$$j_t = \operatorname*{argmax}_j \frac{(\langle \mathbf{u}_j, \mathbf{y}\rangle)^2}{\|\mathbf{u}_j\|^2} .$$

The rest of the update is to set

$$V_t = \left[V_{t-1}, \ \frac{\mathbf{u}_{j_t}}{\|\mathbf{u}_{j_t}\|^2}\right] \quad , \quad \boldsymbol{\theta}_t = \left[\boldsymbol{\theta}_{t-1}\ ;\ \frac{\langle \mathbf{u}_{j_t}, \mathbf{y}\rangle}{\|\mathbf{u}_{j_t}\|^2}\right] .$$

The OMP procedure maintains an orthonormal basis of the selected features, where in the preceding description, the orthonormalization property is obtained by a procedure similar to Gram-Schmidt orthonormalization. In practice, the Gram-Schmidt procedure is often numerically unstable. In the pseudocode that follows we use SVD (see Section C.4) at the end of each round to obtain an orthonormal basis in a numerically stable manner.

```
Orthogonal Matching Pursuit (OMP)

input:
    data matrix $X \in \mathbb{R}^{m,d}$, labels vector $\mathbf{y} \in \mathbb{R}^m$,
    budget of features $T$
initialize: $I_1 = \emptyset$
for $t = 1, \ldots, T$
    use SVD to find an orthonormal basis $V \in \mathbb{R}^{m,t-1}$ of $X_{I_t}$
      (for $t = 1$ set $V$ to be the all zeros matrix)
    foreach $j \in [d] \setminus I_t$ let $\mathbf{u}_j = X_j - VV^\top X_j$
    let $j_t = \mathrm{argmax}_{j \notin I_t : \|\mathbf{u}_j\| > 0} \frac{(\langle \mathbf{u}_j, \mathbf{y} \rangle)^2}{\|\mathbf{u}_j\|^2}$
    update $I_{t+1} = I_t \cup \{j_t\}$
output $I_{T+1}$
```

More Efficient Greedy Selection Criteria

Let $R(\mathbf{w})$ be the empirical risk of a vector $\mathbf{w}$. At each round of the forward greedy selection method, and for every possible j, we should minimize $R(\mathbf{w})$ over the vectors $\mathbf{w}$ whose support is $I_{t-1} \cup \{j\}$. This might be time consuming.

A simpler approach is to choose j_t that minimizes

$$\underset{j}{\mathrm{argmin}} \min_{\eta \in \mathbb{R}} R(\mathbf{w}_{t-1} + \eta \mathbf{e}_j),$$

where $\mathbf{e}_j$ is the all zeros vector except 1 in the jth element. That is, we keep the weights of the previously chosen coordinates intact and only optimize over the new variable. Therefore, for each j we need to solve an optimization problem over a single variable, which is a much easier task than optimizing over t.

An even simpler approach is to upper bound $R(\mathbf{w})$ using a "simple" function and then choose the feature which leads to the largest decrease in this upper bound. For example, if R is a β-smooth function (see Equation (12.5) in Chapter 12), then

$$R(\mathbf{w} + \eta \mathbf{e}_j) \le R(\mathbf{w}) + \eta \frac{\partial R(\mathbf{w})}{\partial w_j} + \beta \eta^2 / 2.$$

Minimizing the right-hand side over η yields $\eta = -\frac{\partial R(\mathbf{w})}{\partial w_j} \cdot \frac{1}{\beta}$ and plugging this value into the above yields

$$R(\mathbf{w} + \eta \mathbf{e}_j) \le R(\mathbf{w}) - \frac{1}{2\beta} \left(\frac{\partial R(\mathbf{w})}{\partial w_j} \right)^2 .$$

This value is minimized if the partial derivative of $R(\mathbf{w})$ with respect to w_j is maximal. We can therefore choose j_t to be the index of the largest coordinate of the gradient of $R(\mathbf{w})$ at $\mathbf{w}$.

Remark 25.3 (AdaBoost as a Forward Greedy Selection Procedure) It is possible to interpret the AdaBoost algorithm from Chapter 10 as a forward greedy selection procedure with respect to the function

$$R(\mathbf{w}) = \log\left(\sum_{i=1}^{m} \exp\left(-y_i \sum_{j=1}^{d} w_j h_j(\mathbf{x}_i)\right)\right). \tag{25.3}$$

See Exercise 3.

Backward Elimination

Another popular greedy selection approach is *backward elimination*. Here, we start with the full set of features, and then we gradually remove one feature at a time from the set of features. Given that our current set of selected features is I, we go over all $i \in I$, and apply the learning algorithm on the set of features $I \setminus \{i\}$. Each such application yields a different predictor, and we choose to remove the feature i for which the predictor obtained from $I \setminus \{i\}$ has the smallest risk (on the training set or on a validation set).

Naturally, there are many possible variants of the backward elimination idea. It is also possible to combine forward and backward greedy steps.

25.1.3 Sparsity-Inducing Norms

The problem of minimizing the empirical risk subject to a budget of k features can be written as

$$\min_{\mathbf{w}} L_S(\mathbf{w}) \quad \text{s.t.} \quad \|\mathbf{w}\|_0 \le k,$$

where[1]

$$\|\mathbf{w}\|_0 = |\{i : w_i \neq 0\}|.$$

In other words, we want $\mathbf{w}$ to be sparse, which implies that we only need to measure the features corresponding to nonzero elements of $\mathbf{w}$.

Solving this optimization problem is computationally hard (Natarajan 1995, Davis, Mallat & Avellaneda 1997). A possible relaxation is to replace the non-convex function $\|\mathbf{w}\|_0$ with the ℓ_1 norm, $\|\mathbf{w}\|_1 = \sum_{i=1}^{d} |w_i|$, and to solve the problem

$$\min_{\mathbf{w}} L_S(\mathbf{w}) \quad \text{s.t.} \quad \|\mathbf{w}\|_1 \le k_1, \tag{25.4}$$

where k_1 is a parameter. Since the ℓ_1 norm is a convex function, this problem can be solved efficiently as long as the loss function is convex. A related problem is minimizing the sum of $L_S(\mathbf{w})$ plus an ℓ_1 norm regularization term,

$$\min_{\mathbf{w}} \left(L_S(\mathbf{w}) + \lambda\|\mathbf{w}\|_1\right), \tag{25.5}$$

where λ is a regularization parameter. Since for any k_1 there exists a λ such that

[1] The function $\|\cdot\|_0$ is often referred to as the ℓ_0 norm. Despite the use of the "norm" notation, $\|\cdot\|_0$ is not really a norm; for example, it does not satisfy the positive homogeneity property of norms, $\|a\mathbf{w}\|_0 \neq |a|\,\|\mathbf{w}\|_0$.

Equation (25.4) and Equation (25.5) lead to the same solution, the two problems are in some sense equivalent.

The ℓ_1 regularization often induces sparse solutions. To illustrate this, let us start with the simple optimization problem

$$\min_{w\in\mathbb{R}} \left(\frac{1}{2}w^2 - xw + \lambda|w|\right). \tag{25.6}$$

It is easy to verify (see Exercise 2) that the solution to this problem is the "soft thresholding" operator

$$w = \operatorname{sign}(x)\left[|x| - \lambda\right]_+ , \tag{25.7}$$

where $[a]_+ \stackrel{\text{def}}{=} \max\{a, 0\}$. That is, as long as the absolute value of x is smaller than λ, the optimal solution will be zero.

Next, consider a one dimensional regression problem with respect to the squared loss:

$$\operatorname*{argmin}_{w\in\mathbb{R}^m} \left(\frac{1}{2m}\sum_{i=1}^{m}(x_i w - y_i)^2 + \lambda|w|\right).$$

We can rewrite the problem as

$$\operatorname*{argmin}_{w\in\mathbb{R}^m} \left(\frac{1}{2}\left(\frac{1}{m}\sum_i x_i^2\right) w^2 - \left(\frac{1}{m}\sum_{i=1}^{m} x_i y_i\right) w + \lambda|w|\right).$$

For simplicity let us assume that $\frac{1}{m}\sum_i x_i^2 = 1$, and denote $\langle \mathbf{x}, \mathbf{y}\rangle = \sum_{i=1}^m x_i y_i$; then the optimal solution is

$$w = \operatorname{sign}(\langle \mathbf{x}, \mathbf{y}\rangle)\left[|\langle \mathbf{x}, \mathbf{y}\rangle|/m - \lambda\right]_+ .$$

That is, the solution will be zero unless the correlation between the feature $\mathbf{x}$ and the labels vector $\mathbf{y}$ is larger than λ.

Remark 25.4 Unlike the ℓ_1 norm, the ℓ_2 norm does not induce sparse solutions. Indeed, consider the problem above with an ℓ_2 regularization, namely,

$$\operatorname*{argmin}_{w\in\mathbb{R}^m} \left(\frac{1}{2m}\sum_{i=1}^{m}(x_i w - y_i)^2 + \lambda w^2\right).$$

Then, the optimal solution is

$$w = \frac{\langle \mathbf{x}, \mathbf{y}\rangle/m}{\|\mathbf{x}\|^2/m + 2\lambda}.$$

This solution will be nonzero even if the correlation between $\mathbf{x}$ and $\mathbf{y}$ is very small. In contrast, as we have shown before, when using ℓ_1 regularization, w will be nonzero only if the correlation between $\mathbf{x}$ and $\mathbf{y}$ is larger than the regularization parameter λ.

Adding ℓ_1 regularization to a linear regression problem with the squared loss yields the LASSO algorithm, defined as

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left(\frac{1}{2m} \|X\mathbf{w} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|_1 \right). \tag{25.8}$$

Under some assumptions on the distribution and the regularization parameter λ, the LASSO will find sparse solutions (see, for example, (Zhao & Yu 2006) and the references therein). Another advantage of the ℓ_1 norm is that a vector with low ℓ_1 norm can be "sparsified" (see, for example, (Shalev-Shwartz, Zhang & Srebro 2010) and the references therein).

25.2 Feature Manipulation and Normalization

Feature manipulations or normalization include simple transformations that we apply on each of our original features. Such transformations may decrease the approximation or estimation errors of our hypothesis class or can yield a faster algorithm. Similarly to the problem of feature selection, here again there are no absolute "good" and "bad" transformations, but rather each transformation that we apply should be related to the learning algorithm we are going to apply on the resulting feature vector as well as to our prior assumptions on the problem.

To motivate *normalization*, consider a linear regression problem with the squared loss. Let $X \in \mathbb{R}^{m,d}$ be a matrix whose rows are the instance vectors and let $\mathbf{y} \in \mathbb{R}^m$ be a vector of target values. Recall that ridge regression returns the vector

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left[\frac{1}{m} \|X\mathbf{w} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2 \right] = (2\lambda m I + X^\top X)^{-1} X^\top \mathbf{y}.$$

Suppose that $d = 2$ and the underlying data distribution is as follows. First we sample y uniformly at random from $\{\pm 1\}$. Then, we set x_1 to be $y + 0.5\alpha$, where α is sampled uniformly at random from $\{\pm 1\}$, and we set x_2 to be $0.0001y$. Note that the optimal weight vector is $\mathbf{w}^\star = [0; 10000]$, and $L_{\mathcal{D}}(\mathbf{w}^\star) = 0$. However, the objective of ridge regression at $\mathbf{w}^\star$ is $\lambda 10^8$. In contrast, the objective of ridge regression at $\mathbf{w} = [1; 0]$ is likely to be close to $0.25 + \lambda$. It follows that whenever $\lambda > \frac{0.25}{10^8 - 1} \approx 0.25 \times 10^{-8}$, the objective of ridge regression is smaller at the suboptimal solution $\mathbf{w} = [1; 0]$. Since λ typically should be at least $1/m$ (see the analysis in Chapter 13), it follows that in the aforementioned example, if the number of examples is smaller than 10^8 then we are likely to output a suboptimal solution.

The crux of the preceding example is that the two features have completely different scales. Feature normalization can overcome this problem. There are many ways to perform feature normalization, and one of the simplest approaches is simply to make sure that each feature receives values between -1 and 1. In the preceding example, if we divide each feature by the maximal value it attains we will obtain that $x_1 = \frac{y+0.5\alpha}{1.5}$ and $x_2 = y$. Then, for $\lambda \leq 10^{-3}$ the solution of ridge regression is quite close to $\mathbf{w}^\star$.

Moreover, the generalization bounds we have derived in Chapter 13 for regularized loss minimization depend on the norm of the optimal vector $\mathbf{w}^\star$ and on the maximal norm of the instance vectors.[2] Therefore, in the aforementioned example, before we normalize the features we have that $\|\mathbf{w}^\star\|^2 = 10^8$, while after we normalize the features we have that $\|\mathbf{w}^\star\|^2 = 1$. The maximal norm of the instance vector remains roughly the same; hence the normalization greatly improves the estimation error.

Feature normalization can also improve the runtime of the learning algorithm. For example, in Section 14.5.3 we have shown how to use the Stochastic Gradient Descent (SGD) optimization algorithm for solving the regularized loss minimization problem. The number of iterations required by SGD to converge also depends on the norm of $\mathbf{w}^\star$ and on the maximal norm of $\|\mathbf{x}\|$. Therefore, as before, using normalization can greatly decrease the runtime of SGD.

Next, we demonstrate in the following how a simple transformation on features, such as *clipping*, can sometime decrease the approximation error of our hypothesis class. Consider again linear regression with the squared loss. Let $a > 1$ be a large number, suppose that the target y is chosen uniformly at random from $\{\pm 1\}$, and then the single feature x is set to be y with probability $(1 - 1/a)$ and set to be ay with probability $1/a$. That is, most of the time our feature is bounded but with a very small probability it gets a very high value. Then, for any w, the expected squared loss of w is

$$\begin{aligned} L_{\mathcal{D}}(w) &= \mathbb{E}\, \frac{1}{2}(wx - y)^2 \\ &= \left(1 - \frac{1}{a}\right)\frac{1}{2}(wy - y)^2 + \frac{1}{a}\,\frac{1}{2}(awy - y)^2. \end{aligned}$$

Solving for w we obtain that $w^\star = \frac{2a-1}{a^2+a-1}$, which goes to zero as a goes to infinity. Therefore, the objective at $w^\star$ goes to 0.5 as a goes to infinity. For example, for $a = 100$ we will obtain $L_{\mathcal{D}}(w^\star) \geq 0.48$. Next, suppose we apply a "clipping" transformation; that is, we use the transformation $x \mapsto \operatorname{sign}(x)\min\{1, |x|\}$. Then, following this transformation, $w^\star$ becomes 1 and $L_{\mathcal{D}}(w^\star) = 0$. This simple example shows that a simple transformation can have a significant influence on the approximation error.

Of course, it is not hard to think of examples in which the same feature transformation actually hurts performance and increases the approximation error. This is not surprising, as we have already argued that feature transformations

[2] More precisely, the bounds we derived in Chapter 13 for regularized loss minimization depend on $\|\mathbf{w}^\star\|^2$ and on either the Lipschitzness or the smoothness of the loss function. For linear predictors and loss functions of the form $\ell(\mathbf{w}, (\mathbf{x}, y)) = \phi(\langle \mathbf{w}, \mathbf{x}\rangle, y)$, where ϕ is convex and either 1-Lipschitz or 1-smooth with respect to its first argument, we have that ℓ is either $\|\mathbf{x}\|$-Lipschitz or $\|\mathbf{x}\|^2$-smooth. For example, for the squared loss, $\phi(a, y) = \frac{1}{2}(a - y)^2$, and $\ell(\mathbf{w}, (\mathbf{x}, y)) = \frac{1}{2}(\langle \mathbf{w}, \mathbf{x}\rangle - y)^2$ is $\|\mathbf{x}\|^2$-smooth with respect to its first argument.

should rely on our prior assumptions on the problem. In the aforementioned example, a prior assumption that may lead us to use the "clipping" transformation is that features that get values larger than a predefined threshold value give us no additional useful information, and therefore we can clip them to the predefined threshold.

25.2.1 Examples of Feature Transformations

We now list several common techniques for feature transformations. Usually, it is helpful to combine some of these transformations (e.g., centering + scaling). In the following, we denote by $\mathbf{f} = (f_1, \ldots, f_m) \in \mathbb{R}^m$ the value of the feature f over the m training examples. Also, we denote by $\bar{f} = \frac{1}{m}\sum_{i=1}^m f_i$ the empirical mean of the feature over all examples.

Centering:
This transformation makes the feature have zero mean, by setting $f_i \leftarrow f_i - \bar{f}$.

Unit Range:
This transformation makes the range of each feature be $[0, 1]$. Formally, let $f_{\max} = \max_i f_i$ and $f_{\min} = \min_i f_i$. Then, we set $f_i \leftarrow \frac{f_i - f_{\min}}{f_{\max} - f_{\min}}$. Similarly, we can make the range of each feature be $[-1, 1]$ by the transformation $f_i \leftarrow 2\frac{f_i - f_{\min}}{f_{\max} - f_{\min}} - 1$. Of course, it is easy to make the range $[0, b]$ or $[-b, b]$, where b is a user-specified parameter.

Standardization:
This transformation makes all features have a zero mean and unit variance. Formally, let $\nu = \frac{1}{m}\sum_{i=1}^m (f_i - \bar{f})^2$ be the empirical variance of the feature. Then, we set $f_i \leftarrow \frac{f_i - \bar{f}}{\sqrt{\nu}}$.

Clipping:
This transformation clips high or low values of the feature. For example, $f_i \leftarrow \operatorname{sign}(f_i) \max\{b, |f_i|\}$, where b is a user-specified parameter.

Sigmoidal Transformation:
As its name indicates, this transformation applies a sigmoid function on the feature. For example, $f_i \leftarrow \frac{1}{1+\exp(b\, f_i)}$, where b is a user-specified parameter. This transformation can be thought of as a "soft" version of clipping: It has a small effect on values close to zero and behaves similarly to clipping on values far away from zero.

Logarithmic Transformation:
The transformation is $f_i \leftarrow \log(b+f_i)$, where b is a user-specified parameter. This is widely used when the feature is a "counting" feature. For example, suppose that the feature represents the number of appearances of a certain word in a text document. Then, the difference between zero occurrences of the word and a single occurrence is much more important than the difference between 1000 occurrences and 1001 occurrences.

Remark 25.5 In the aforementioned transformations, each feature is transformed on the basis of the values it obtains on the training set, independently of other features' values. In some situations we would like to set the parameter of the transformation on the basis of other features as well. A notable example is a transformation in which one applies a scaling to the features so that the empirical average of some norm of the instances becomes 1.

25.3 Feature Learning

So far we have discussed feature selection and manipulations. In these cases, we start with a predefined vector space $\mathbb{R}^d$, representing our features. Then, we select a subset of features (feature selection) or transform individual features (feature transformation). In this section we describe *feature learning*, in which we start with some instance space, $\mathcal{X}$, and would like to learn a function, $\psi : \mathcal{X} \to \mathbb{R}^d$, which maps instances in $\mathcal{X}$ into a representation as d-dimensional feature vectors.

The idea of feature learning is to automate the process of finding a good representation of the input space. As mentioned before, the No-Free-Lunch theorem tells us that we must incorporate some prior knowledge on the data distribution in order to build a good feature representation. In this section we present a few feature learning approaches and demonstrate conditions on the underlying data distribution in which these methods can be useful.

Throughout the book we have already seen several useful feature constructions. For example, in the context of polynomial regression, we have mapped the original instances into the vector space of all their monomials (see Section 9.2.2 in Chapter 9). After performing this mapping, we trained a *linear* predictor on top of the constructed features. Automation of this process would be to learn a transformation $\psi : \mathcal{X} \to \mathbb{R}^d$, such that the composition of the class of linear predictors on top of ψ yields a good hypothesis class for the task at hand.

In the following we describe a technique of feature construction called *dictionary learning*.

25.3.1 Dictionary Learning Using Auto-Encoders

The motivation of dictionary learning stems from a commonly used representation of documents as a "bag-of-words": Given a dictionary of words $D = \{w_1, \ldots, w_k\}$, where each w_i is a string representing a word in the dictionary, and given a document, $(p_1, \ldots, p_d)$, where each p_i is a word in the document, we represent the document as a vector $\mathbf{x} \in \{0, 1\}^k$, where x_i is 1 if $w_i = p_j$ for some $j \in [d]$, and $x_i = 0$ otherwise. It was empirically observed in many text processing tasks that linear predictors are quite powerful when applied on this representation. Intuitively, we can think of each word as a feature that measures some aspect of the document. Given labeled examples (e.g., topics of the documents), a learning algorithm searches for a linear predictor that weights these features so that a right combination of appearances of words is indicative of the label.

While in text processing there is a natural meaning to words and to the dictionary, in other applications we do not have such an intuitive representation of an instance. For example, consider the computer vision application of object recognition. Here, the instance is an image and the goal is to recognize which object appears in the image. Applying a linear predictor on the pixel-based representation of the image does not yield a good classifier. What we would like to have is a mapping ψ that would take the pixel-based representation of the image and would output a bag of "visual words," representing the content of the image. For example, a "visual word" can be "there is an eye in the image." If we had such representation, we could have applied a linear predictor on top of this representation to train a classifier for, say, face recognition. Our question is, therefore, how can we learn a dictionary of "visual words" such that a bag-of-words representation of an image would be helpful for predicting which object appears in the image?

A first naive approach for dictionary learning relies on a *clustering* algorithm (see Chapter 22). Suppose that we learn a function $c : \mathcal{X} \to \{1, \ldots, k\}$, where $c(\mathbf{x})$ is the cluster to which $\mathbf{x}$ belongs. Then, we can think of the clusters as "words," and of instances as "documents," where a document $\mathbf{x}$ is mapped to the vector $\psi(\mathbf{x}) \in \{0, 1\}^k$, where $\psi(\mathbf{x})_i$ is 1 if and only if $\mathbf{x}$ belongs to the ith cluster. Now, it is straightforward to see that applying a linear predictor on $\psi(\mathbf{x})$ is equivalent to assigning the same target value to all instances that belong to the same cluster. Furthermore, if the clustering is based on distances from a class center (e.g., k-means), then a linear predictor on $\psi(\mathbf{x})$ yields a piece-wise constant predictor on $\mathbf{x}$.

Both the k-means and PCA approaches can be regarded as special cases of a more general approach for dictionary learning which is called *auto-encoders*. In an auto-encoder we learn a pair of functions: an "encoder" function, $\psi : \mathbb{R}^d \to \mathbb{R}^k$, and a "decoder" function, $\phi : \mathbb{R}^k \to \mathbb{R}^d$. The goal of the learning process is to find a pair of functions such that the reconstruction error, $\sum_i \|\mathbf{x}_i - \phi(\psi(\mathbf{x}_i))\|^2$, is small. Of course, we can trivially set $k = d$ and both ψ, ϕ to be the identity mapping, which yields a perfect reconstruction. We therefore must restrict ψ and ϕ in some way. In PCA, we constrain $k < d$ and further restrict ψ and ϕ to be linear functions. In k-means, k is not restricted to be smaller than d, but now ψ and ϕ rely on k centroids, $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$, and $\psi(\mathbf{x})$ returns an indicator vector in $\{0,1\}^k$ that indicates the closest centroid to $\mathbf{x}$, while ϕ takes as input an indicator vector and returns the centroid representing this vector.

An important property of the k-means construction, which is key in allowing k to be larger than d, is that ψ maps instances into *sparse* vectors. In fact, in k-means only a single coordinate of $\psi(\mathbf{x})$ is nonzero. An immediate extension of the k-means construction is therefore to restrict the range of ψ to be vectors with at most s nonzero elements, where s is a small integer. In particular, let ψ and ϕ be functions that depend on $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$. The function ψ maps an instance vector $\mathbf{x}$ to a vector $\psi(\mathbf{x}) \in \mathbb{R}^k$, where $\psi(\mathbf{x})$ should have at most s nonzero elements. The function $\phi(\mathbf{v})$ is defined to be $\sum_{i=1}^k v_i \boldsymbol{\mu}_i$. As before, our goal is to have a small reconstruction error, and therefore we can define

$$\psi(\mathbf{x}) = \underset{\mathbf{v}}{\operatorname{argmin}} \|\mathbf{x} - \phi(\mathbf{v})\|^2 \quad \text{s.t.} \quad \|\mathbf{v}\|_0 \le s,$$

where $\|\mathbf{v}\|_0 = |\{j : v_j \neq 0\}|$. Note that when $s = 1$ and we further restrict $\|\mathbf{v}\|_1 = 1$ then we obtain the k-means encoding function; that is, $\psi(\mathbf{x})$ is the indicator vector of the centroid closest to $\mathbf{x}$. For larger values of s, the optimization problem in the preceding definition of ψ becomes computationally difficult. Therefore, in practice, we sometime use ℓ_1 regularization instead of the sparsity constraint and define ψ to be

$$\psi(\mathbf{x}) = \underset{\mathbf{v}}{\operatorname{argmin}} \left[\|\mathbf{x} - \phi(\mathbf{v})\|^2 + \lambda \|\mathbf{v}\|_1 \right],$$

where $\lambda > 0$ is a regularization parameter. Anyway, the dictionary learning problem is now to find the vectors $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ such that the reconstruction error, $\sum_{i=1}^m \|\mathbf{x}_i - \phi(\psi(\mathbf{x}))\|^2$, is as small as possible. Even if ψ is defined using the ℓ_1 regularization, this is still a computationally hard problem (similar to the k-means problem). However, several heuristic search algorithms may give reasonably good solutions. These algorithms are beyond the scope of this book.

25.4 Summary

Many machine learning algorithms take the feature representation of instances for granted. Yet the choice of representation requires careful attention. We discussed approaches for feature selection, introducing filters, greedy selection algorithms, and sparsity-inducing norms. Next we presented several examples for feature transformations and demonstrated their usefulness. Last, we discussed feature learning, and in particular dictionary learning. We have shown that feature selection, manipulation, and learning all depend on some prior knowledge on the data.

25.5 Bibliographic Remarks

Guyon & Elisseeff (2003) surveyed several feature selection procedures, including many types of filters.

Forward greedy selection procedures for minimizing a convex objective subject to a polyhedron constraint date back to the Frank-Wolfe algorithm (Frank & Wolfe 1956). The relation to boosting has been studied by several authors, including, (Warmuth, Liao & Ratsch 2006, Warmuth, Glocer & Vishwanathan 2008, Shalev-Shwartz & Singer 2008). Matching pursuit has been studied in the signal processing community (Mallat & Zhang 1993). Several papers analyzed greedy selection methods under various conditions. See, for example, Shalev-Shwartz, Zhang & Srebro (2010) and the references therein.

The use of the ℓ_1-norm as a surrogate for sparsity has a long history (e.g. Tibshirani (1996) and the references therein), and much work has been done on understanding the relationship between the ℓ_1-norm and sparsity. It is also closely related to compressed sensing (see Chapter 23). The ability to sparsify low ℓ_1 norm predictors dates back to Maurey (Pisier 1980-1981). In Section 26.4 we also show that low ℓ_1 norm can be used to bound the estimation error of our predictor.

Feature learning and dictionary learning have been extensively studied recently in the context of deep neural networks. See, for example, (Lecun & Bengio 1995, Hinton et al. 2006, Ranzato et al. 2007, Collobert & Weston 2008, Lee et al. 2009, Le et al. 2012, Bengio 2009) and the references therein.

25.6 Exercises

1. Prove the equality given in Equation (25.1). *Hint*: Let a^*, b^* be minimizers of the left-hand side. Find a, b such that the objective value of the right-hand side is smaller than that of the left-hand side. Do the same for the other direction.
2. Show that Equation (25.7) is the solution of Equation (25.6).
3. **AdaBoost as a Forward Greedy Selection Algorithm:** Recall the AdaBoost algorithm from Chapter 10. In this section we give another interpretation of AdaBoost as a forward greedy selection algorithm.
   - Given a set of m instances $\mathbf{x}_1, \ldots, \mathbf{x}_m$, and a hypothesis class $\mathcal{H}$ of finite VC dimension, show that there exist d and $h_1, \ldots, h_d$ such that for every $h \in \mathcal{H}$ there exists $i \in [d]$ with $h_i(\mathbf{x}_j) = h(\mathbf{x}_j)$ for every $j \in [m]$.
   - Let $R(\mathbf{w})$ be as defined in Equation (25.3). Given some $\mathbf{w}$, define $f_{\mathbf{w}}$ to be the function

$$f_{\mathbf{w}}(\cdot) = \sum_{i=1}^{d} w_i h_i(\cdot).$$

Let $\mathbf{D}$ be the distribution over $[m]$ defined by

$$D_i = \frac{\exp(-y_i f_{\mathbf{w}}(\mathbf{x}_i))}{Z},$$

where Z is a normalization factor that ensures that $\mathbf{D}$ is a probability vector. Show that

$$\frac{\partial R(\mathbf{w})}{w_j} = -\sum_{i=1}^{m} D_i y_i h_j(\mathbf{x}_i).$$

Furthermore, denoting $\epsilon_j = \sum_{i=1}^{m} D_i \mathbb{1}_{[h_j(\mathbf{x}_i) \neq y_i]}$, show that

$$\frac{\partial R(\mathbf{w})}{w_j} = 2\epsilon_j - 1.$$

Conclude that if $\epsilon_j \leq 1/2 - \gamma$ then $\left|\frac{\partial R(\mathbf{w})}{w_j}\right| \geq \gamma/2$.

- Show that the update of AdaBoost guarantees $R(\mathbf{w}^{(t+1)}) - R(\mathbf{w}^{(t)}) \leq \log(\sqrt{1-4\gamma^2})$. *Hint*: Use the proof of Theorem 10.2.