8 | Bias and Fairness

Science and everyday life cannot
and should not be separated. – Rosalind Franklin

Learning Objectives:

•

At the end of Chapter 1, you saw the "Russian Tank" example of a biased data set leading to a classifier that *seemed* like it was doing well, but was really relying on some artifact of the data collection process. As machine learning algorithms have a greater and greater impact on the real world, it is crucially important to ensure that they are making decisions based on the "right" aspects of the input, rather than exploiting arbitrary idiosyncracies of a particular training set.

For the rest of this chapter, we will consider two real world examples of bias issues that have had significant impact: the effect of gender in speech recognition systems and the effect of race in predicting criminal recidivism (i.e., will a convicted criminal commit further crimes if released).[1] The gender issue is that early speech recognition systems in cars failed to recognized the voices of many people who were not men. The race issue is that a specific recidivism predictor based on standard learning algorithms was biased against minorities.

Dependencies: Chapter 1,Chapter 3,Chapter 4,Chapter 5

[1] See Autoblog and ProPublica for press coverage of these two issues.

8.1 Train/Test Mismatch

One of the most common issues in bias is a mismatch between the training distribution and the testing distribution. In the running example of speech recognition failing to work on many non-men speakers, a large part of this happened because most of the training data on which the speech recognition system was trained was spoken by men. The learning algorithm learned—very well—how to recognize men's speech, but its accuracy dropped significantly when faced with a different distribution of test data.

To understand why this happens, recall the Bayes Optimal classifier from Chapter 2. This was the classifier than magically know the data distribution $\mathcal{D}$, and then when presented with an example x predicted $\text{argmax}_y \mathcal{D}(x, y)$. This was optimal, but *only* because $\mathcal{D}$ was the correct distribution. Even if one has access to the true distribution for male speakers, say $\mathcal{D}^{(\text{male})}$, the Bayes Optimal classifier under

$\mathcal{D}^{(\text{male})}$ will generally *not* be optimal under the distribution for any other gender.

Another example occurs in sentiment analysis. It is common to train sentiment analysis systems on data collected from reviews: product reviews, restaurant reviews, movie reviews, etc. This data is convenient because it includes both text (the review itself) and also a rating (typically one to five stars). This yields a "free" dataset for training a model to predict rating given text. However, one should be wary when running such a sentiment classifier on text *other than* the types of reviews it was trained on. One can easily imagine that a sentiment analyzer trained on movie reviews may not work so well when trying to detect sentiment in a politician's speech. Even moving between one type of product and another can fail wildly: "very small" typically expresses positive sentiment for USB drives, but not for hotel rooms.

The issue of **train/test mismatch** has been widely studied under many different names: **covariate shift** (in statistics, the input features are called "covariates"), **sample selection bias** and **domain adaptation** are the most common. We will refer to this problem simply as **adaptation**. The adaptation challenge is: given training data from one "old" distribution, learn a classifier that does a good job on *another* related, but different, "new" distribution.

It's important to recognize that in general, adaptation is impossible. For example, even if the task remains the same (positive/negative sentiment), if the old distribution is text reviews and the new distribution is images of text reviews, it's hard to imagine that doing well on the old distribution says anything about the new. As a less extreme example, if the old distribution is movie reviews in English and the new distribution is movie reviews in Mandarin, it's unlikely that adaptation will be easy.

These examples give rise to the tension in adaptation problems.

1. What does it mean for two distributions to be related? We might believe that "reviews of DVDs" and "reviews of movies" are "highly related" (though somewhat different: DVD reviews often discuss bonus features, quality, etc.); while "reviews of hotels" and "reviews of political policies" are "less related." But how can we formalize this?

2. When two distributions *are* related, how can we build models that effectively share information between them?

8.2 *Unsupervised Adaptation*

The first type of adaptation we will cover is **unsupervised adaptation**. The setting is the following. There are two distributions, $\mathcal{D}^{\text{old}}$ and $\mathcal{D}^{\text{new}}$. We have *labeled* training data from $\mathcal{D}^{\text{old}}$, say $(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)$ totalling N examples. We also have M many *unlabeled examples* from $\mathcal{D}^{\text{new}}$: $\boldsymbol{z}_1, \dots, \boldsymbol{z}_M$. We assume that the examples live in the same space, $\mathbb{R}^D$. This is called *unsupervised* adaptation because we do not have access to any labels in the new distribution.[2]

[2] Sometimes this is called **semi-supervised adaptation** in the literature.

Our goal is to learn a classifier f that achieves low expected loss *under the new distribution*, $\mathcal{D}^{new}$. The challenge is that we do not have access to any labeled data from $\mathcal{D}^{\text{new}}$. As a warm-up, let's suppose that we have a black box machine learning algorithm $\mathcal{A}$ that takes in *weighted* examples and produces a classifier. At the very least, this can be achieved using either undersampling or oversampling (see Section 6.1). We're going to attempt to *reweigh* the (old distribution) labeled examples based on how similar they are to the new distribution. This is justified using the **importance sampling** trick for switching expectations:

$$
\begin{aligned}
&\text{test loss} && && (8.1)\\
&= \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}^{\text{new}}}\left[\ell(y, f(\boldsymbol{x}))\right] && \text{definition} && (8.2)\\
&= \sum_{(\boldsymbol{x},y)} \mathcal{D}^{\text{new}}(\boldsymbol{x}, y)\ell(y, f(\boldsymbol{x})) && \text{expand expectation} && (8.3)\\
&= \sum_{(\boldsymbol{x},y)} \mathcal{D}^{\text{new}}(\boldsymbol{x}, y)\frac{\mathcal{D}^{\text{old}}(\boldsymbol{x}, y)}{\mathcal{D}^{\text{old}}(\boldsymbol{x}, y)}\ell(y, f(\boldsymbol{x})) && \text{times one} && (8.4)\\
&= \sum_{(\boldsymbol{x},y)} \mathcal{D}^{\text{old}}(\boldsymbol{x}, y)\frac{\mathcal{D}^{\text{new}}(\boldsymbol{x}, y)}{\mathcal{D}^{\text{old}}(\boldsymbol{x}, y)}\ell(y, f(\boldsymbol{x})) && \text{rearrange} && (8.5)\\
&= \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}^{\text{old}}}\left[\frac{\mathcal{D}^{\text{new}}(\boldsymbol{x}, y)}{\mathcal{D}^{\text{old}}(\boldsymbol{x}, y)}\ell(y, f(\boldsymbol{x}))\right] && \text{definition} && (8.6)
\end{aligned}
$$

What we have achieved here is rewriting the test loss, which is an expectation over $\mathcal{D}^{\text{new}}$, as an expectation over $\mathcal{D}^{\text{old}}$ instead.[3] This is useful because we have access to labeled examples from $\mathcal{D}^{\text{old}}$ but not $\mathcal{D}^{\text{new}}$. The implicit suggested algorithm by this analysis to to train a classifier using our learning algorithm $\mathcal{A}$, but where each training example $(\boldsymbol{x}_n, y_n)$ is weighted according to the ratio $\mathcal{D}^{\text{new}}(\boldsymbol{x}_n, y_n)/\mathcal{D}^{\text{old}}(\boldsymbol{x}_n, y_n)$. Intuitively, this makes sense: the classifier is being told to pay more attention to training examples that have high probability under the new distribution, and less attention to training that have low probability under the new distribution.

[3] In this example, we assumed a discrete distribution; if the distributions are continuous, the sums are simply replaced with integrals.

The problem with this approach is that we do not have access to $\mathcal{D}^{\text{new}}$ or $\mathcal{D}^{\text{old}}$, so we cannot compute this ratio and therefore cannot run this algorithm. One approach to this problem is to try to explicitly estimate these distributions, a task known as **density estimation**. This is an *incredibly* difficult problem; far harder than the original adaptation problem.

A solution to this problem is to try to estimate the *ratio* directly, rather than separately estimating the two probability distributions[4]. The key idea is to think of the adaptation as follows. All examples are drawn according to some fixed base distribution $\mathcal{D}^{\text{base}}$. Some of these are selected to go into the new distribution, and some of them are selected to go into the old distribution. The mechanism for deciding which ones are kept and which are thrown out is governed by a *selection variable*, which we call s. The choice of selection-or-not, s, is based *only* on the input example $\boldsymbol{x}$ and not on it's label. In particular, we define:

[4] Bickel et al. 2007

? What could go wrong if s got to look at the label, too?

$$\mathcal{D}^{\text{old}}(\boldsymbol{x},y) \propto \mathcal{D}^{\text{base}}(\boldsymbol{x},y)p(s=1 \mid \boldsymbol{x}) \tag{8.7}$$

$$\mathcal{D}^{\text{new}}(\boldsymbol{x},y) \propto \mathcal{D}^{\text{base}}(\boldsymbol{x},y)p(s=0 \mid \boldsymbol{x}) \tag{8.8}$$

That is, the probability of drawing some pair $(\boldsymbol{x},y)$ in the old distribution is proportional to the probability of first drawing that example according to the base distribution, and then the probability of selecting that particular example into the old distribution. If we can successfully estimate $p(s=1 \mid \boldsymbol{x})$, then the ratio that we sought, then we can compute the importance ratio as:

$$\begin{aligned}
\frac{\mathcal{D}^{\text{new}}(\boldsymbol{x},y)}{\mathcal{D}^{\text{old}}(\boldsymbol{x},y)} &= \frac{\frac{1}{Z^{\text{new}}}\mathcal{D}^{\text{base}}(\boldsymbol{x},y)p(s=0 \mid \boldsymbol{x})}{\frac{1}{Z^{\text{old}}}\mathcal{D}^{\text{base}}(\boldsymbol{x},y)p(s=1 \mid \boldsymbol{x})} && \text{definition} \quad (8.9)\\
&= \frac{\frac{1}{Z^{\text{new}}}p(s=0 \mid \boldsymbol{x})}{\frac{1}{Z^{\text{old}}}p(s=1 \mid \boldsymbol{x})} && \text{cancel base} \quad (8.10)\\
&= Z\frac{p(s=0 \mid \boldsymbol{x})}{p(s=1 \mid \boldsymbol{x})} && \text{consolidate} \quad (8.11)\\
&= Z\frac{1-p(s=1 \mid \boldsymbol{x})}{p(s=1 \mid \boldsymbol{x})} && \text{binary selection} \quad (8.12)\\
&= Z\left[\frac{1}{p(s=1 \mid \boldsymbol{x})}-1\right] && \text{rearrange} \quad (8.13)
\end{aligned}$$

This means that if we can estimate the selection probability $p(s=1 \mid \boldsymbol{x})$, we're done. We can therefore use $1/p(s=1 \mid \boldsymbol{x}_n)-1$ as an example weight on example $(\boldsymbol{x}_n,y_n)$ when feeding these examples into our learning algorithm $\mathcal{A}$.

? As a check: make sure that these weights are always non-negative. Furthermore, why is it okay to ignore the Z factor?

The remaining question is how to estimate $p(s=1 \mid \boldsymbol{x}_n)$. Recall that $s=1$ denotes the case that $\boldsymbol{x}$ is selected into the old distribution and $s=0$ denotes the case that $\boldsymbol{x}$ is selected into the new distribution. This means that predicting s is *exactly* a binary classification problem, where the "positive" class is the set of N examples from the old distribution and the "negative" class is the set of M examples from the new distribution.

Algorithm 23 SelectionAdaptation($\langle(x_n, y_n)\rangle_{n=1}^N$, $\langle z_m\rangle_{m=1}^M$, $\mathcal{A}$)

```
1: $D^{dist} \leftarrow \langle(x_n, +1)\rangle_{n=1}^N \bigcup \langle(z_m, -1)\rangle_{m=1}^M$    // assemble data for distinguishing
                                                                                              // between old and new distributions
2: $\hat{p} \leftarrow$ train logistic regression on $D^{\text{dist}}$
3: $D^{weighted} \leftarrow \left\langle (x_n, y_n, \frac{1}{\hat{p}(x_n)} - 1) \right\rangle_{n=1}^N$    // assemble weight classification
                                                                                              // data using selector
4: return $\mathcal{A}(D^{weighted})$                                                          // train classifier
```

This analysis gives rise to Algorithm 8.2, which consists of essentially two steps. The first is to train a logistic regression classifier[5] to distinguish between old and new distributions. The second is to use that classifier to produce weights on the labeled examples from the old distribution and then train whatever learning algorithm you wish on that.

[5] The use of logistic regression is arbitrary: it need only be a classification algorithm that can produce probabilities.

In terms of the questions posed at the beginning of this chapter, this approach to adaptation measures *nearness* of the two distributions by the degree to which the selection probability is constant. In particular, if the selection probability is independent of x, then the two distributions are identical. If the selection probabilities vary sigificantly as x changes, then the two distributions are considered very different. **More generally, if it is *easy* to train a classifier to distinguish between the old and new distributions, then they are very different.**

In the case of speech recognition failing as a function of gender, a core issue is that speech from men was massively over-represented in the training data but not the test data. When the selection logistic regression is trained, it is likely to say "old" on speech from men and "new" on other speakers, thereby *downweighting* the significance of male speakers and *upweighting* the significance of speakers of other genders on the final learned model. This would (hopefully) address many of the issues confounding that system.

? Make up percentages for fraction of speakers who are male in the old and new distributions; estimate (you'll have to make some assumptions) what the importance weights would look like in this case.

8.3 Supervised Adaptation

Unsupervised adaptation is very challenging because we never get to see try labels in the new distribution. In many ways, unsupervised adaptation attempts to *guard against* bad things happening. That is, if an old distribution training example looks very unlike the new distribution, it (and consequently it's features) are downweighted so much as to be ignored. In supervised adaptation, we can hope for more: we can hope to actually do *better* on the new distribution than the old because we have labeled data there.

The typical setup is similar to the unsupervised case. There are

Algorithm 24 EASYADAPT($\langle (x_n^{(old)}, y_n^{(old)}) \rangle_{n=1}^N$, $\langle (x_m^{(new)}, y_m^{(new)}) \rangle_{m=1}^M$, $\mathcal{A}$)

```
1: $D \leftarrow \left\langle (\langle x_n^{(old)}, x_n^{(old)}, \mathbf{0} \rangle, y_n^{(old)}) \right\rangle_{n=1}^N \cup \left\langle (\langle x_m^{(new)}, \mathbf{0}, x_m^{(new)} \rangle, y_m^{(new)}) \right\rangle_{m=1}^M$   // union
                                                                                         // of transformed data
2: return $\mathcal{A}(D)$                                                               // train classifier
```

two distributions, $\mathcal{D}^{\text{old}}$ and $\mathcal{D}^{\text{new}}$, and our goal is to learn a classifier that does well in expectation on $\mathcal{D}^{\text{new}}$. However, now we have labeled data from both distributions: N labeled examples from $\mathcal{D}^{\text{old}}$ and M labeled examples from $\mathcal{D}^{\text{new}}$. Call them $\langle x_n^{(old)}, y_n^{(old)} \rangle_{n=1}^N$ from $\mathcal{D}^{\text{old}}$ and $\langle x_m^{(new)}, y_m^{(new)} \rangle_{m=1}^M$ from $\mathcal{D}^{\text{new}}$. Again, suppose that both $x_n^{(old)}$ and $x_m^{(new)}$ both live in $\mathbb{R}^D$.

One way of answer the question of "how do we share information between the two distributions" is to say: when the distributions agree on the value of a feature, let them share it, but when they disagree, allow them to learn separately. For instance, in a sentiment analysis task, $\mathcal{D}^{\text{old}}$ might be reviews of electronics and $\mathcal{D}^{\text{new}}$ might be reviews of hotel rooms. In both cases, if the review contains the word "awesome" then it's probably a positive review, regardless of which distribution we're in. We would want to *share* this information across distributions. On the other hand, "small" might be positive in electronics and negative in hotels, and we would like the learning algorithm to be able to learn separate information for that feature.

A very straightforward way to accomplish this is the **feature augmentation** approach[6]. This is a simple preprocessing step after which one can apply any learning algorithm. The idea is to create three versions of every feature: one that's shared (for words like "awesome"), one that's old-distribution-specific and one that's new-distribution-specific. The mapping is:

[6] Daumé III 2007

$$\begin{array}{cccccc} & & \text{shared} & \text{old-only} & \text{new-only} & \\ x_n^{(old)} & \mapsto \Big\langle & x_n^{(old)}, & x_n^{(old)}, & \underbrace{0, 0, \dots, 0}_{D\text{-many}} & \Big\rangle \end{array} \quad (8.14)$$

$$x_m^{(new)} \mapsto \Big\langle \; x_m^{(new)}, \; \underbrace{0, 0, \dots, 0}_{D\text{-many}}, \; x_m^{(new)} \; \Big\rangle \quad (8.15)$$

Once you've applied this transformation, you can take the union of the (transformed) old and new labeled examples and feed the entire set into your favorite classification algorithm. That classification algorithm can then choose to share strength between the two distributions by using the "shared" features, if possible; or, if not, it can learn distribution-specific properties on the old-only or new-only parts. This is summarized in Algorithm 8.3.

Note that this algorithm can be *combined* with the instance weighting (unsupervised) learning algorithm. In this case, the logistic regression separator should be trained on the *untransformed* data, and then weights can be used exactly as in Algorithm 8.2. This is particularly useful if you have access to *way* more old distribution data than new distribution data, and you don't want the old distribution data to "wash out" the new distribution data.

? Why is it crucial that the separator be trained on the untransformed data?

Although this approach is general, it is most effective when the two distributions are "not too close but not too far":

- If the distributions are too far, and there's little information to share, you're probably better off throwing out the old distribution data and training just on the (untransformed) new distribution data.
- If the distributions are too close, then you might as well just take the union of the (untransformed) old and new distribution data, and training on that.

In general, the interplay between how far the distributions are and how much new distribution data you have is complex, and you should always try "old only" and "new only" and "simple union" as baselines.

8.4 Fairness and Data Bias

Almost any data set in existence is biased in some way, in the sense that it captures an imperfect view of the world. The degree to which this bias is obvious can vary greatly:

- There might be obvious bias in the labeling. For instance, in criminology, learning to predict sentence lengths by predicting the sentences assigned by judges will simply learn to reproduce whatever bias already exists in judicial sentencing.
- There might be sample selection bias, as discussed early. In the same criminology example, the only people for which we have training data are those that have already been arrested, charged with and convicted of a crime; these processes are inherantly biased, and so any model learned on this data may exhibit similar biases.
- The task itself might be biased because the designers had blindspots. An intelligent billboard that predicts the gender of the person walking toward it so as to show "better" advertisements may be trained as a binary classifier between male/female and thereby excludes anyone who does not fall in the gender binary. A similar example holds for a classifier that predicts political affiliation in

the US as Democrat/Republican, when in fact there are far more possibilities.

- There might be bias in the features or model structure. Machine translation systems often exhibit incorrect gender stereotypes[7] because the relevant context, which would tell them the correct gender, is not encoded in the feature representation. Alone, or coupled with biased data, this leads to errors.
- The loss function may favor certain types of errors over others. You've already seen such an example: using zero/one loss on a highly imbalanced class problem will often lead to a learned model that ignores the minority class entirely.
- A deployed system creates feedback loops when it begins to consume it's own output as new input. For instance, once a spam filter is in place, spammers will adjust their strategies, leading to distribution shift in the inputs. Or a car guidance system for predicting which roads are likely to be unoccupied may find those roads now occupied by other cars that it directed there.

[7] For instance, many languages have verbal marking that agrees with the gender of the subject. In such cases, "the doctor treats . . . " puts masculine markers on the translation of "treats" but "the nurse treats . . . " uses feminine markers.

These are all difficult questions, none of which has easy answers. Nonetheless, it's important to be aware of how our systems may fail, especially as they take over more and more of our lives. Most computing, engineering and mathematical societies (like the ACM, IEEE, BCS, etc.) have codes of ethics, all of which include a statement about avoiding harm; the following is taken from the ACM code[8]:

[8] ACM Code of Ethics

> To minimize the possibility of indirectly harming others, computing professionals must minimize malfunctions by following generally accepted standards for system design and testing. Furthermore, it is often necessary to assess the social consequences of systems to project the likelihood of any serious harm to others.

In addition to ethical questions, there are often related *legal* questions. For example, US law prohibits discrimination by race and gender (among other "protected attributes") in processes like hiring and housing. The current legal mechanism for measuring discimination is **disparate impact** and the **80% rule**. Informally, the 80% rule says that your rate of hiring women (for instance) must be at least 80% of your rate of hiring men. Formally, the rule states:

$$\Pr(y = +1 \mid G \neq \text{male}) \geq 0.8 \times \Pr(y = +1 \mid G = \text{male}) \tag{8.16}$$

Of course, gender/male can be replaced with any other protected attribute.

One non-solution to the disparate impact problem is to simply throw out protected attributes from your dataset. Importantly, yet unfortunately, this is **not** sufficient. Many other features often correlate strongly with gender, race, and other demographic information, and just because you've removed *explicit* encodings of these factors does not imply that a model trained as such would satisfy the 80% rule. A natural question to ask is: can we build machine learning algorithms that have high accuracy but simultaneously avoid disparate impact?

Disparate impact is an imperfect measure of (un)fairness, and there are alternatives, each with it's own pros and cons[9]. All of these rely on predefined categories of protected attributes, and a natural question is where these come from if not governmental regulation. Regardless of the measure you choose, the most important thing to keep in mind is that just because something comes from data, or is algorithmically implemented, does not mean it's fair.

[9] Friedler et al. 2016, Hardt et al. 2016

8.5 How Badly can it Go?

To help better understand how badly things can go when the distribution over inputs changes, it's worth thinking about how to analyze this situation formally. Suppose we have two distributions $\mathcal{D}^{\text{old}}$ and $\mathcal{D}^{\text{new}}$, and let's assume that the only thing that's different about these is the distribution they put on the inputs $\mathcal{X}$ and not the outputs $\mathcal{Y}$. (We will return later to the usefulness of this assumption.) That is: $\mathcal{D}^{\text{old}}(\boldsymbol{x}, y) = \mathcal{D}^{\text{old}}(\boldsymbol{x})\mathcal{D}(y \mid \boldsymbol{x})$ and $\mathcal{D}^{\text{new}}(\boldsymbol{x}, y) = \mathcal{D}^{\text{new}}(\boldsymbol{x})\mathcal{D}(y \mid \boldsymbol{x})$, where $\mathcal{D}(y \mid \boldsymbol{x})$ is shared between them.

Let's say that you've learned a classifier f that does really well on $\mathcal{D}^{\text{old}}$ and achieves some *test error* of $\epsilon^{(\text{old})}$. That is:

$$\epsilon^{(\text{old})} = \mathbb{E}_{\substack{\boldsymbol{x} \sim \mathcal{D}^{\text{old}} \\ y \sim \mathcal{D}(\cdot \mid \boldsymbol{x})}} \left[\mathbf{1}[f(\boldsymbol{x}) \neq y] \right] \tag{8.17}$$

The question is: how badly can f do on the new distribution?

We can calculate this directly.

$$\begin{aligned}
&\epsilon^{(\text{new})} \\
&= \mathbb{E}_{\substack{\boldsymbol{x} \sim \mathcal{D}^{\text{new}} \\ y \sim \mathcal{D}(\cdot \mid \boldsymbol{x})}} \left[\mathbf{1}[f(\boldsymbol{x}) \neq y] \right] && (8.18) \\
&= \int_{\mathcal{X}} \int_{\mathcal{Y}} \mathcal{D}^{\text{new}}(\boldsymbol{x}) \mathcal{D}(y \mid \boldsymbol{x}) \mathbf{1}[f(\boldsymbol{x}) \neq y] \mathrm{d}y \mathrm{d}\boldsymbol{x} && \text{def. of } \mathbb{E} \quad (8.19) \\
&= \int_{\mathcal{X}} \left(\mathcal{D}^{\text{new}}(\boldsymbol{x}) - \mathcal{D}^{\text{old}}(\boldsymbol{x}) + \mathcal{D}^{\text{old}}(\boldsymbol{x}) \right) \times \\
&\qquad \int_{\mathcal{Y}} \mathcal{D}(y \mid \boldsymbol{x}) \mathbf{1}[f(\boldsymbol{x}) \neq y] \mathrm{d}y \mathrm{d}\boldsymbol{x} && \text{add zero} \quad (8.20) \\
&= \epsilon^{(\text{old})} + \\
&\qquad \int_{\mathcal{X}} \left(\mathcal{D}^{\text{new}}(\boldsymbol{x}) - \mathcal{D}^{\text{old}}(\boldsymbol{x}) \right) \times \\
&\qquad \int_{\mathcal{Y}} \mathcal{D}(y \mid \boldsymbol{x}) \mathbf{1}[f(\boldsymbol{x}) \neq y] \mathrm{d}y \mathrm{d}\boldsymbol{x} && \text{def. } \epsilon^{\text{old}} \quad (8.21)
\end{aligned}$$

$$\leq \epsilon^{(\text{old})} + \int_{\mathcal{X}} \left| \mathcal{D}^{\text{new}}(\boldsymbol{x}) - \mathcal{D}^{\text{old}}(\boldsymbol{x}) \right| \left(1\right) \mathrm{d}\boldsymbol{x} \qquad \text{worst case } \int_{\mathcal{Y}} \tag{8.22}$$

$$= \epsilon^{(\text{old})} + 2 \left| \mathcal{D}^{\text{new}} - \mathcal{D}^{\text{old}} \right|_{\text{Var}} \qquad \text{def. } |\cdot|_{\text{Var}} \tag{8.23}$$

Here, $|\cdot|_{\text{Var}}$ is the **total variation distance** (or **variational distance**) between two probability distributions, defined as:

$$|P - Q|_{\text{Var}} = \sup_{e} |P(e) - Q(e)| = \frac{1}{2} \int_{\mathcal{X}} |P(\boldsymbol{x}) - Q(\boldsymbol{x})| \, \mathrm{d}\boldsymbol{x} \tag{8.24}$$

Is a standard measure of dissimilarity between probability distributions. In particular, the variational distance is the largest difference in probability that P and Q assign to the same event (in this case, the event is an example $\boldsymbol{x}$).

The bound tells us that we might not be able to hope for very good error on the new distribution, even if we have very good error on the old distribution, when $\mathcal{D}^{\text{old}}$ and $\mathcal{D}^{\text{new}}$ are very different (assign very different probabilities to the same event). Of course, this is an *upper bound*, and possibly a very loose one.

The second observation is that we barely used the assumption that $\mathcal{D}^{\text{new}}$ and $\mathcal{D}^{\text{old}}$ share the same label distribution $\mathcal{D}(y \mid \boldsymbol{x})$ in the above analysis. (In fact, as an exercise, repeat the analysis *without* this assumption. It will still go through.) In general, this assumption buys us very little. In an extreme case, $\mathcal{D}^{\text{new}}$ and $\mathcal{D}^{\text{old}}$ can essentially "encode" which distribution a given $\boldsymbol{x}$ came from in one of the features. Once the origin distribution is completely encoded in a feature, then $\mathcal{D}(y \mid \boldsymbol{x})$ could look at the encoding, and completely flip the label based on which distribution it's from. How could $\mathcal{D}^{\text{new}}$ and $\mathcal{D}^{\text{old}}$ encode the distribution? One way would be to set the 29th decimal digit of feature 1 to an odd value in the old distribution and an even value in the new distribution. This tiny change will be essentially imperceptible if one looks at the data, but would give $\mathcal{D}(y \mid \boldsymbol{x})$ enough power to make our lives miserable.

If we want a way out of this dilemma, we need more technology. The core idea is that if we're learning a function f from some hypothesis class $\mathcal{F}$, and this hypothesis class isn't rich enough to peek at the 29th decimal digit of feature 1, then perhaps things are not as bad as they could be. This motivates the idea of looking at a measure of distance between probability distributions that *depends on the hypothesis class*. A popular measure is the **$d_{\mathcal{A}}$-distance** or the **discrepancy**. The discrepancy measure distances between probability distributions based on how much two function f and f' in the hypothesis class can disagree on their labels. Let:

$$\epsilon_P(f, f') = \mathbb{E}_{\boldsymbol{x} \sim P}\left[\mathbf{1}[f(\boldsymbol{x}) \neq f'(\boldsymbol{x})]\right] \tag{8.25}$$

You can think of $\epsilon_P(f, f')$ as the *error* of f' when the ground truth is given by f, where the error is taken with repsect to examples drawn from P. Given a hypothesis class $\mathcal{F}$, the discrepancy between P and Q is defined as:

$$d_{\mathcal{A}}(P, Q) = \max_{f, f' \in \mathcal{F}} \left| \epsilon_P(f, f') - \epsilon_Q(f, f') \right| \tag{8.26}$$

The discrepancy very much has the flavor of a classifier: if you think of f as providing the ground truth labeling, then the discrepancy is large whenever there exists a function f' that agrees strongly with f on distribution P but *disagrees* strongly with f on Q. This feels natural: if all functions behave similarly on P and Q, then the discrepancy will be small, and we also expect to be able to generalize across these two distributions easily.

One very attractive property of the discrepancy is that you can estimate it from finite *unlabeled* samples from $\mathcal{D}^{\text{old}}$ and $\mathcal{D}^{\text{new}}$. Although not obvious at first, the discrepancy is very closely related to a quantity we saw earlier in unsupervised adaptation: a classifier that distinguishes between $\mathcal{D}^{\text{old}}$ and $\mathcal{D}^{\text{new}}$. In fact, the discrepancy is precisely twice the *accuracy* of the best classifier from $\mathcal{H}$ at separating $\mathcal{D}^{\text{old}}$ from $\mathcal{D}^{\text{new}}$.

How does this work in practice? Exactly as in the section on unsupervised adaptation, we train a classifier to distinguish between $\mathcal{D}^{\text{old}}$ and $\mathcal{D}^{\text{new}}$. It needn't be a probabilistic classifier; any binary classifier is fine. This classifier, the "domain separator," will have some (heldout) accuracy, call it *acc*. The discrepancy is then exactly $d_{\mathcal{A}} = 2(acc - 0.5)$.

Intuitively the accuracy of the domain separator is a natural measure of how different the two distributions are. If the two distributions are identical, you shouldn't expect to get very good accuracy at separating them. In particular, you expect the accuracy to be around 0.5, which puts the discrepancy at zero. On the other hand, if the two distributions are really far apart, separation is easy, and you expect an accuracy of about 1, yielding a discrepancy also of about 1.

One can, in fact, prove a generalization bound—generalizing from finite samples from $\mathcal{D}^{\text{old}}$ to expected loss on $\mathcal{D}^{\text{new}}$—based on the discrepancy[10].

[10] Ben-David et al. 2007

Theorem 9 (Unsupervised Adaptation Bound)**.** *Given a fixed representation and a fixed hypothesis space $\mathcal{F}$, let $f \in \mathcal{F}$ and let $\epsilon^{(best)} = \min_{f^* \in \mathcal{F}} \frac{1}{2}\left[\epsilon^{(old)}(f^*) + \epsilon^{(new)}(f^*)\right]$, then, for all $f \in \mathcal{F}$:*

$$\underbrace{\epsilon^{(new)}(f)}_{\text{error on } \mathcal{D}^{new}} \leq \underbrace{\epsilon^{(old)}(f)}_{\text{error on } \mathcal{D}^{old}} + \underbrace{\epsilon^{(best)}}_{\text{minimal avg error}} + \underbrace{d_{\mathcal{A}}(\mathcal{D}^{old}, \mathcal{D}^{new})}_{\text{distance}} \tag{8.27}$$

In this bound, $\epsilon^{(\text{best})}$ denotes the error rate of the best possible classifier from $\mathcal{F}$, where the error rate is measured as the *average* error rate on $\mathcal{D}^{\text{new}}$ and $\mathcal{D}^{\text{old}}$; this term ensures that at least some good classifier exists that does well on both distributions.

The main practical use of this result is that it suggests a way to look for representations that are good for adaptation: on the one hand, we should try to get our training error (on $\mathcal{D}^{\text{old}}$) as low as possible; on the other hand, we want to make sure that it is *hard* to distinguish between $\mathcal{D}^{\text{old}}$ and $\mathcal{D}^{\text{new}}$ in this representation.

8.6 Further Reading

TODO further reading