Chapter 14. Data Privacy for Machine Learning

In this chapter, we introduce some aspects of data privacy as they apply to machine learning pipelines. Privacy-preserving machine learning is a very active area of research that is just beginning to be incorporated into TensorFlow and other frameworks. We'll explain some of the principles behind the most promising techniques at the time of writing and show some practical examples for how they can fit into a machine learning pipeline.

We'll cover three main methods for privacy-preserving machine learning in this chapter: differential privacy, federated learning, and encrypted machine learning.

Data Privacy Issues

Data privacy is all about trust and limiting the exposure of data that people would prefer to keep private. There are many different methods for privacy-preserving machine learning, and in order to choose between them, you should try to answer the following questions:

- Who are you trying to keep the data private from?
- Which parts of the system can be private, and which can be exposed to the world?
- Who are the trusted parties that can view the data?

The answers to these questions will help you decide which of the methods described in this chapter best fits your use case.

Why Do We Care About Data Privacy?

Data privacy is becoming an important part of machine learning projects. There are many legal requirements surrounding user privacy, such as the EU's General Data Protection Regulation (GDPR), which went into effect in May 2018, and the California Consumer Privacy Act of January 2020.

There are ethical considerations around the use of personal data for machine learning, and users of products powered by ML are starting to care deeply about what happens to their data. Because machine learning has traditionally been hungry for data, and because many of the predictions made by machine learning models are based on personal data collected from users, machine learning is at the forefront of debates around data privacy.

At the time of writing, there's always a cost to privacy: adding privacy comes with a cost in model accuracy, computation time, or both. At one extreme, collecting no data keeps an interaction completely private but is completely useless for machine learning. At the other extreme, knowing all the details about a person might endanger that person's privacy, but it allows us to make very accurate machine learning models. We're just now starting to see the development of privacy-preserving ML, in which privacy can be increased without such a large trade-off in model accuracy.

In some situations, privacy-preserving machine learning can help you use data that would otherwise be unavailable for training a machine learning model due to privacy concerns. It doesn't, however, give you free rein to do whatever you like with the data just because you use one of the methods in this chapter. You should discuss your plans with other stakeholders, for example, the data owners, privacy experts, and even your company's legal team.

The Simplest Way to Increase Privacy

Often, the default strategy for building a product powered by machine learning is to collect all the data possible and then decide afterward what is useful for training a machine learning model. Even though this is done with the user's consent, the simplest way to increase user privacy is to only collect the data that is necessary for the training of a particular model. In the case of structured data, fields such as name, gender, or race can simply be deleted. Text or image data can be processed to remove much personal information, such as deleting faces from images or names from text. However, in some cases this can reduce the utility of the data or make it impossible to train an accurate model. And if data on race and

gender is not collected, it's impossible to tell whether a model is biased against a particular group.

Control of what data is collected can also be passed to the user: consent to collect data can be made more nuanced than a simple opt-in or opt-out selection, and the user of a product can specify exactly what data may be collected about them. This raises design challenges: should users who provide less data receive less accurate predictions than the users who contribute more data? How do we track consent through machine learning pipelines? How do we measure the privacy impact of a single feature in our models? These are all questions that need more discussion in the machine learning community.

What Data Needs to Be Kept Private?

In machine learning pipelines, data is often collected from people, but some data has a higher need for privacy-preserving machine learning. Personally identifying information (PII) is data that can directly identify a single person, such as their name, email address, street address, ID number, and so on, and this needs to be kept private. PII can appear in free text, such as feedback comments or customer service data, not just when users are directly asked for this data. Images of people may also be considered PII in some circumstances. There are often legal standards around this—if your company has a privacy team, it's best to consult them before embarking on a project using this type of data.

Sensitive data also requires special care. This is often defined as data that could cause harm to someone if it were released, such as health data or proprietary company data (e.g., financial data). Care should be taken to ensure that this type of data is not leaked in the predictions of a machine learning model.

Another category is quasi-identifying data. Quasi-identifiers can uniquely identify someone if enough of them are known, such as location tracking or credit card transaction data. If several location points are known about the same person, this provides a unique trace that can be combined with other datasets to reidentify that person. In December 2019, the *New York Times* published an in-depth piece on reidentification using cellphone

data, which represents just one of several voices questioning the release of such data.

Differential Privacy

If we have identified a need for additional privacy in the machine learning pipeline, there are different methods that can help increase privacy while retaining as much data utility as possible. The first one we'll discuss is *differential privacy*.[1] Differential privacy (DP) is a formalization of the idea that a query or a transformation of a dataset should not reveal whether a person is in that dataset. It gives a mathematical measure of the privacy loss that a person experiences by being included in a dataset and minimizes this privacy loss through the addition of noise.

> *Differential privacy describes a promise, made by a data holder, or curator, to a data subject, and the promise is like this: "You will not be affected, adversely or otherwise, by allowing your data to be used in any study or analysis, no matter what other studies, datasets or information sources are available."*
>
> —Cynthia Dwork[2]

To put it another way, a transformation of a dataset that respects privacy should not change if one person is removed from that dataset. In the case of machine learning models, if a model has been trained with privacy in mind, then the predictions that a model makes should not change if one person is removed from the training set. DP is achieved by the addition of some form of noise or randomness to the transformation.

To give a more concrete example, one of the simplest ways of achieving differential privacy is the concept of randomized response, as shown in Figure 14-1. This is useful in surveys that ask sensitive questions, such as "Have you ever been convicted of a crime?" To answer this question, the person being asked flips a coin. If it comes up heads, they answer truthfully. If it comes up tails, they flip again and answer "Yes" if the coin comes up heads, and "No" if the coin comes up tails. This gives them deniability—they can say that they gave a random answer rather than a truthful answer. Because we know the probabilities for a coin flip, if we ask a

lot of people this question, we can calculate the proportion of people who have been convicted of a crime with reasonable accuracy. The accuracy of the calculation increases when larger numbers of people participate in the survey.

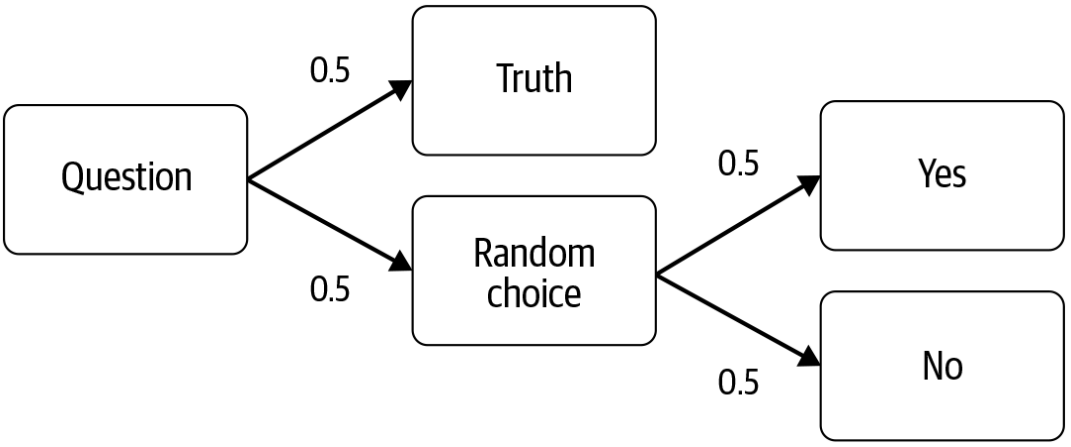

Figure 14-1. Randomized response flowchart

These randomized transformations are the key to DP.

ASSUME ONE TRAINING EXAMPLE PER PERSON

Throughout this chapter, for simplicity, we assume that each training example in a dataset is associated with or collected from one individual person.

Local and Global Differential Privacy

DP can be divided into two main methods: local and global DP. In local DP, noise or randomness is added at the individual level, as in the randomized response example earlier, so privacy is maintained between an individual and the collector of the data. In global DP, noise is added to a transformation on the entire dataset. The data collector is trusted with the raw data, but the result of the transformation does not reveal data about an individual.

Global DP requires us to add less noise compared to local DP, which leads to a utility or accuracy improvement of the query for a similar privacy guarantee. The downside is that the data collector must be trusted for global DP, whereas for local DP only individual users see their own raw data.

Epsilon, Delta, and the Privacy Budget

Probably the most common way of implementing DP is using $\epsilon - \delta$ (epsilon-delta) DP ϵ. When comparing the result of a randomized transformation on a dataset that includes one specific person with another result that does not contain that person, e^{ϵ} describes the maximum difference between the outcomes of these transformations. So, if ϵ is 0, both transformations return exactly the same result. If the value of ϵ is smaller, the probability that our transformations will return the same result is greater—a lower value of ϵ is more private because ϵ measures the strength of the privacy guarantee. If you query a dataset more than once, you need to sum the epsilons of each query to get your total privacy budget.

δ is the probability that ϵ does not hold, or the probability that an individual's data is exposed in the results of the randomized transformation. We generally set δ to be approximately the inverse of the population size: for a dataset containing 2,000 people, we would set δ to be 1/1,000.[3]

What value of epsilon should you choose? ϵ allows us to compare the privacy of different algorithms and approaches, but the absolute value that gives us "sufficient" privacy depends on the use case.[4]

To decide on a value to use for ϵ, it can be helpful to look at the accuracy of the system as ϵ is decreased. Choose the most private parameters possible while retaining acceptable data utility for the business problem. Alternatively, if the consequences of leaking data are very high, you may wish to set the acceptable values of ϵ and δ first, and then tune your other hyperparameters to get the best model accuracy possible. One weakness of $\epsilon - \delta$ DP is that ϵ is not easily interpretable. Other approaches are being developed to help with this, such as planting secrets within a model's training data and measuring how likely it is that they are exposed in a model's predictions.[5]

Differential Privacy for Machine Learning

If you want to use DP as part of your machine learning pipeline, there are a few current options for where it can be added, though we expect to see more in the future. First, DP can be included in a federated learning system (see "Federated Learning"), and this can use either global or local DP.

Second, the TensorFlow Privacy library is an example of global DP: raw data is available for model training.

A third option is the Private Aggregation of Teacher Ensembles (PATE) approach.[6] This is a data-sharing scenario: in the case that 10 people have labelled data, but you haven't, they train a model locally and each make a prediction on your data. A DP query is then performed to generate the final prediction on each example in your dataset so that you don't know which of the 10 models has made the prediction. A new model is then trained from these predictions—this model includes the information from the 10 hidden datasets in such a way that it's not possible to learn about those hidden datasets. The PATE framework shows how ϵ is being spent in this scenario.

Introduction to TensorFlow Privacy

TensorFlow Privacy (TFP) adds DP to an optimizer during model training. The type of DP used in TFP is an example of global DP: noise is added during training so that private data is not exposed in a model's predictions. This lets us offer the strong DP guarantee that an individual's data has not been memorized while still maximizing model accuracy. As shown in Figure 14-2, in this situation, the raw data is available to the trusted data store and model trainer, but the final predictions are untrusted.

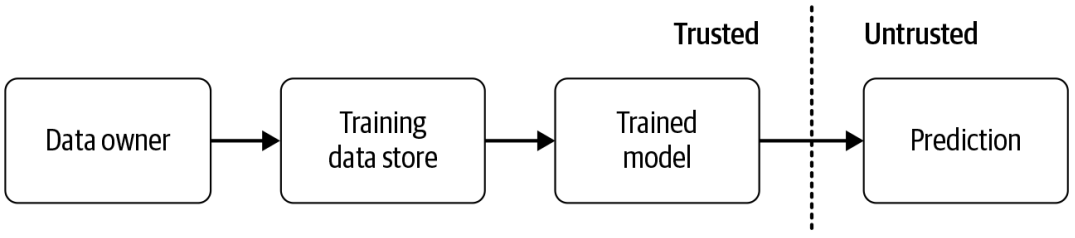

Figure 14-2. Trusted parties for DP

Training with a Differentially Private Optimizer

The optimizer algorithm is modified by adding random noise to the gradients at each training step. This compares the gradient's updates with or without each individual data point and ensures that it is not possible to tell whether a specific data point was included in the gradient update. In addition, gradients are clipped so that they do not become too large—this

limits the contribution of any one training example. As a nice bonus, this also helps prevent overfitting.

TFP can be installed with pip. At the time of writing, it requires TensorFlow version 1.X:

```
$ pip install tensorflow_privacy
```

We start with a simple `tf.keras` binary classification example:

```
import tensorflow as tf

model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

The differentially private optimizer requires that we set two extra hyper-parameters compared to a normal `tf.keras` model: the noise multiplier and the L2 norm clip. It's best to tune these to suit your dataset and measure their impact on ϵ:

```
NOISE_MULTIPLIER = 2
NUM_MICROBATCHES = 32 ❶
LEARNING_RATE = 0.01
POPULATION_SIZE = 5760 ❷
L2_NORM_CLIP = 1.5
BATCH_SIZE = 32 ❸
EPOCHS = 70
```

❶ The batch size must be exactly divisible by the number of microbatches.

❷ The number of examples in the training set.

❸ The population size must be exactly divisible by the batch size.

Next, initialize the differentially private optimizer:

```
from tensorflow_privacy.privacy.optimizers.dp_optimizer \
    import DPGradientDescentGaussianOptimizer

optimizer = DPGradientDescentGaussianOptimizer(
    l2_norm_clip=L2_NORM_CLIP,
    noise_multiplier=NOISE_MULTIPLIER,
    num_microbatches=NUM_MICROBATCHES,
    learning_rate=LEARNING_RATE)

loss = tf.keras.losses.BinaryCrossentropy(
    from_logits=True, reduction=tf.losses.Reduction.NONE)
```
❶

❶ Loss must be calculated on a per-example basis rather than over an entire minibatch.

Training the private model is just like training a normal `tf.keras` model:

```
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

model.fit(X_train, y_train,
            epochs=EPOCHS,
            validation_data=(X_test, y_test),
            batch_size=BATCH_SIZE)
```

Calculating Epsilon

Now, we calculate the differential privacy parameters for our model and our choice of noise multiplier and gradient clip:

```
from tensorflow_privacy.privacy.analysis import compute_dp_sgd_privacy


compute_dp_sgd_privacy.compute_dp_sgd_privacy(n=POPULATION_SIZE,
                                              batch_size=BATCH_SIZE,
                                              noise_multiplier=NOISE_MULTIPLIER,
                                              epochs=EPOCHS,
                                              delta=1e-4)
```
❶

❶ The value of delta is set to 1/the size of the dataset, rounded to the nearest order of magnitude.

TFP SUPPORTS ONLY TENSORFLOW 1.X

We show how you can convert the example project from previous chapters to a DP model in our GitHub repo. The differentially private optimizer is added into the `get_model` function from Chapter 6. However, this model can't be used in our TFX pipeline until TFP supports TensorFlow 2.X.

The final output of this calculation, the value of epsilon, tells us the strength of the privacy guarantee for our particular model. We can then explore how changing the L2 norm clip and noise multiplier hyperparameters discussed earlier affects both epsilon and our model accuracy. If the values of these two hyperparameters are increased, keeping all others fixed, epsilon will decrease (so the privacy guarantee becomes stronger). At some point, accuracy will begin to decrease and the model will stop being useful. This trade-off can be explored to get the strongest possible privacy guarantees while still maintaining useful model accuracy.

Federated Learning

Federated learning (FL) is a protocol where the training of a machine learning model is distributed across many different devices and the trained model is combined on a central server. The key point is that the raw data never leaves the separate devices and is never pooled in one place. This is very different from the traditional architecture of gathering a dataset in a central location and then training a model.

FL is often useful in the context of mobile phones with distributed data, or a user's browser. Another potential use case is in the sharing of sensitive data that is distributed across multiple data owners. For example, an AI startup may want to train a model to detect skin cancer. Images of skin cancer are owned by many hospitals, but they can't be centralized in one location due to privacy and legal concerns. FL lets the startup train a model without the data leaving the hospitals.

In an FL setup, each client receives the model architecture and some instructions for training. A model is trained on each client's device, and the weights are returned to a central server. This increases privacy slightly, in that it's more difficult for an interceptor to learn anything about a user from model weights than from raw data, but it doesn't provide any guarantee of privacy. The step of distributing the model training doesn't provide the user with any increased privacy from the company collecting the data because the company can often work out what the raw data would have been with a knowledge of the model architecture and the weights.

However, there is one more very important step to increase privacy using FL: the secure aggregation of the weights into the central model. There are a number of algorithms for doing this, but they all require that the central party has to be trusted to not attempt to inspect the weights before they are combined.

Figure 14-3 shows which parties have access to the personal data of users in the FL setting. It's possible for the company collecting the data to set up secure averaging such that they don't see the model weights that are returned from users. A neutral third party could also perform the secure aggregation. In this case, only the users would see their data.

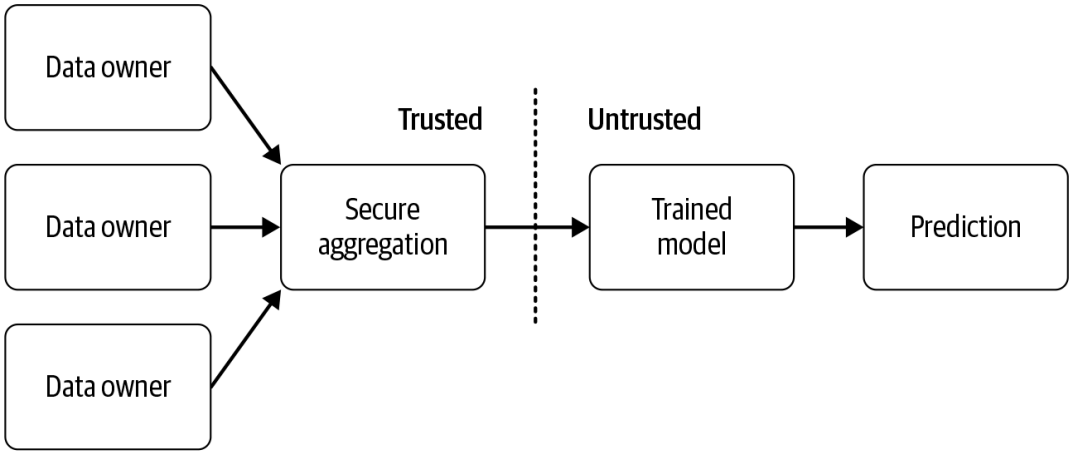

Figure 14-3. Trusted parties in federated learning

An additional privacy-preserving extension to FL is the incorporation of DP into this technique. In this situation, DP limits the amount of information that each user can contribute to the final model. Research has shown that the resulting models are almost as accurate as non-DP models if the number of users is large.[7] However, as yet, this hasn't been implemented for either TensorFlow or PyTorch.

An example of FL in production is Google's Gboard keyboard for Android mobile phones. Google is able to train a model to make better next-word predictions without learning anything about users' private messaging. FL is most useful in use cases that share the following characteristics:[8]

- The data required for the model can only be collected from distributed sources.
- The number of data sources is large.
- The data is sensitive in some way.
- The data does not require extra labelling—the labels are provided directly by the user and do not leave the source.
- Ideally, the data is drawn from close to identical distributions.

FL introduces many new considerations into the design of a machine learning system: for example, not all data sources may have collected new data between one training run and the next, not all mobile devices are powered on all the time, and so on. The data that is collected is often unbalanced and practically unique to each device. It's easiest to get sufficient data for each training run when the pool of devices is large. New secure infrastructure must be developed for any project using FL.[9]

Care must be taken to avoid performance issues on devices that train an FL model. Training can quickly drain the battery on a mobile device or cause large data usage, leading to expense for the user. Even though the processing power of mobile phones is increasing rapidly, they are still only capable of training small models, so more complex models should be trained on a central server.

Federated Learning in TensorFlow

TensorFlow Federated (TFF) simulates the distributed setup of FL and contains a version of stochastic gradient descent (SGD) that can calculate updates on distributed data. Conventional SGD requires that updates are computed on batches of a centralized dataset, and this centralized dataset doesn't exist in a federated setting. At the time of writing, TFF is mainly aimed at research and experimentation on new federated algorithms.

PySyft is an open source Python platform for privacy-preserving machine learning developed by the OpenMined organization. It contains an imple-

mentation of FL using secure multiparty computation (explained further in the following section) to aggregate data. It was originally developed to support PyTorch models, but a TensorFlow version has been released.

Encrypted Machine Learning

Encrypted machine learning is another area of privacy-preserving machine learning that's currently receiving a lot of attention from both researchers and practitioners. It leans on technology and research from the cryptographic community and applies these techniques to machine learning. The major methods that have been adopted so far are homomorphic encryption (HE) and secure multiparty computation (SMPC). There are two ways to use these techniques: encrypting a model that has already been trained on plain text data and encrypting an entire system (if the data must stay encrypted during training).

HE is similar to public-key encryption but differs in that data does not have to be decrypted before a computation is applied to it. The computation (such as obtaining predictions from a machine learning model) can be performed on the encrypted data. A user can provide their data in its encrypted form using an encryption key that is stored locally and then receive the encrypted prediction, which they can then decrypt to get the prediction of the model on their data. This provides privacy to the user because their data is not shared with the party who has trained the model.

SMPC allows several parties to combine data, perform a computation on it, and see the results of the computation on their own data without knowing anything about the data from the other parties. This is achieved by *secret sharing*, a process in which any single value is split into shares that are sent to separate parties. The original value can't be reconstructed from any share, but computations can still be carried out on each share individually. The result of the computations is meaningless until all the shares are recombined.

Both of these techniques come with a cost. At the time of writing, HE is rarely used for training machine learning models: it causes several orders of magnitudes of slowdown in both training and predictions. Because of

this, we won't discuss HE any further. SMPC also has an overhead in terms of networking time when the shares and the results are passed between parties, but it is significantly faster than HE. These techniques, along with FL, are useful for situations which data can't be gathered in one place. However, they do not prevent models from memorizing sensitive data—DP is the best solution for that.

Encrypted ML is provided for TensorFlow by TF Encrypted (TFE), primarily developed by Cape Privacy. TFE can also provide the secure aggregation required for FL.

Encrypted Model Training

The first situation in which you might want to use encrypted machine learning is training models on encrypted data. This is useful when the raw data needs to be kept private from the data scientist training the model or when two or more parties own the raw data and want to train a model using all parties' data, but don't want to share the raw data. As shown in Figure 14-4, only the data owner or owners are trusted in this scenario.

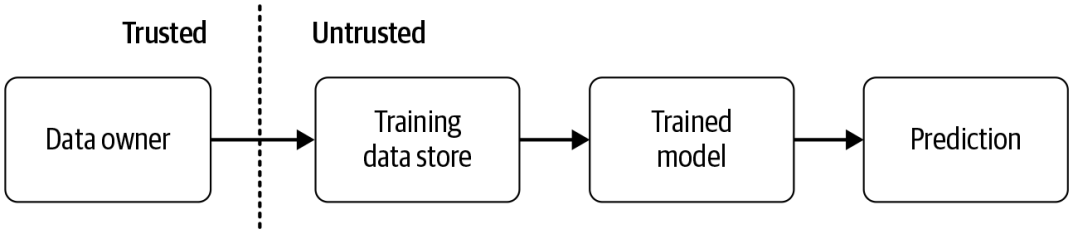

Figure 14-4. Trusted parties with encrypted model training

TFE can be used to train an encrypted model for this use case. It's installed using `pip` as usual:

```
$ pip install tf_encrypted
```

The first step in building a TFE model is to define a class that yields training data in batches. This class is implemented locally by the data owner(s). It is converted to encrypted data using a decorator:

```
@tfe.local_computation
```

Writing model training code in TFE is almost identical to regular Keras models—simply replace `tf` with `tfe`:

```
import tf_encrypted as tfe

model = tfe.keras.Sequential()
model.add(tfe.keras.layers.Dense(1, batch_input_shape=[batch_size, num_features]))
model.add(tfe.keras.layers.Activation('sigmoid'))
```

The only difference is that the argument `batch_input_shape` must be supplied to the `Dense` first layer.

Working examples of this are given in the TFE documentation. At the time of writing, not all functionality of regular Keras was included in TFE, so we can't show our example project in this format.

Converting a Trained Model to Serve Encrypted Predictions

The second scenario where TFE is useful is when you'd like to serve encrypted models that have been trained on plain-text data. In this case, as shown in Figure 14-5, you have full access to the unencrypted training data, but you want the users of your application to be able to receive private predictions. This provides privacy to the users, who upload encrypted data and receive an encrypted prediction.

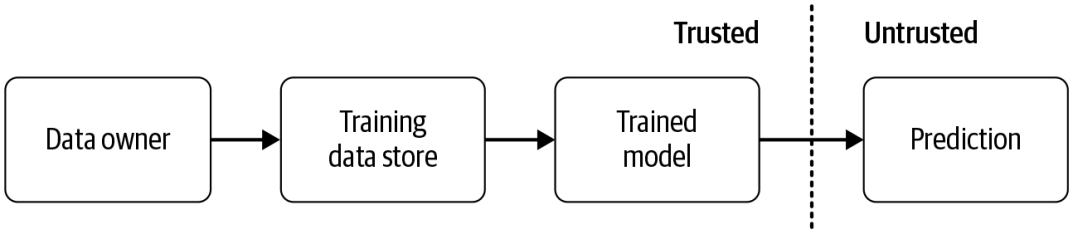

Figure 14-5. Trusted parties when encrypting a trained model

This method may be the best fit with today's machine learning pipelines, as models can be trained as normal and converted to an encrypted version. It can also be used for models that have been trained using DP. The main difference from unencrypted models is that multiple servers are required: each one hosts a share of the original model. If anyone views a

share of the model on one server or one share of the data that is sent to any one server, it reveals nothing about the model or the data.

Keras models can be converted to TFE models via:

```
tfe_model = tfe.keras.models.clone_model(model)
```

In this scenario, the following steps need to be carried out:

- Load and preprocess the data locally on the client.
- Encrypt the data on the client.
- Send the encrypted data to the servers.
- Make a prediction on the encrypted data.
- Send the encrypted prediction to the client.
- Decrypt the prediction on the client and show the result to the user.

TFE provides a series of notebooks showing how to serve private predictions.

Other Methods for Data Privacy

There are many other techniques for increasing privacy for the people who have their data included in machine learning models. Simply scrubbing text data for names, addresses, phone numbers, and so on, can be surprisingly easy using regular expressions and named-entity recognition models.

K-ANONYMITY

K-anonymity, often simply known as *anonymization*, is not a good candidate for increasing privacy in machine learning pipelines. *K*-anonymity requires that each individual in a dataset is indistinguishable from $k - 1$ others with respect to their quasi-identifiers (data that can indirectly identify individuals, such as gender, race, and zip code). This is achieved by aggregating or removing data until the dataset satisfies this requirement. This removal of data generally causes a large decrease in the accuracy of machine learning models.[10]

Summary

When you're working with personal or sensitive data, choose the data privacy solution that best fits your needs regarding who is trusted, what level of model performance is required, and what consent you have obtained from users.

All of the techniques described in this chapter are extremely new, and their production use is not yet widespread. Don't assume that using one of the frameworks described in this chapter ensures complete privacy for your users. There is always a substantial additional engineering effort involved in adding privacy to a machine learning pipeline. The field of privacy-preserving machine learning is evolving rapidly, and new research is being undertaken right now. We encourage you to look for improvements in this field and support open source projects around data privacy, such as PySyft and TFE.

The goals of data privacy and machine learning are often well aligned, in that we want to learn about a whole population and make predictions that are equally good for everyone, rather than learning only about one individual. Adding privacy can stop a model from overfitting to one person's data. We expect that, in the future, privacy will be designed into machine learning pipelines from the start whenever models are trained on personal data.

1 Cynthia Dwork, "Differential Privacy," in *Encyclopedia of Cryptography and Security*, ed. Henk C. A. van Tilborg and Sushil Jajodia (Boston: Springer, 2006).

2 Cynthia Dwork and Aaron Roth, "The Algorithmic Foundations of Differential Privacy," *Foundations and Trends* in *Theoretical Computer Science* 9, no.3–4: 211–407, (2014), https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf.

3 More details on the math behind this can be found in Dwork and Roth, "The Algorithmic Foundations of Differential Privacy."

4 Further details may be found in Justin Hsu et al., "Differential Privacy: An Economic Method for Choosing Epsilon" (Paper presentation, 2014 IEEE Computer Security Foundations Symposium, Vienna, Austria, February 17, 2014), *https://arxiv.org/pdf/1402.3329.pdf*.

[5] Nicholas Carlini et al., “The Secret Sharer,” July 2019. *https://arxiv.org/pdf/1802.08232.pdf*.

[6] Nicolas Papernot et al., “Semi-Supervised Knowledge Transfer for Deep Learning from Private Training Data,” October 2016, *https://arxiv.org/abs/1610.05755*.

[7] Robin C. Geyer et al., “Differentially Private Federated Learning: A Client Level Perspective,” December 2017, *https://arxiv.org/abs/1712.07557*.

[8] This is covered in more detail in the paper by H. Brendan McMahan et al.,“Communication-Efficient Learning of Deep Networks from Decentralized Data,” Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, *PMLR* 54 (2017): 1273–82, *https://arxiv.org/pdf/1602.05629.pdf*.

[9] For more details on system design for FL, refer to the paper by Keith Bonawitz et al.,“Towards Federated Learning at Scale: System Design” (Presentation, Proceedings of the 2nd SysML Conference, Palo Alto, CA, 2019), *https://arxiv.org/pdf/1902.01046.pdf*.

[10] In addition, individuals in “anonymized” datasets can be reidentified using outside information; see Luc Rocher et al., “Estimating the Success of Re-identifications in Incomplete Datasets Using Generative Models,” *Nature Communications* 10, Article no. 3069 (2019), *https://www.nature.com/articles/s41467-019-10933-3*.