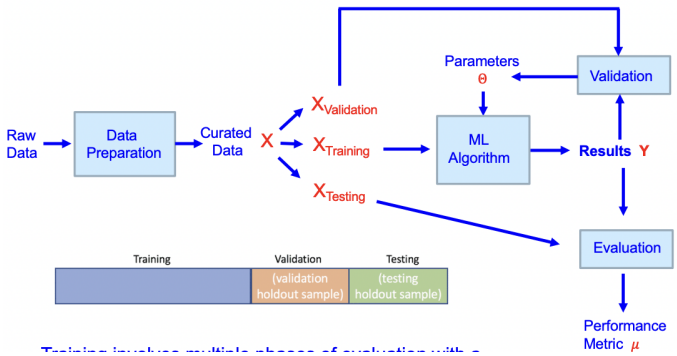
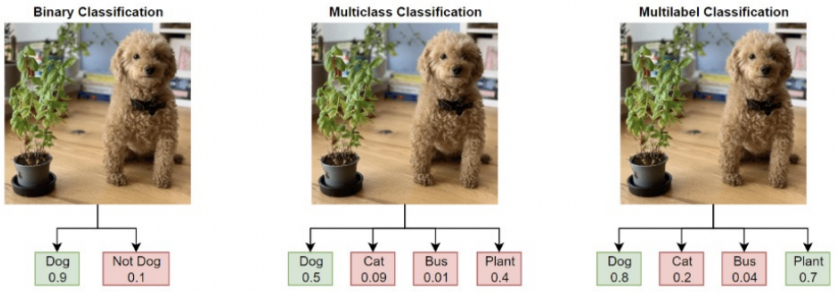
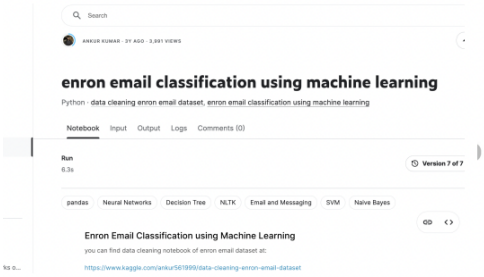
CAS CS 505
Lec 10

Deep Learning: Evaluating Classifiers; Generalization

1. Recall: Supervise Machine Learning Workflow
   a. 

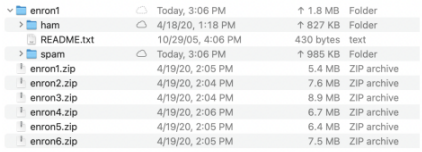
   b. Data – divide into training (most part), validation, testing
   c. Training involves multiple phases of evaluation with a validation set to find optimal values for the hyperparameters
   d. Hyperparameters (what optimizer, learning rate, geometry, batch size, etc.)
   e. Parameters are the weights inside the model (thousands or millions of them)
2. Multiclass and Multilabel Classification
   a. In Multiclass Classification, we have more than 2 labels, and our task is to assign a single label to each sample:
   b. In Multilabel Classification, we have more than 2 labels, and our task is to assign any appropriate labels (not just one):
   c. 

3. Binary Classification
    a. Spam or not spam?




| Dataset | N°of Legitimate | N°of Spam | Total |
|---|---|---|---|
| Enron 1 | 3,672 | 1,500 | 5,172 |
| Enron 2 | 4,361 | 1,496 | 5,857 |
| Enron 3 | 4,012 | 1,500 | 5,512 |
| Enron 4 | 1,500 | 4,500 | 6,000 |
| Enron 5 | 1,500 | 3,675 | 5,175 |
| Enron 6 | 1,500 | 4,500 | 6,000 |
| **Total** | **16,545** | **17,171** | **33,716** |

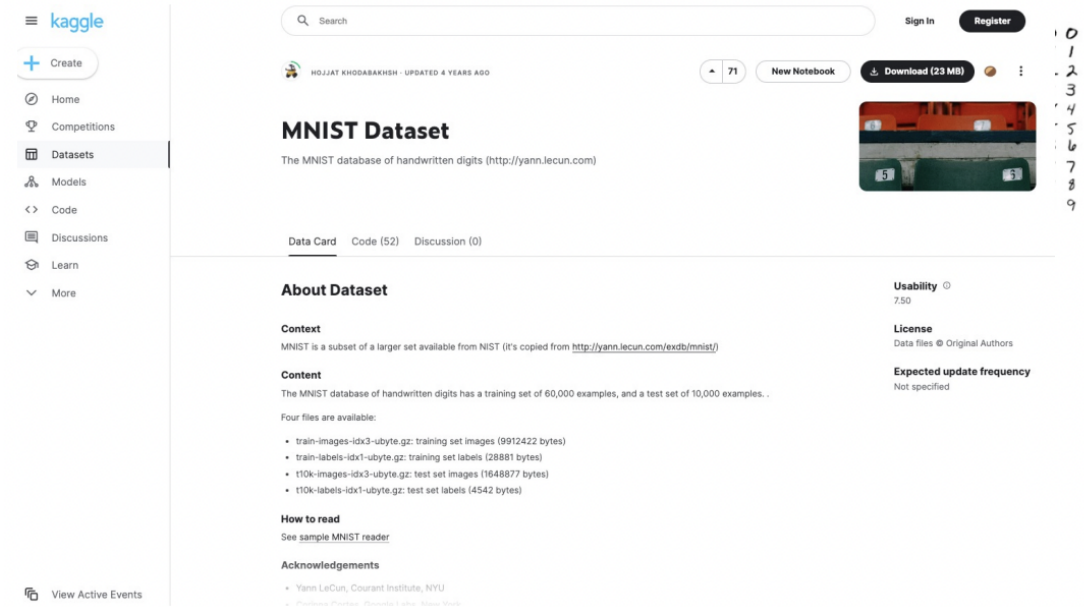
**Labels: 1 = Spam 0 = Not Spam**



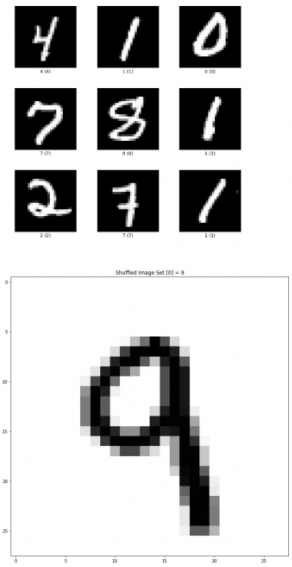

Notice that this data set is fairly well balanced:

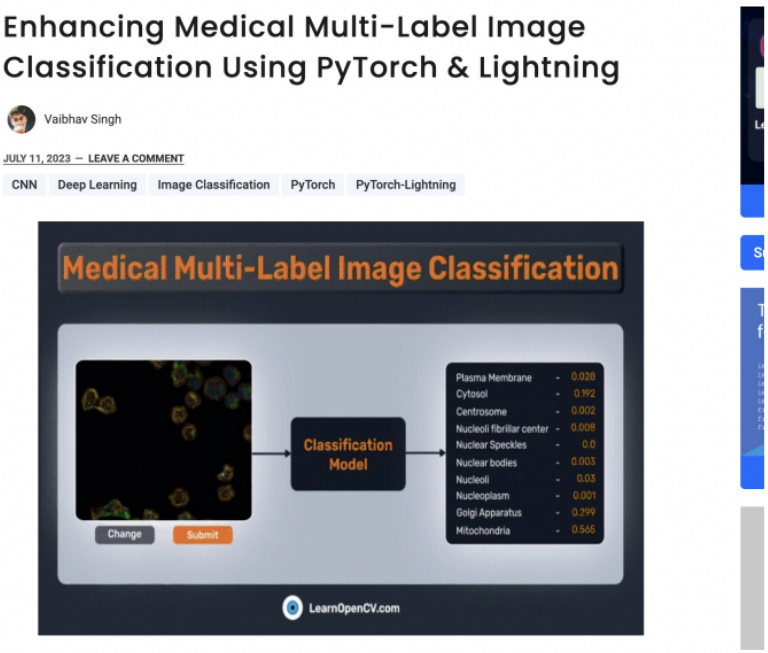
51% Spam vs. 49% Not Spam

    b.
    c. Network will have input (frequency, some kind of representation) and will output whether the email is spam or not (two neurons) – use crossentrophy
    d. Data being balanced – there can be issues with this
4. Multiclass Classification
    a. The well-known MNIST dataset of handwritten digits is a classic multiclass problem




    b.
    c. You get 60,000 (60,000 28 * 28 matrixes)
    d. Labels will be 0 - 9
    e.

5. Multiclass Classification
    a. The MNIST digit database consists of 70,000 28x28 BW pixel images, stored as 28*28=784 floating-point numbers in the range [0..1]; labels are integers 0..9:
    b. 

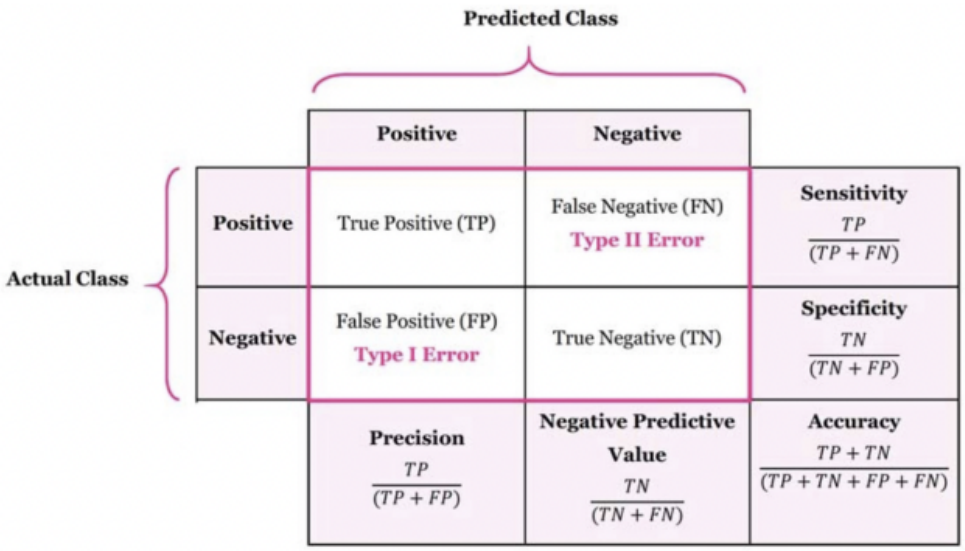

    c. This data has been flattened (put it in a linear sequence)
6. Multilabel Classification
    a. Multilabel classification is a very important problem in medical diagnosis and object recognition in images:
    b. 

    c. Image of the body – identify multiple problems
    d. With NLP, it might be adding keywords automatically to an article (to help for search in the future – identify the themes, headings)

7. Evaluation of Classifiers: “It’s complicated!”
    a. How the model is performing, how accurate it is, what does accuracy mean
    b. Precise evaluation of classifier performance is complex, even for the simplest case of binary classification:
    c. We have four different outcomes for predictions:
    d.

| True Label | Prediction | Type of Outcome |
|---|---|---|
| Dog | Dog | True Positive (TP) |
| Not Dog | Not Dog | True Negative (TN) |
| Dog | Not Dog | False Negative (FN) [Type II error] |
| Not Dog | Dog | False Positive (FP) [Type I error] |

    e. Confusion matrix (chart of the four cases):




        i. Count how many of each of the categories
    f. Gets more complicated when there are more than two classes
    g. Intuitively, these make good sense:
        i. Accuracy is the percentage of all predictions which were correct; (true positive and true negative) → how many did you get right?
        ii. Precision is the percentage of positive predictions which were correct; (of the ones you said positive, how many are actually positive)
        iii. Sensitivity (or Recall or "true positive rate") is the percentage of the actual positives identified correctly
        iv. Specificity (or "true negative rate") is the percentage of the actual negatives identified correctly (of the ones you have negative, how many are actually negative)
8. Evaluation of Binary Classifiers (still complicated)
    a. The idea of accuracy is fairly straight-forward: how many did it get correct? This is the most common in NLP.

b. In situations where the cost of errors are significant (not just misidentification), such as medical, control system for airplanes, missiles, nuclear system, there are serious consequences (mistakes can have costs that are not trivial)
c. However, the others are useful in many circumstances where the cost of errors may be unacceptable:
    i. If positive = the patient has cancer, then a FN is disasterous, since you have missed diagnosing a deadly disease, so we want to increase sensitivity; (sensitivity = recall)
    ii. If positive = the patient does NOT have cancer, then a FP is disasterous, since again you have missed diagnosing a deadly disease, so we want to increase precision;
    iii. If you are querying a database of documents, and positive = document retrieved, and you want to retrieve all possible documents, then you need to increase recall (hence the name)
d. Precision and recall(querying for something but the result is not the perfect result but is related, sensitivity) often in conflict: increasing one will decrease the other; therefore a composite measure is often used, the "harmonic mean" of prediction and recall, which attempts to equalize the error between these two

$$F_1 \text{ score } = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

e. But even accuracy is not that simple! (there is lots of reasons why – this is one of them)
f. Consider two classifiers with accuracy metrics as follows:
    i. Accuracy of 4-way classification of blobs: 90%
    ii. Accuracy of 10-way classification of digits: 85%
    iii. Alternatively: your baseline model is your model before training, assuming parameters are initialized randomly
g. Which is the more accurate classifier?
    i. Answer 1:
        1. 90% > 85% A is more accurate
        2. But wait! A baseline classifier which chooses randomly gets
        3. 25% accuracy for A and 10% accuracy for B !!
        4. [ Assuming equal distribution of classes – else it is the percentage of biggest class. ]
        5. Punchline:
            a. Always compare your model with a suitable baseline model, for example a random model which always chooses the largest class.

ii. Answer 2 (most common in ML):
    1. Measure improvement above baseline:
        a. Instead of comparing them by numbers, compare it to the baseline (choosing the answer in random)
        b. A improves on random (25%) by +65%;
        c. B improves on random (10%) by +75%, so B is a better model.

iii. Answer 3:
    1. Measure improvement normalized to inaccuracy of baseline model: (statistics researched)

Cohen's Kappa: $$\kappa = \frac{\text{accuracy} - \text{baseline accuracy}}{1.0 - \text{baseline accuracy}}$$

B still wins: $$\kappa(A) = \frac{0.9 - 0.25}{1.0 - 0.1} = 0.72 \qquad \kappa(B) = \frac{0.85 - 0.1}{1.0 - 0.1} = 0.83$$

h. Statisticians have been arguing about the right way to do this for at least a century, here is a recent paper which compares the principal methods:

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 20, 2021, accepted May 21, 2021, date of publication May 26, 2021, date of current version June 3, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3084050*

# The Matthews Correlation Coefficient (MCC) is More Informative Than Cohen's Kappa and Brier Score in Binary Classification Assessment

**DAVIDE CHICCO[1], MATTHIJS J. WARRENS[2], AND GIUSEPPE JURMAN[3]**
[1]Institute of Health Policy, Management and Evaluation, University of Toronto, Toronto, ON, Canada
[2]Groningen Institute for Educational Research, University of Groningen, Groningen, The Netherlands
[3]Data Science for Health Unit, Fondazione Bruno Kessler, Trento, Italy
Corresponding author: Davide Chicco (davidechicco@davidechicco.it)

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP) \cdot (TP+FN) \cdot (TN+FP) \cdot (TN+FN)}}$$
$$(\text{worst value} = -1; \text{best value} = +1)$$

i. ML people seem to like Answer 2 : percent improvement over baseline model which always chooses the largest class.

j. Quick Check:
    i. Suppose you have a spam/not spam binary classification task, but your data set is heavily unbalanced: 30% is spam, and 70% is not spam.
    ii. You train two different models, A gets 88% accuracy and B gets 92%.
    iii. How much better is B than A?
        1. A baseline model would simply say that all samples are not spam, and would have 70% accuracy.

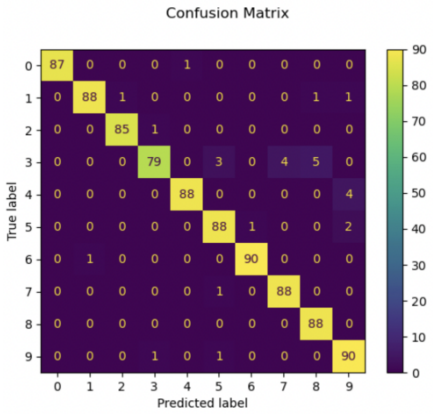
2. Standard answer: B is 22% above the baseline, and A is 18% above.
3. This doesn't sound that great, compared with 92% for your best model!
4. You can also see why balanced data sets are preferred!

9. Evaluation of Multiclass Classifiers (yup, complicated)
    a. Large Confusion Matrices
    b. If you are doing multiclass classification, you can extend the confusion matrix to have as many rows and columns as the number of classes.
    c. Here is a simple example of a large Confusion Matrix showing how two strings match up (as if each character were a label!):

```
>>> reference = 'This is the reference data.  Testing 123.  aoaeoeoe'
>>> test =      'Thos iz_the rifirenci data.  Testeng 123.  aoaeoeoe'
>>> print(ConfusionMatrix(reference, test))
  |   . 1 2 3 T _ a c d e f g h i n o r s t z |
--+-----------------------------------------+
  |<8>. . . . 1 . . . . . . . . . . . . . . |
. | .<2>. . . . . . . . . . . . . . . . . . |
1 | . .<1>. . . . . . . . . . . . . . . . . |
2 | . . .<1>. . . . . . . . . . . . . . . . |
3 | . . . .<1>. . . . . . . . . . . . . . . |
T | . . . . .<2>. . . . . . . . . . . . . . |
_ | . . . . . .<.>. . . . . . . . . . . . . |
a | . . . . . .<4>. . . . . . . . . . . . . |
c | . . . . . .<1>. . . . . . . . . . . . . |
d | . . . . . .<1>. . . . . . . . . . . . . |
e | . . . . . .<6>. . . 3 . . . . . . . . . |
f | . . . . . .<1>. . . . . . . . . . . . . |
g | . . . . . .<1>. . . . . . . . . . . . . |
h | . . . . . .<2>. . . . . . . . . . . . . |
i | . . . . . 1 . . .<1>. 1 . . . . . . . . |
n | . . . . . .<2>. . . . . . . . . . . . . |
o | . . . . . .<3>. . . . . . . . . . . . . |
r | . . . . . .<2>. . . . . . . . . . . . . |
s | . . . . . .<2>. 1 . . . . . . . . . . . |
t | . . . . . .<3>. . . . . . . . . . . . . |
z | . . . . . .<.>. . . . . . . . . . . . . |
--+-----------------------------------------+
(row = reference; col = test)
```

    d.
        i. The y-axis represents true labels, x-axis represents predicted labels
        ii. The same measures are used as in binary classification:
        iii. Accuracy = % correct
        iv. Sensitivity and specificity are the average for each row.
        v. Recall and NPV are the average for each column.
    e. These displays can help you to understand your data and how your model is performing with respect to individual samples.
    f. Here is a confusion matrix for the MNIST digit-recognition task:

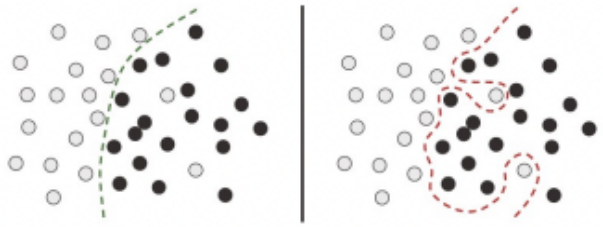
g.



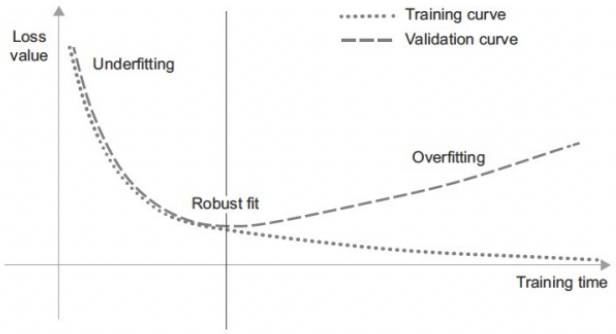

```
Classification report rebuilt from confusion matrix:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        88
           1       0.99      0.97      0.98        91
           2       0.99      0.99      0.99        86
           3       0.98      0.87      0.92        91
           4       0.99      0.96      0.97        92
           5       0.95      0.97      0.96        91
           6       0.99      0.99      0.99        91
           7       0.96      0.99      0.97        89
           8       0.94      1.00      0.97        88
           9       0.93      0.98      0.95        92

    accuracy                           0.97       899
   macro avg       0.97      0.97      0.97       899
weighted avg       0.97      0.97      0.97       899
```

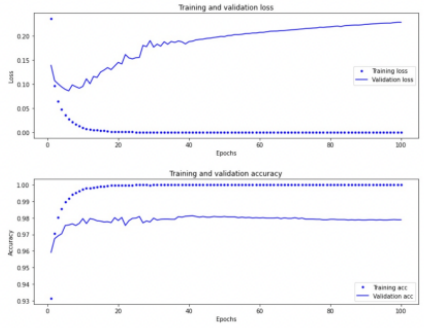
h. The errors are small in most cases, but there is a specific case where the error is pretty large (3 gives 5, 7, 8) → can dig into your data to see the problem

10. Generalization and Overfitting
    a. Goal in deep learning → more than high accuracy
    b. Generalization – the ability of a NN to learn the patterns in a data set so as to perform well on data it has never seen – is the most important goal in developing deep learning models.
    c. The problem is overfitting – the NN is starting to "memorize" the training set without learning the most important patterns which characterize the essential information present in the data.
    d. Overfitting can be seen when the training loss goes down, but the validation loss goes up. In general, you will see the validation accuracy peak at some epoch and then goes down (generally not as noticably as the rise in the validation loss):



e.
    i. The two of points are wrong in the left but it is good in general → that is what we want
    ii. If we memorize, then we are not getting it 100% correct (we are not learning the data) → we are not learning the actual pattern (not the general information)




Best Validation Accuracy: 0.9813 at epoch 40

Figure 5.1 Canonical overfitting behavior

f.

i. At some point, we get a robust fit during training (training curve will go down and the validation curve will go up) → validation is the data we have not seen during training stage
ii. You are memorizing the data in trainset → looks bad but not that bad (discuss later)
iii. Loss on the validation set will go down
iv. For accuracy, the validation will converge to some value (it is the final accuracy indication)

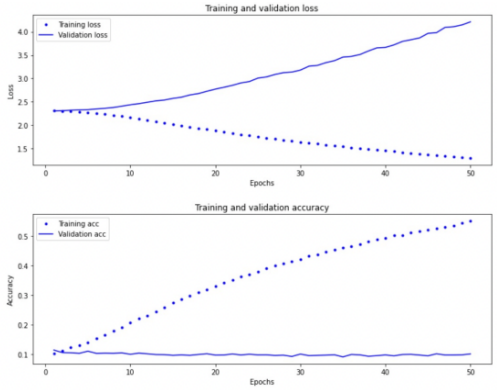
g. This problem will happen (not necessarily bad)

11. Generalization: Overfitting

a. The problem is that a NN can learn ANY data set you give it, essentially by memorizing the exact training set. Here is a dramatic example: we randomly permute the labels, so that there is no correspondence between data and labels. The model continues to "learn" the training set, but the validation accuracy remains around the baseline of 10%.

```
shuffled_indices = np.random.permutation(num_data_instances)
shuffled_indices2 = np.random.permutation(num_data_instances)

images = images[shuffled_indices]
labels = labels[shuffled_indices2]
```

```
[37]: model.evaluate(testing_images, randint(10,size=len(testing_labels)), batch_size = 128)
110/110 [==============================] - 0s 2ms/step - loss: 4.1812 - accuracy: 0.1021
:[37]: [4.181239128112793, 0.10207142680883408]
```



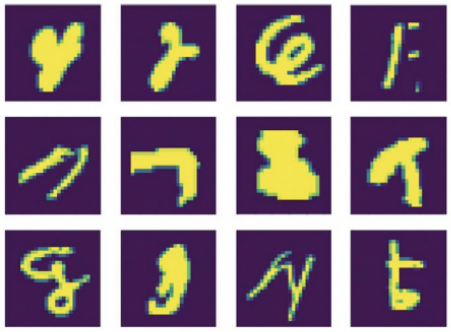

Best Validation Accuracy: 0.1139 at epoch 0

b.

c. Overfitting is often due to data which is

i. Noisy (non-data, ambigious, or outliers) → misspelling, abbreviation in NLP
ii. Mislabeled → emojis
iii. Or has rare features or spurious correlations.

Noisy Data in MNIST:



Mislabeled data in MNIST:


Figure 5.3 Mislabeled MNIST training samples

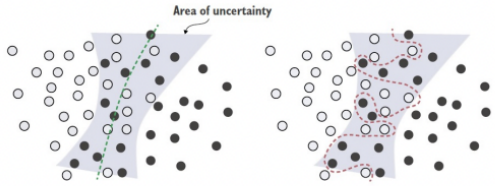



Figure 5.5 Robust fit vs. overfitting giving an ambiguous area of the feature space

iv.

d. Rare features
    i. If your data contains "one-off" features (e.g., a "Getty Images" logo in one image, or a unique or misspelled word in an email), the NN will learn to associate that feature with its label – it is overfitting!
e. Spurious Correlations
    i. his is actually worse—and more common—than rare features. A word may occur 100s of time in movie reviews, but by a statistical fluke, it occurs in 58% of the positive reviews, and 42% of the negative reviews. The NN will give this word undue weight in learning the data set, and it won't generalize well.

12. Generalization: A Deep Dive into the Matrix…
    a. The Manifold Hypothesis
    b. A manifold in an N dimensional space is a set of points which is isomophic to a lowerdimensional space that is Euclidean, i.e., is continuous and has a notion of "distance."
    c. Ex 1: A curved line is literally in 2 D, but can be mapped 1-to-1 (isomophic) to a 1 D line:

---

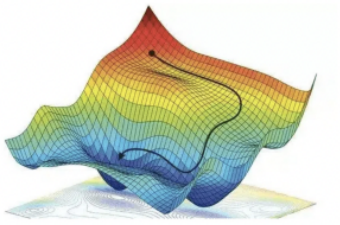
d. Ex 2: A crumpled piece of paper is 3 D, but is isomophic to a 2D (flat) piece of paper:

Figure 5.9 Uncrumpling a complicated manifold of data

e. Ex 3: Moblus strip

f. The Manifold Hypothesis is "that many high-dimensional data sets that occur in the real world actually lie along low-dimensional latent manifolds inside that high-dimensional space. As a consequence of the manifold hypothesis, many data sets that appear to initially require many variables to describe, can actually be described by a comparatively small number of variables, likened to the local coordinate system of the underlying manifold. It is suggested that this principle underpins the effectiveness of machine learning algorithms in describing high-dimensional data sets by considering a few common features." -Wikipedia

g. Your model is searching in a high-dimensional space (= number of parameters attached as weights to neurons) for a representation of the data (lower dimensional manifold). The spaces are continuous and have a notion of distance, which are intrinsic to the gradient descent algorithm:



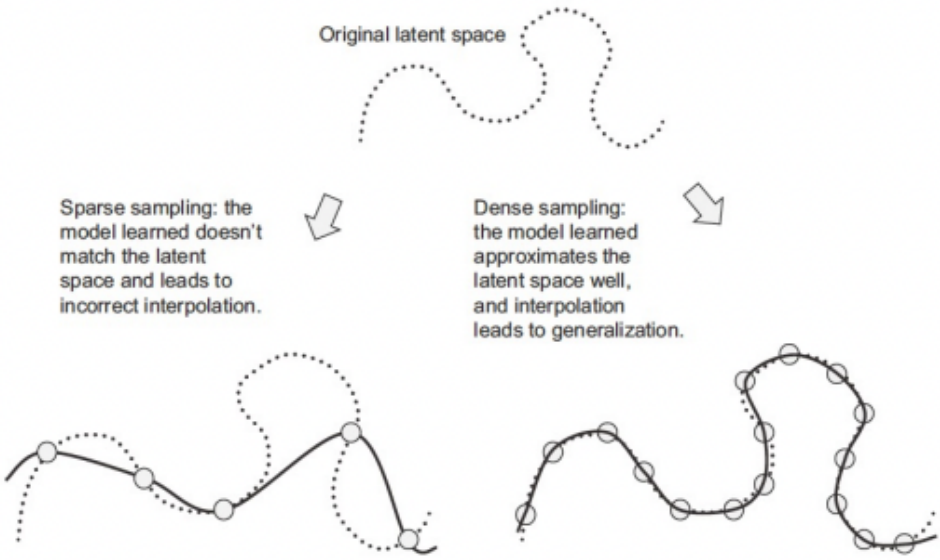



**Figure 5.11 A dense sampling of the input space is necessary in order to learn a model capable of accurate generalization.**

h.

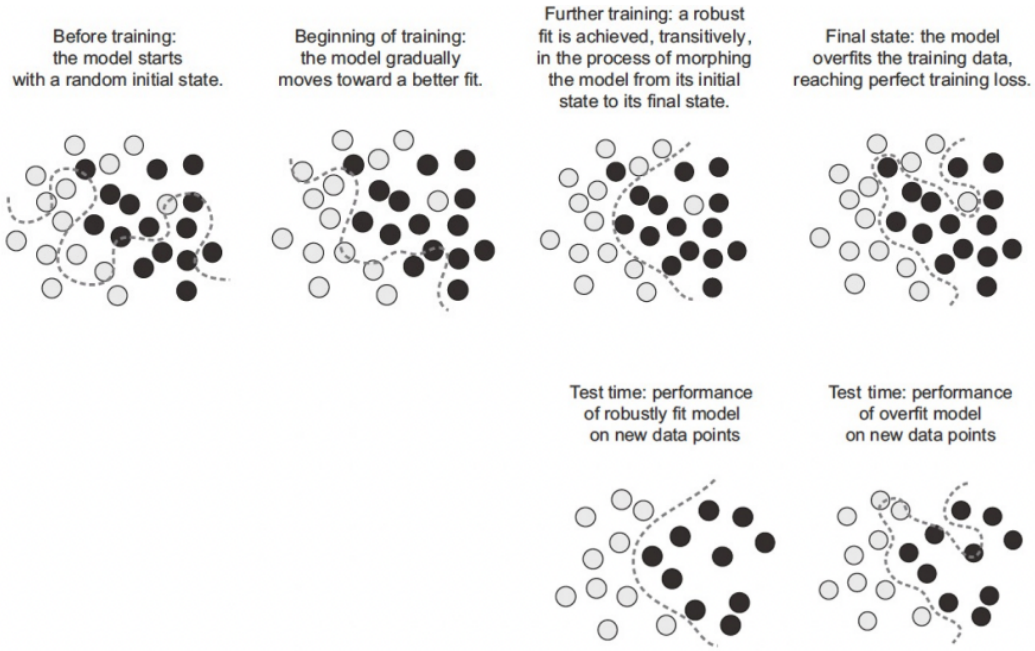



**Figure 5.10 Going from a random model to an overfit model, and achieving a robust fit as an intermediate state**

i.

j. Point is finding between knowing too much of the dataset and knowing generally about the data set

13. Generalization: Underfitting and Overfitting
    a. Overfitting is not a sign that something is wrong with your model (you want the model that overfits – it is the evidence that the model is complex enough to learn the data), in fact, it shows that your model has sufficient power to represent the

patterns that characterize the true "meaning" of the data. You just have to find ways to control this awesome power.

b. Chollet, p.138: "The first big milestone of a machine learning project: getting a model that has some generalization power (it can beat a trivial baseline) and that is able to overfit." p.141: "Remember that it should always be possible to overfit."

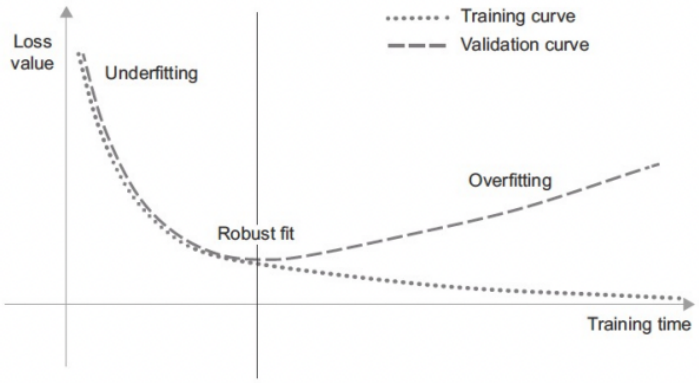
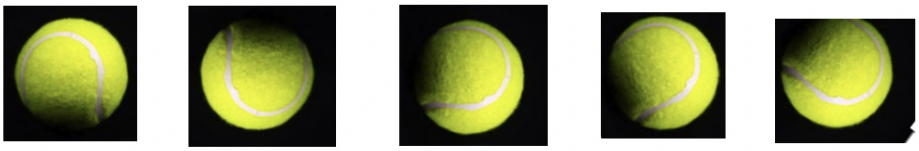


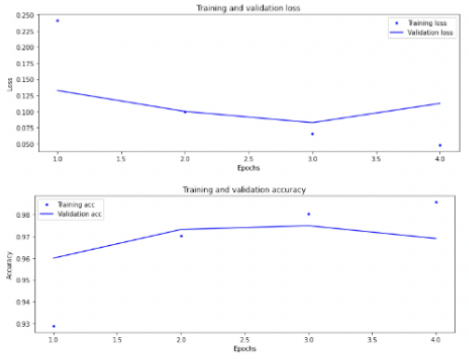

Figure 5.1 Canonical overfitting behavior

c.

14. Improving Generalization
    a. Improving generalization can be accomplished by various techniques.
    b. Getting more data, improving your data: more data is almost always better; make sure there are minimal labeling errors, reconsider your data normalization.
    c. If you can not get more data, consider data augmentation: manipulating your existing data in ways that produce different samples with the same essential information. (not useful for NLP → manipulate the data so that it has the same content → unclear for text and is complicated)
    d. NLP way → use synonyms (not exactly the same), go to embeddings and find the closest word and replace it
    e. 
    f. Unfortunately, data augmentation is a little tricky in NLP: how do you create new documents or text that has the same "essential information"? Typical approaches: replace words by synonyms, translate to another language, then back, etc. (not very satisfying).
    g. Reconsider your choice of architecture: Add more layers, or fewer, or of different widths. Consider starting with wider layers, and getting narrower as you go deeper. Consider different kinds of layers better suited to your data (this works better with images than in NLP). Google around to see what others have done successfully with similar data.

h. Tuning hyperparameters: Play with the hyperparameters, including type of optimizer, the learning rate (if getting bizarre behavior → learning rate is too huge, so lower the learning rate), and the batch size.
i. Better feature engineering: Use domain knowledge about the data, and experience with the model you are using to better represent the data. Tools can help with feature selection (find out which features are making the most difference).
    i. Example: What kind of word vector? TF? TF-IDF? DIY embeddings? Glove embeddings? etc. ....
j. In some simple cases, we can observe which features were most important (in a classification problem)
k. Early Stopping:
    i. Stop training when a robust fit is achieved. This can often be done automatically by setting a parameter in your model. An elegant method is to save the best model after every K epoches, then refer back after you've gone too far.... Here is a naive example of early stopping, which does not do so well: (or save the best model with respect to the validation set)

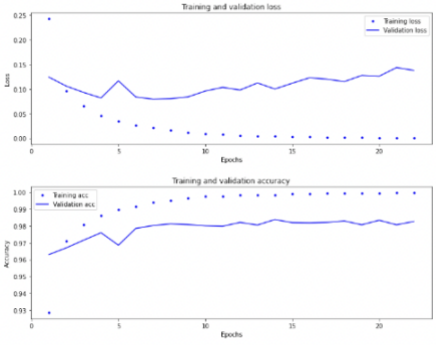
```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping()
custom_early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=8,
    min_delta=0.001,
    mode='max'
)


model = keras.Sequential([
    layers.Dense(512,activation="relu"),
    layers.Dense(10,activation='softmax')
])
model.compile( optimizer = 'rmsprop',
               loss = 'sparse_categorical_crossentropy',
               metrics = ['accuracy'] )

history = model.fit(training_images,
                    training_labels,
                    verbose= 0,
                    epochs=50,
                    batch_size=64,
                    callbacks=[early_stopping],
                    validation_data=(validation_images,validation_labels))

display_results(history)
model.evaluate(testing_images, testing_labels,batch_size = 128)
```




```
Best Validation Accuracy: 0.9749 at epoch  2
110/110 [==============================] - 0s 1ms/step - loss: 0.1168 - accuracy: 0.9689
[0.11679539084434509, 0.9688571691513062]
```

    ii. Tuning the early stopping callback results in better results:

```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping()
custom_early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=8,
    min_delta=0.001,
    mode='max'
)


model = keras.Sequential([
    layers.Dense(512,activation="relu"),
    layers.Dense(10,activation='softmax')
])
model.compile( optimizer = 'rmsprop',
               loss = 'sparse_categorical_crossentropy',
               metrics = ['accuracy'] )

history = model.fit(training_images,
                    training_labels,
                    verbose= 0,
                    epochs=50,
                    batch_size=64,
                    callbacks=[custom_early_stopping],
                    validation_data=(validation_images,validation_la

display_results(history)
model.evaluate(testing_images, testing_labels,batch_size = 128)
```




```
Best Validation Accuracy: 0.9839 at epoch  13
In [36]: model.evaluate(testing_images, testing_labels,batch_size = 128)
110/110 [==============================] - 0s 1ms/step - loss: 0.1475 - accuracy: 0.9807
Out[36]: [0.14754530787467957, 0.9807142615318298]
```

l. Regularization:
   i. Hampering the power of the model somehow
   ii. Various techniques which "actively impede the model's ability to fit perfectly to the training data, with the goal of making the model perform better during validation." The model is simpler, more "regular."
   iii. Reduce model size (but not too small) → is one way to achieve it

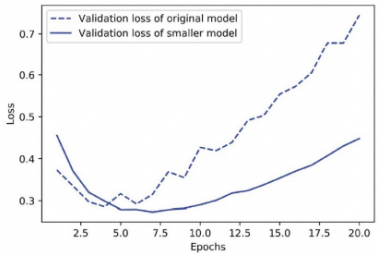



Figure 5.17 Original model vs. smaller model on IMDB review classification

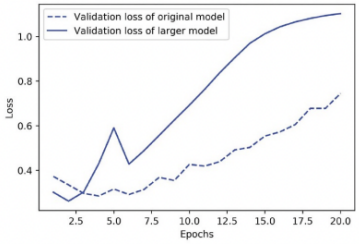



Figure 5.18 Original model vs. much larger model on IMDB review classification

   iv. Regularization: Various techniques which "actively impede the model's ability to fit perfectly to the training data, with the goal of making the model perform better during validation." The model is simpler, more "regular."
   v. Weight Regularization: Place limits on how large the weights in the model can become, so that the model is forced to be simpler (having fewer possibilities of weights). (the weights, parameters → cannot let them get too extreme) There are two flavors:

- *L1 regularization*—The cost added is proportional to the *absolute value of the weight coefficients* (the *L1 norm* of the weights).
- *L2 regularization*—The cost added is proportional to the *square of the value of the weight coefficients* (the *L2 norm* of the weights). L2 regularization is also called *weight decay* in the context of neural networks. Don't let the different name confuse you: weight decay is mathematically the same as L2 regularization.

1. The Ls are the losses
2. L1 → absolute value
3. L2 → square of the loss
4. You add L1/L2 to the cost of the model (the bigger the values go, the worse the cost goes)
5. Add this term to some alpha/weight → if it gets too complicated, it affects the loss and gets worse
6. L2 is fairly common

**Listing 5.13 Adding L2 weight regularization to the model**

```
from tensorflow.keras import regularizers
model = keras.Sequential([
    layers.Dense(16,
                 kernel_regularizer=regularizers.l2(0.002),
                 activation="relu"),
    layers.Dense(16,
                 kernel_regularizer=regularizers.l2(0.002),
                 activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

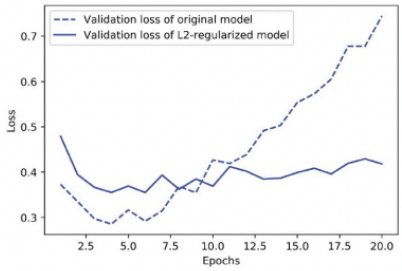



Figure 5.19 Effect of L2 weight regularization on validation loss

vi. Adding Dropout: Dropout is applied to a layer, and is very simple: with some probability p, set each parameter in a layer to 0.0: (weird idea but works well)

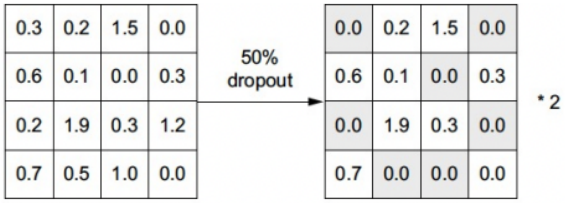



**Figure 5.20 Dropout applied to an activation matrix at training time, with rescaling happening during training. At test time the activation matrix is unchanged.**

1. You have a layer and add another layer
2. With some probability (p = 0.5), we have the probability that the next layer becomes 0 (you have p% of deleting the next layer)
3. So p = 0.5 sounds bizarre, but it works fine

vii. This is one of the weirdest great ideas in Deep Learning: it seems like it can't possibly help, but it is one of the most effective and most common ways to regularize your model.

viii. This is because it means that you cannot simply memorize little pieces of the data and store it in a network but has to learn it in an overall way

ix. Learning in the presence of forgetting is powerful (cannot simply memorize → has to learn the overall principle)

**Listing 5.15 Adding dropout to the IMDB model**

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history_dropout = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

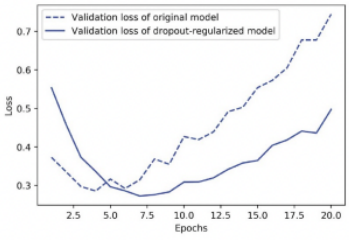



Figure 5.21 Effect of dropout on validation loss