Introduction to Machine Learning

March 6, 2023

INSTITUTO TECNOLOGICO AUTONOMO DE MEXICO (ITAM)

Economic Research Seminar / Applied Research 2[1]

Professor: Horacio Larreguy

TA: Eduardo Zago

[1]We give credits to the Deep Learning course taught at ITAM's Master in Data Science because some of the content in this tutorial is taken from the notes made by PhD. Edgar Francisco Roman-Angel

1 Introduction to Machine Learning

1.1 What is Machine Learning?

Machine learning is a field of artificial intelligence that involves teaching computers to learn from data, without being explicitly programmed. The goal of machine learning is to develop algorithms that can automatically learn patterns and relationships in data and use them to make predictions or decisions.

Mathematically, the general set-up would be the next one:

$$y = f(x; w)$$

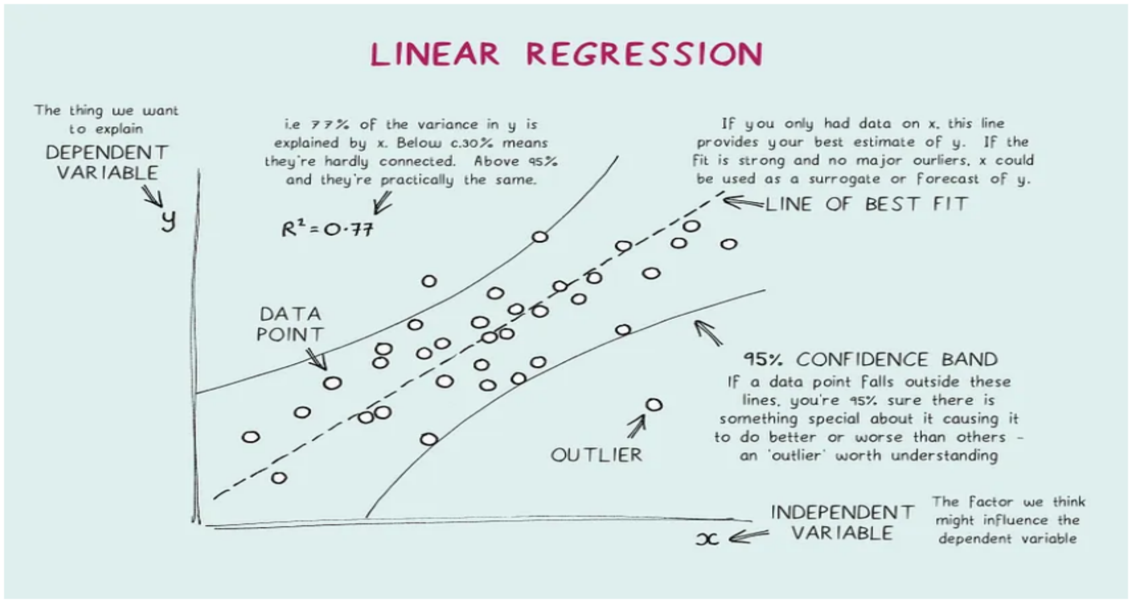
Where x is the input data, which can be vectors, text, images, etc. $f()$ is a mapping function, w is the set of parameters to be trained and y is the expected output of the mapping function. So when we say that we are **training a model**, we are learning or estimating the values of w that best map x on y. Our prediction would be $\hat{y}$.

With Machine Learning and Deep Learning we can solve different types of problems, which is very important to be able to distinguish between them:

- **Regression:** A problem where the output y lives in a continuous space. $f()$ returns a real value $y \in \mathbf{R}$[2]

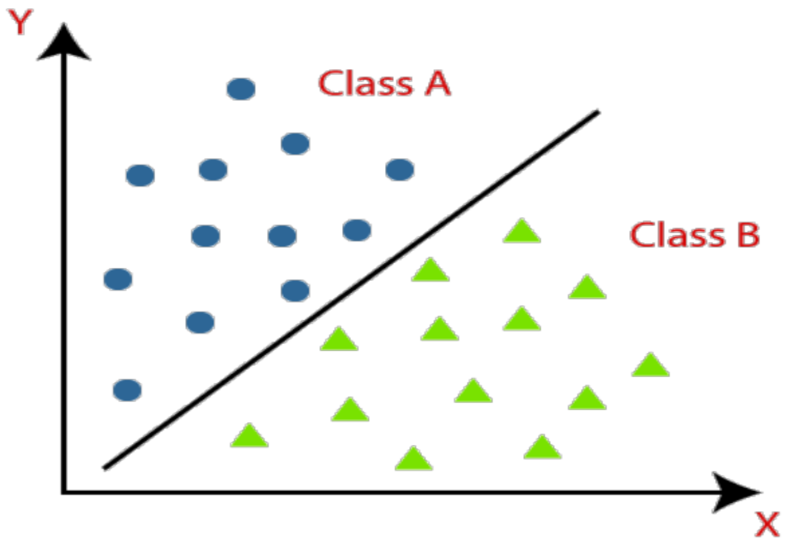
[2]Image by Dan White 1000

Figure 1: Regression


LINEAR REGRESSION
The thing we want to explain
DEPENDENT VARIABLE
y
i.e 77% of the variance in y is explained by x. Below c.30% means they're hardly connected. Above 95% and they're practically the same.
R² = 0·77
If you only had data on x, this line provides your best estimate of y. If the fit is strong and no major ourliers, x could be used as a surrogate or forecast of y.
LINE OF BEST FIT
DATA POINT
95% CONFIDENCE BAND
If a data point falls outside these lines, you're 95% sure there is something special about it causing it to do better or worse than others - an 'outlier' worth understanding
OUTLIER
INDEPENDENT VARIABLE
x
The factor we think might influence the dependent variable

- **Classification:** A problem where the output y lives in a discrete space. $f()$ returns a categorical value $y \in \{b_1, b_2, ..., b_n\}$[3]

Figure 2: Classification



[3]Image taken from ITAM's Deep Leaning Course Slides

1.2 Supervised vs. Unsupervised Learning

There are three main types of machine learning: supervised learning, unsupervised learning, and reinforcement learning.

1. **Supervised learning** involves learning from labeled data. Examples: classification, regression, etc.
2. **Unsupervised learning** involves learning from unlabeled data. Examples: Cluster K-Means, Topic Modeling, etc
3. **Reinforcement learning** involves learning from feedback in a dynamic environment (prices and punishments, hits and misses).

1.3 Training, Validation and Test Data Sets

To solve the problem of classification, say that we would like to code an algorithm that accurately distinguishes between a dog and a cat, we could code up each and every one of the categories and characteristics that identify them. This could take a big amount of time and a lots of lines of code. Therefore, instead of trying to specify what every one of the categories of interest look like directly in code, the approach that we can take is to provide the computer with many examples of each class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class. This approach is referred to as a *data-driven approach*, since it relies on first accumulating a **training dataset** of labeled observations (Stanford CNN Course).

To train a ML model, we will divide our labeled data-set into three subsets:

1. **Training set:** Initial set used to estimate the values for our parameters $\{w_i\}$, through error minimization and parameter updating.
2. **Validation set:** Once we have adjusted the values of $\{w_i\}$, we use the validation set to evaluate how well the model performs on data it has never seen. We use this

set to see how we can improve our prediction by modifying the *hyper-parameters*.

3. **Test set:** This data set is used for final evaluation of the performance of the model. This data set was never used to train the weights or modify the hyper-parameters, so it is the final test for **generalization**.

1.4 Metrics

When training a model, we will look at several metrics to know the goodness of the fit. How good our model is for prediction.

1.4.1 Accuracy vs. Loss

Accuracy and loss are two common metrics used in machine learning to evaluate the performance of a model.

Accuracy is a measure of how often the model makes correct predictions, expressed as a percentage. It is calculated by dividing the number of correct predictions by the total number of predictions. For example, if a model correctly predicts 75 out of 100 samples, its accuracy is 75%.

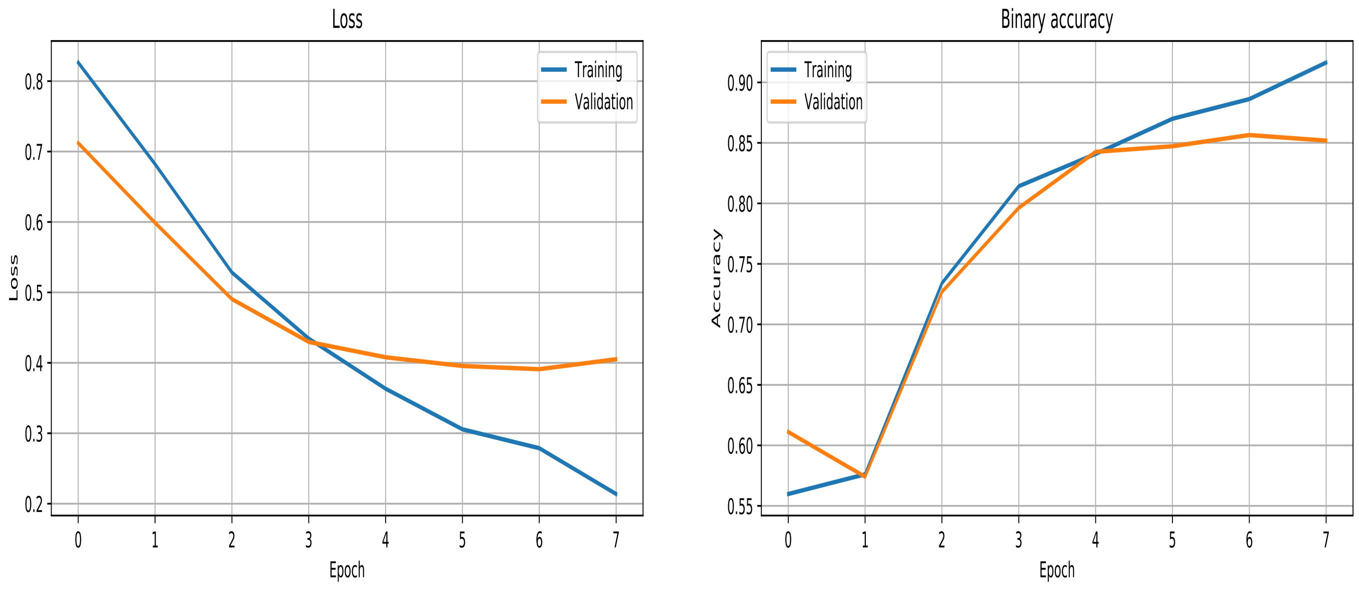
Loss, on the other hand, measures the difference between the predicted output and the actual output for a given input. It is a mathematical function that computes the error between the predicted output and the actual output. The goal of the model is to minimize the loss function, so that the predicted output is as close as possible to the actual output.

Thus, accuracy measures the overall performance of the model, while loss measures the error of the individual predictions made by the model. One of the goals of a machine learning model is to have **both high accuracy and low loss**.

1.4.2 The Learning Curve

A **learning curve** is a graph that shows the performance of a machine learning model on a task over time, as the amount of training data increases. Specifically, the learning curve plots the model's training and validation error (or accuracy) on the y-axis against the size of the training dataset on the x-axis[4]

Figure 3: Learning Curve, Loss and Accuracy



The learning curve is useful for diagnosing problems with a model, such as overfitting or underfitting, and for determining whether collecting more data would help to improve the model's performance.

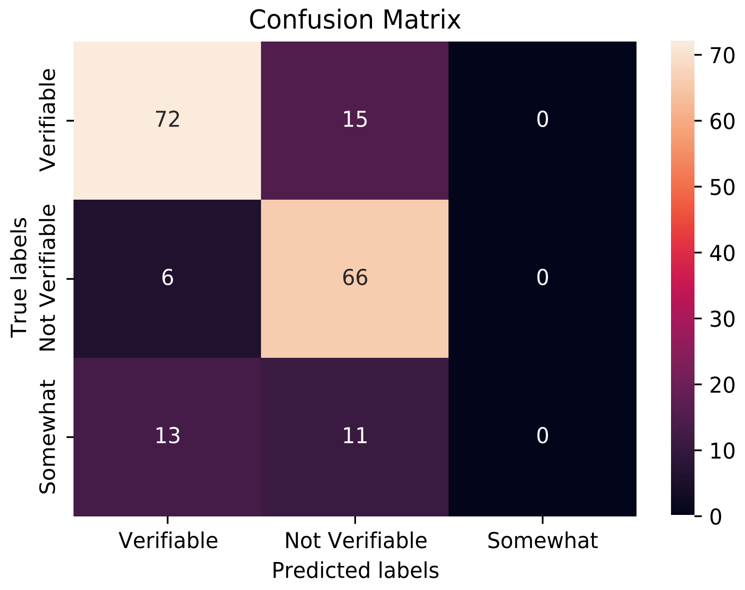
1.4.3 Confusion Matrix

A **confusion matrix** is a table used to evaluate the performance of a classification model. It shows the number of true positives, false positives, true negatives, and false

[4]An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one pass. Usually, the training data is given to the model in random batches (subsets of this data).

negatives for each class. The prediction is in the x-axis while the actual labels are plotted in the y-axis.

Figure 4: Confusion Matrix, Three Labels



The confusion matrix is used to calculate evaluation metrics such as accuracy, precision, recall, and F1 score.

1.5 Over-fitting and Under-fitting

When doing ML, our main objective will be to choose the model that **best fits our data** and provides the most understandable results. To do this we will probably have to choose between different types of model. Some are better for working with financial data (LSTMs, Random Forest), others are better for working with text data (BERT, GPT-3). Even more, once we have chosen our model, some of them have what we call *hyper-parameters*, which are parameters chosen by the programmer that can improve or worsen the results of our model. External variables of the model set through trial and error. One great example of a *hyper-parameter* chosen before training any particular

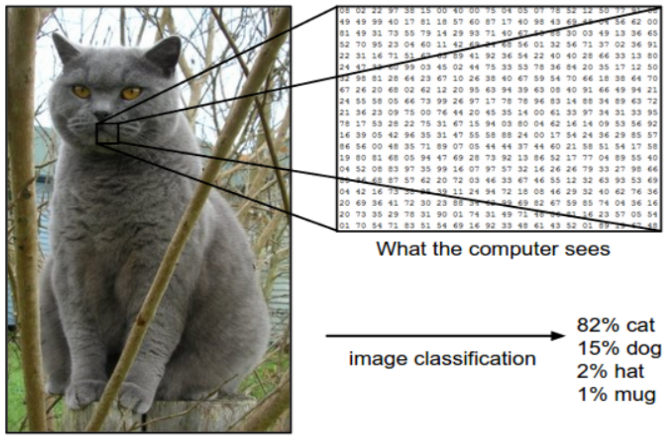
ML model is the train-test split ratio. How much of the whole data set do we want to be used for training and how much for test? The more common one and the one we will use is 80% training and 20% test.

When we choose between these models/hyper-parameters, we are choosing between models to obtain the one that best fits **the true distribution of our data**. Take a look at the following picture of a cat and a dog, and think of the real distribution of this data.

Figure 5: Cat vs. Dog

One can go as deep as in differences in their DNA to obtain a distribution that accurately differentiates between them. However, we, as humans, understand better visual characteristics, so we differentiate them by the shape of their face and body, and probably size and weight, pretty accurately. Moreover, we can assign descriptive variables such as the weight, height, color, number of legs, teeth, that would all together form a distribution that if known, would also accurately differentiate between them (probably less than us). Finally, we can also take pictures of them and represent them mathematically through pixels, as shown in Figure 2. This numbers form as well a distribution.

Figure 6: Cat Image to Pixel



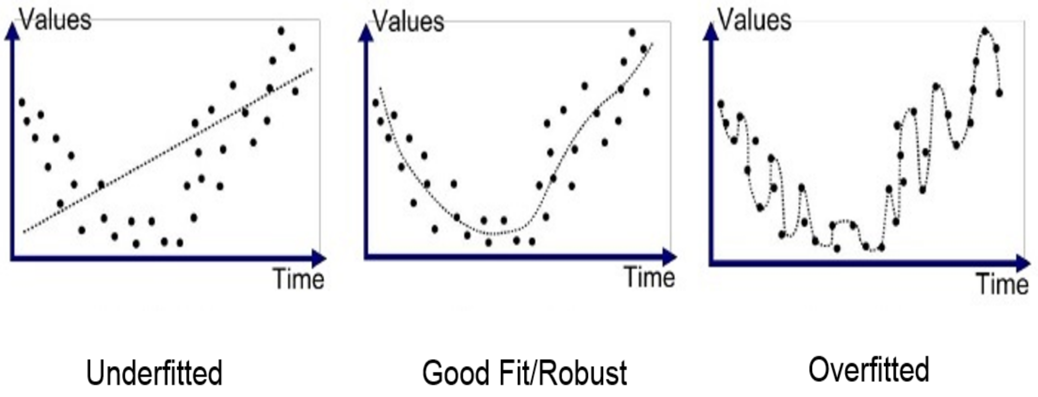
Thus, the ML and DL model's objective, will be to learn this data generating process, this distribution. When training a model, one can deal with 3 particular situations:

1. **Under-fitting**: training excluded the true data-generating process and induces bias (high training error). Usually happens when you use a simple model for a complex problem (e.g. Probit for Image Classification) or the data does not have a inherent pattern (randomly generated), then the model does not learn all the patterns in the data.

2. Training matched the true data-generating process. The model actually learnt the distribution of the data, rather than memorizing the training set or failing to extract the patterns.

3. **Over-fitting:** included the generating process but also other possible generating processes. Model fails to learn the patterns in the data and rather memorizes the data set, so it fails to *generalize* in unseen data. Variance rather than bias dominates the estimation error. Usually happens when you use a complex model for a simple problem which picks up the noise from the data (Neural Network for the Iris Data Set).

Figure 7: Under-fitted, Robust and Over-fitted

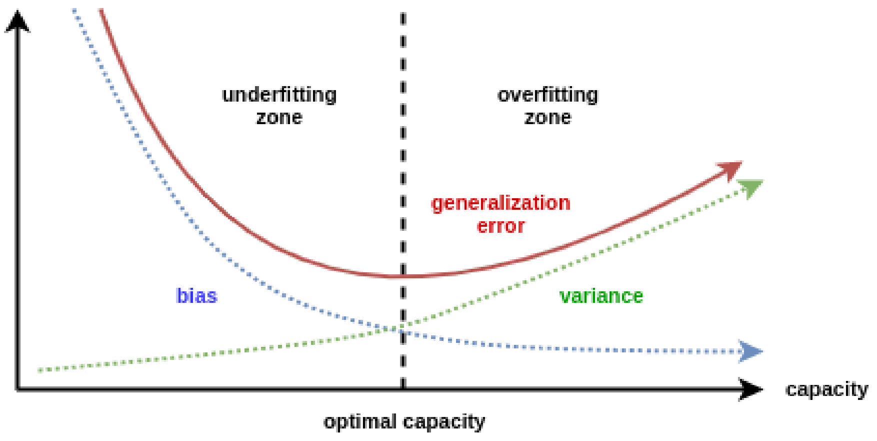


There are few easy ways to know if your model is **under-fitted** or **over-fitted** by looking at the metrics, expecially the learning curve.

- If the training accuracy and validation accuracy are low, then the model is probably under-fitted. Same goes if training error and validation error are high.[5]

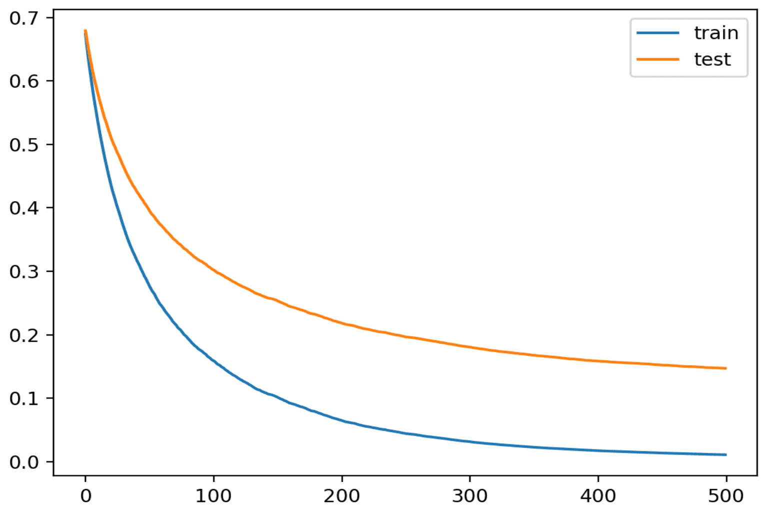
[5]One has to be cautious with this, since determining what is a high/low level of accuracy its contextual to the problem in hand . There's usually an **acceptable level of accuracy** which is usually dictated either by the literature or by the minimum accepted accuracy you have to obtain for the model to be useful. For instance, the acceptable level of accuracy for a sentiment prediction problem is lower than for a COVID detection one. For the latter, a wrong prediction (false negative) could mean dead.

Figure 8: Under-Fitted vs. Over-fitted



- If the training accuracy is high, but the validation accuracy is low or significantly lower than the training one, there is over-fitting. Same goes with low training loss and high validation loss.

Figure 9: Over-fitted Curve



2 Probit and Logit

Probit and logit are both types of generalized linear models (GLMs) that are used for binary classification. Probit is based on the cumulative distribution function of a normal distribution, while logit is based on the logistic function. Both models use a set of predictor variables to estimate the probability that an individual observation belongs to a particular class.

2.1 Understanding Probit and Logit Regression: Linear Probability Model

When thinking about the Probit and Logit prediction models, it is impossible not to think about linear regression, as seen in our Econometric's classes. Recall that the prediction $\hat{y}$ from a linear model with regressors X_1 to X_m is given by:

$$\hat{y} = \hat{\beta}_0 + \sum_{i=1}^{m} \hat{\beta}_i X_i$$

When this outcome variable is a dummy that only takes values of 1 and 0, the prediction could be taken as a probability (thus the name, Linear Probability Model). However, the prediction may deviate from the range between zero and one, being the main disadvantage of the LPM. Here is when Probit and Logit come in hand.

2.2 Logit Model

The Logit Model estimates the probability of belonging to a certain class (binary classification) by using a set of predictors (independent variables) X and the Logistic Function for 'activation'. We are passing the linear combination of the beta coefficients with the regressors through a Logistic function to guarantee that the probability is between 0 and 1, and to add a non-linearity, where everything else is linear. The Logistic function is the following:

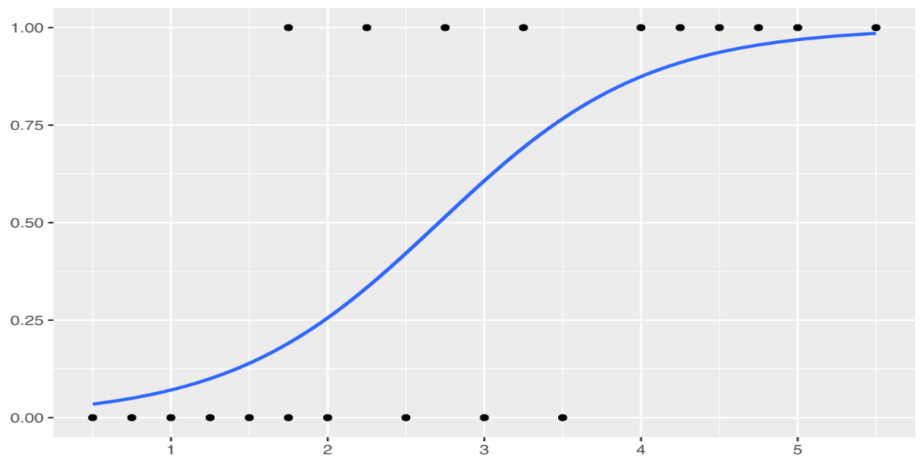
$$l(t) = \frac{1}{1+e^{-t}}$$

Where,

$$t = \hat{\beta}_0 + \sum_{i=1}^{m} \hat{\beta}_i X_i$$

X_i being the relevant regressors. The function looks like this:

Figure 10: Logistic Function



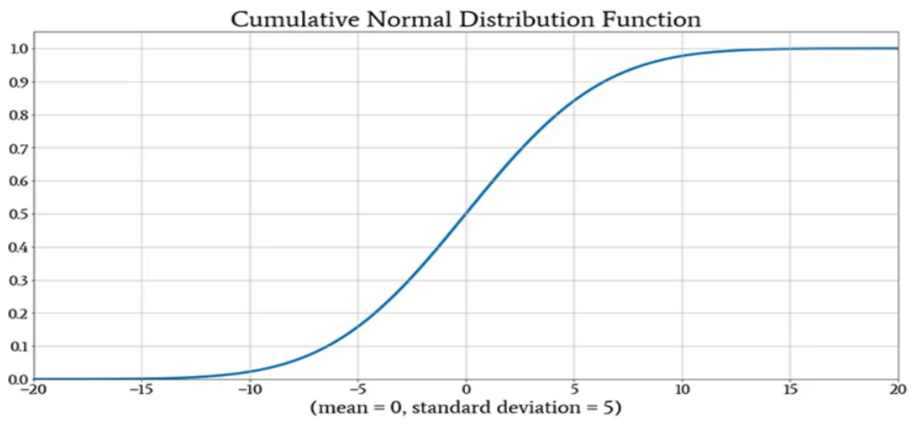
2.3 Probit Regression

Similarly, the Probit Model estimates the probability of belonging to a certain class (binary classification) by using a set of predictors (independent variables) X but uses the cumulative normal distribution for 'activation', again, to default probabilities that lie between 0 and 1. Thus the prediction is given by:

$$\hat{y} = \Phi(\hat{\beta}_0 + \sum_{i=1}^{m} \hat{\beta}_i X_i)$$

And as we all know, the cumulative normal distribution looks like this:

Figure 11: Normal CDF



2.4 Example using the Iris Data Set

We must install the following packages:

```
!pip install sklearn
!pip install tensorflow
!pip install statsmodels
```

and load them into the kernel:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import statsmodels.api as smf
```

Next, we will load a dataset that contains binary classification data. The most common used example for this is the iris dataset that comes with scikit-learn.

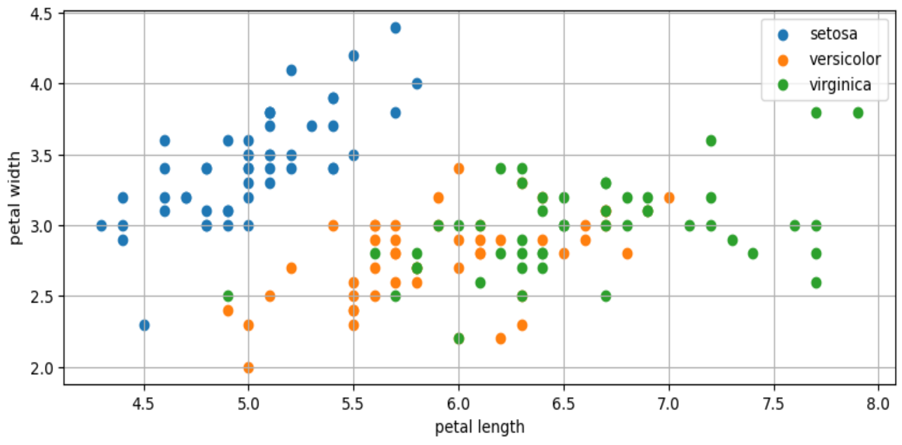
```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
Y = iris.target
```

X are the features that we are going to use to train our model, these are and can be understood as the regressors of our Logit/Probit. For this particular case these X was given to us, since the Iris data set provides with the characteristics of the plants. However, when training a real world model, one has to come up with this X, both in terms of the variables (which ones do we think will help us solve the problem) and their parametrization. When dealing with a classification problem, we might find it useful to plot the distribution of the classes. To do this we will use matplotlib package:

```
# Plot petal length and petal width
plt.figure(figsize=(10, 4))
plt.scatter(X[Y==0, 0], X[Y==0, 1], label=iris.target_names[0])
plt.scatter(X[Y==1, 0], X[Y==1, 1], label=iris.target_names[1])
plt.scatter(X[Y==2, 0], X[Y==2, 1], label=iris.target_names[2])
plt.legend()
plt.grid(True)
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.show()
```

And we get the following result:

Figure 12: Plotting Three Classes



Probit and Logit are binary classification models, meaning they only do prediction for two classes. Thus, for this example we will stay with only two classes and two features:

```
X = iris.data[iris.target != 2, :2]
y = iris.target[iris.target != 2]
```

To start training our classifiers, we need to divide the data set into training and test/-validation data. To do this we can use sklearn function `train_test_split()`.

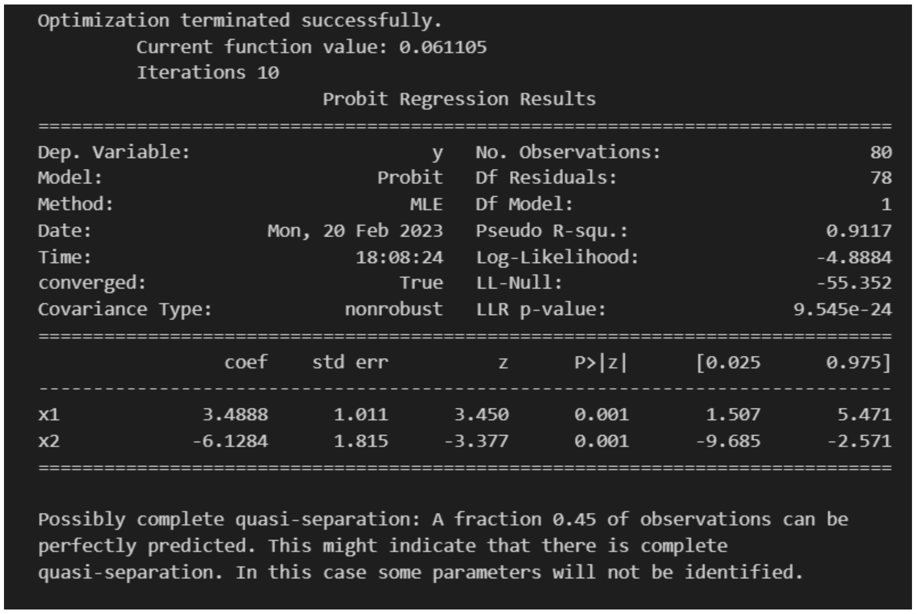
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)
```

Now we can call our functions, for the Probit prediction we will use the package `statsmodels` and the function `.Probit()`.

```
probit=smf.Probit(y_train,X_train)
probit.fit()
print(probit.fit().summary())
```

We get the next metrics and coefficients:

Figure 13: Plotting Three Classes



```
Optimization terminated successfully.
         Current function value: 0.061105
         Iterations 10
                          Probit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   80
Model:                         Probit   Df Residuals:                       78
Method:                           MLE   Df Model:                            1
Date:                Mon, 20 Feb 2023   Pseudo R-squ.:                  0.9117
Time:                        18:08:24   Log-Likelihood:                -4.8884
converged:                       True   LL-Null:                       -55.352
Covariance Type:            nonrobust   LLR p-value:                 9.545e-24
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1             3.4888      1.011      3.450      0.001       1.507       5.471
x2            -6.1284      1.815     -3.377      0.001      -9.685      -2.571
==============================================================================

Possibly complete quasi-separation: A fraction 0.45 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

To obtain the accuracy, we must do the prediction in the test set a bit by hand:

```python
params = pd.DataFrame(probit.fit().params, columns=['coef'])
df_test = pd.DataFrame(X_test, columns=['x1', 'x2'])

# Defining the probit function:
import scipy.stats as si
def normsdist(z):
    z = si.norm.cdf(z,0.0,1.0)
    return (z)

# Doing the prediction:
df_test['y_pred'] = df_test['x1'] * params['coef'][0] + df_test['x2'] * params['
    coef'][1]
df_test['y_pred_Probit'] = normsdist(df_test['y_pred'])
df_test['y_pred_0_1'] = np.where(df_test['y_pred_Probit'] > 0.5, 1, 0)
df_test['y_actual_0_1'] = y_test

print('Accuracy of Probit Model on test set: {:.2f}'.format(accuracy_score(df_test
    ['y_actual_0_1'], df_test['y_pred_0_1'])))
```

And we obtained an accuracy in validation of 100%. Now for the Logistic Regression we can use the package `sklearn` and the function `LogisticRegression()`:

```python
logit = LogisticRegression(max_iter=1000)
logit.fit(X_train, y_train)
y_pred_logit = logit.predict(X_test)
acc_logit = accuracy_score(y_test, y_pred_logit)
print("Accuracy of the Logit model:", acc_logit)
```

And we obtain again a 100% accuracy in validation. **As an exercise, repeat this but for the versicolor and virginica classes.**

3 Fixing Over-fitting in Probit and Logit: Regularizers

Recall from 1.5, that there are three possible outcomes when training a ML model: (1) under-fitting, (2) matching the true data generating process and (3) over-fitting. In simple terms, regularization is going from (3) to (2). In a more technical sense, *regularization* are a set of strategies designed to reduce the test (validation) error, possibly at the expense of increased training error. Out of these strategies the most relevant for ML models are: L1 and L2 regularization.[6]

LASSO (Least Absolute Shrinkage and Selection Operator) or *L1 Regularization* is a type of regularization method that is used to **prevent over-fitting** in linear and logistic regression models. It adds a penalty term to the ordinary least squares (OLS) objective function, which encourages the coefficients of the model to be small and close to zero. This helps to reduce the number of features included in the model and increase its interpretability.

L2 Regularization or *Ridge Regression* is a method of reducing over-fitting in Machine Learning and Deep Learning models by adding a penalty term to the loss function that discourages large weights in the model. The penalty term is proportional to the square of the L2 norm of the weights. Let's take a look at both graphically with an image taken from Hastie, Tibshirani, and Friedman (2008).

[6]For a complete revision of all these regularizers look at Goodfellow, Bengio, and Courville (2016) or the following link

Figure 14: Plotting LASSO and Ridge Regression

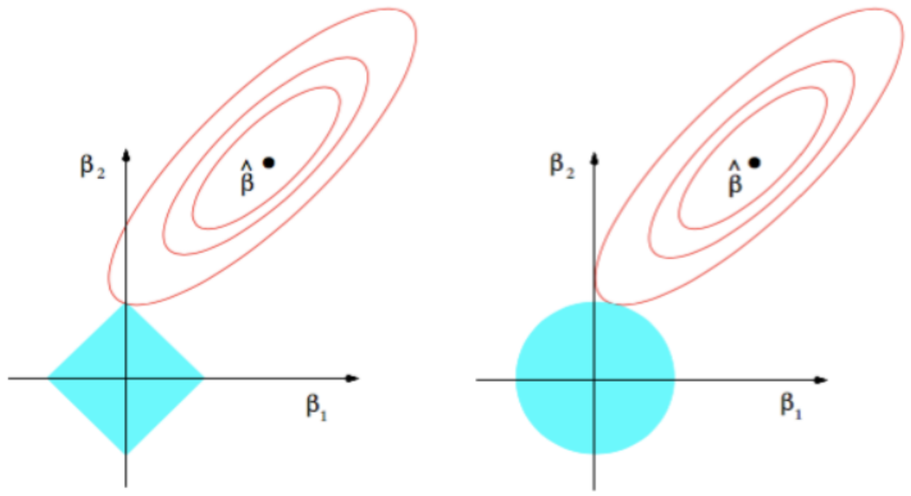


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

Mathematical Representation: L1 and L2 Reg

Let $J(\Theta; X, y)$ be the objective function of a certain model with parameters (weights) Θ, which is trained with features X and labels y. Thus the regularized version of this function is defined as follows:

$$\hat{J}(\Theta; X, y) = J(\Theta; X, y) + \alpha\Omega(\Theta)$$

Where $\Omega(\Theta)$ is known as the norm penalty term applied to parameters Θ, and α is another hyper-parameter that weighs the relative contribution of the norm penalty term relative to J. Parameter norm penalty is technically similar to imposing a constraint on the weights. Notice that if α increases, the constraint area becomes smaller, thus the model is more heavily regularized.

The difference between L1 and L2 reg[7] is solely on the way $\Omega(\Theta)$ is specified. For **L2 reg** we have that:

$$\Omega(\Theta) = \frac{1}{2}||w||_2^2$$

Adding this term affects the parameter's update, modifying the learning rule to multiplicatively shrink the weight vector by a constant factor on each step. In contrast, **L1 reg** adds the following term:

$$\Omega(\Theta) = ||w||_1 = \sum_i |w_i|$$

This term differs from L2 in two ways: (1) the first one is that the reg. contribution to the gradient no longer scales linearly with each w_i and (2) L1 reg solution is more sparse, since a lot of parameters result in an optimal value of 0. This sparsity condition could be used as a feature selection mechanism.

3.1 Adding L1 and L2 in Python

L1 and L2 Reg can only be applied in a OLS type of context, where there is a vector to take the norm of. Thus, it cannot be used for Random Forest. However, is as easy as this for Logistic Regression:

```python
# For L1
logit = LogisticRegression(penalty = 'l1', max_iter=1000)

# For L2
logit = LogisticRegression(penalty = 'l2', max_iter=1000)
```

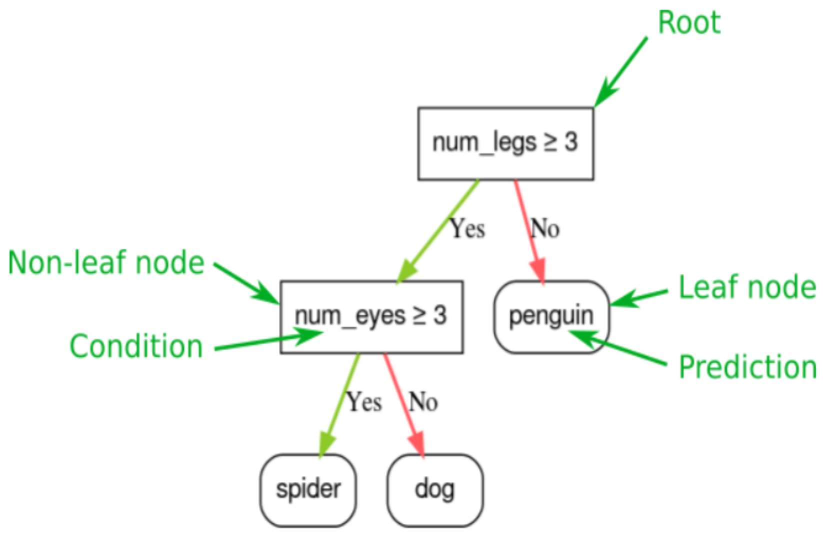
[7]Other names for these regularization techniques more commonly used in ML are: weight decay (L2) and Tikhonov Regression (L2)

4 Random Forest

Random forest is an ensemble learning algorithm that combines multiple decision trees to improve the accuracy and stability of predictions. It works by creating a large number of decision trees, each based on a random subset of the training data and a random subset of the features.

Decision trees are models composed of a collection of "questions" organized hierarchically in the shape of a tree. The questions are usually called a condition, a split, or a test.

Figure 15: Decision Tree



One could build these conditions manually, however, recall that we are using a *data-driven approach*, thus these conditions and their outputs are learnt as the model consumes more and more labeled data Google Developers. The collection of these decision trees is called a **forest**. And since these trees are chosen at random, thus the name *random forest*.

Finally, the algorithm then uses a combination of the decision trees to make a final prediction. These make the model more robust and stable than a normal decision tree. Its main advantage with respect to Probit and Logit is that it is a more complex model, which according to the literature, performs better in big data sets with a lot of features. Also, you still need to choose your features, but not how you parametrize them. The main disadvantage is that we must choose a set of hyper-parameters before start training our model.

Random forest has 4 main *hyper-parameters* that we choose: The number of estimators (number of trees in the forest), the depth of the trees, the minimum number of samples required to split an internal node and the minimum number of samples required to be at a leaf node. To solve the problem of over-fitting, we will turn to hyper-parameter tuning, since there are no regularizers for this model.

4.1 Example using the Iris Data Set

We will go over DataCamp's example together. We load the necessary packages into the kernel:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

We will again work with the Iris data-set, giving it facility to use.

```python
from sklearn import datasets
iris = datasets.load_iris()
```

However, we will now work with all the classes and all the features. To do this, let's transform the iris object into a Data Frame:

```python
data=pd.DataFrame({
        'sepal length':iris.data[:,0],
        'sepal width':iris.data[:,1],
        'petal length':iris.data[:,2],
        'petal width':iris.data[:,3],
        'species':iris.target
        })
```

We generate the label y and the matrix with our features X, and generate the training and the test sets:

```
    X = data[['sepal length', 'sepal width', 'petal length', 'petal width']]
    y = data['species']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state = 3)
```

For this case, we will default every *hyper-parameter* except the number of estimators (which we will set at 100) and later we will see how to obtain the set of parameters that maximizes the accuracy of the Forest.

```
    clf = RandomForestClassifier(n_estimators = 100)
    y_pred = clf.predict(X_test)
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)
```

And we obtain the prediction accuracy:

```
    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

We obtained an accuracy in test of 90%, which is pretty good (can we do better?). Let us first observe which features contributed most to this prediction, i.e. were more important:

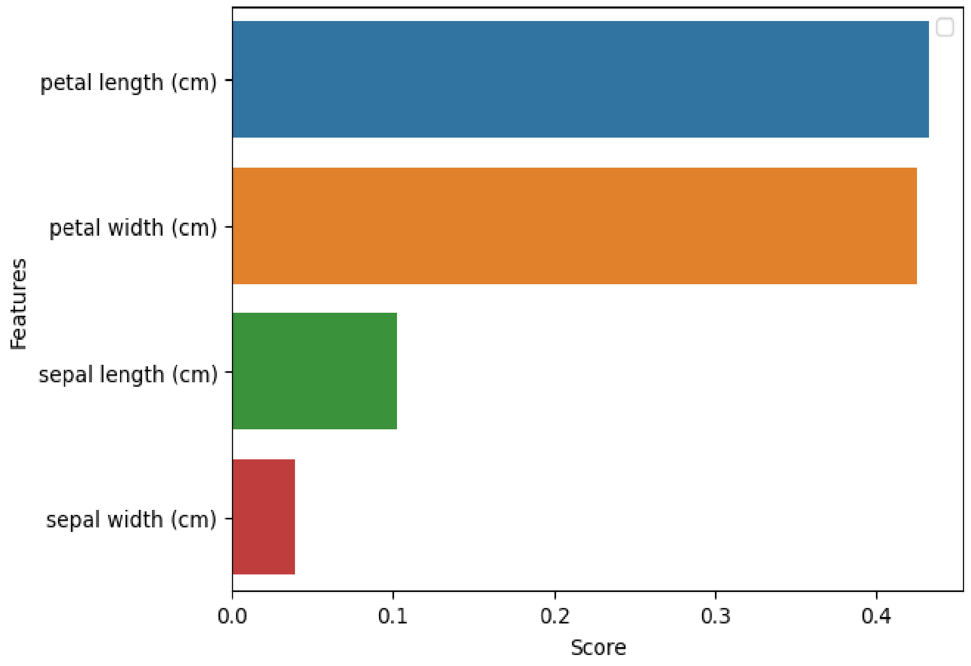
```
    feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).
        sort_values(ascending=False)
    feature_imp

    petal length (cm)    0.432701
    petal width (cm)     0.425739
    sepal length (cm)    0.102160
```

```
sepal width (cm)      0.039400
dtype: float64
```

And we can also visualize them in a graph:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.barplot(x=feature_imp, y=feature_imp.index)

plt.xlabel('Score')
plt.ylabel('Features')
plt.title("Feature Importance")
plt.legend()
plt.show()
```

Figure 16: Feature Contribution to the Prediction



Finally, we observe that although this level of validation accuracy is really high, there is some over-fitting since the training accuracy is 100%.

```
print("Accuracy:",metrics.accuracy_score(y_train,  clf.predict(X_train))*100)
```

To solve this, we can automatically choose between a set of hyper-parameters to see which one gives the model with the best validation accuracy. To do this we will use `sklearn` funtion `GridSearchCV`. We first define the set of parameters:

```python
from sklearn.model_selection import GridSearchCV

param_grid = {
    "n_estimators": [10, 50, 100],
    "max_depth": [3, 5, 7],
    "min_samples_split": [2, 3, 4],
    "min_samples_leaf": [1, 2, 3],
}
```

We estimate all the Random Forests with all the possible combinations of these parameters:

```python
grid_search = GridSearchCV(clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best parameters
print("Best parameters:", grid_search.best_params_)
```

The best parameters were n˙estimators: 10, max_depth: 5, min˙samples˙leaf: 1 and min˙samples˙split: 3. Now we can re-train our model with these set of parameters and see if the accuracy increased:

```python
from sklearn.metrics import accuracy_score
y_pred = grid_search.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy after Grid Search:", acc)
```

An we obtain an accuracy of 93.33% in test, which is a significant increase at the cost of a few lines of code.

References

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf. Springer.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2008). *The Elements of Statistical Learning*. https://hastie.su.domains/Papers/ESLII.pdf. Springer Series in Statistics.