Machine Learning types

1. Supervised

weather, future based on features.

Example:

house price:
size, bedrooms, year built =>
supervised ML algorithm =>
trained model =>
predict price.

2. Unsupervised

analyze input data and identify groups of data that share the same traits, detect voice from overlapped voices, future based on groups, clusters.

Example:

recording a room with people talking, values recorded:
pitch, intonation, inflection =>
unsupervised ML algorithm analyzes voices =>
classify model to create clusters of data that have certain speech patterns =>
isolate an individual voice from the mixture of voices.

Differences between supervised and unsupervised:

Supervised	**Unsupervised**
Value prediction	Identify **clusters** of like data
Needs training **data** containing value being predicted	**Data** has no cluster memberships
Trained **model** predicts value in new data	**Model** provides access to data by cluster

Python introduction for ML

Version: Python 3.5
Libraries 3.0:

- numpy - scientific computing
- pandas - data frames which hold, manipulate data
- matplotlib - 2D plotting
- scikit-learn - algorithms, pre-processing, performance evaluation, etc.

Jupyter notebook ide

https://www.continuum.io/downloads

Run *anaconda prompt* and write *jupyter notebook* to launch the web server.
Create a folder *Notebooks*, create a new *file*, open it and write the *code* in the tutorial.
Run code by pressing *shift enter*.

Markdown:

```
# Learning
This notebook contains some simple examples of using *Jupyter Notebook* and *Python3*
## Example 1: *Hello, World*
```

Code:

```
my_name="Jerry"
hello_statement="Hello, " + my_name
print(hello_statement, end="\n\n")
x = 10
for j in range(1, 5):
    x = x + j
    print("j={0} x={1}".format(j, x))
```

Machine learning workflow

Orchestrated and repeatable pattern which systematically **transforms (into a format) and processes** information to **create prediction** solutions (performance criteria that we specify).

Steps in the machine learning workflow

1. **Asking the right question -** will guide the following steps: goals we want to achieve, the data we need and the process we can perform.
2. **Preparing data -** gather the data we need to answer the question (tricky because the data can be incomplete, inaccurate, and conflicting).
3. **Selecting the algorithm -** not an easy task as there are many algorithms available; the question will help us select the proper algorithm; once it is selected we need to use a subset of the data we have to train the algorithm. This training will create a training model.
4. **Training model -** predicts results on similar data.
5. **Testing the model -** with new data that wasn't used to train the model. This testing will generate statistics with which we can determine if the model meets our requirements or needs to be refined. If refinement is needed we need to go back: get more data, change algorithm, adjust training parameters and sometimes some combination of all three.

Guidelines

1. Early steps are most important, each step depends on previous steps.
2. Expect to go backwards, later knowledge affects previous steps.
3. Data is never as you need it, not in the right format, data will have to be altered.
4. More data, better results.
5. Don't pursue a bad solution, reevaluate, fix or quit.

Example following ML Workflow

Let's use the workflow to build a solution that will predict whether a person will develop diabetes.

Asking the right question

We will convert the general goal of whether a person will develop diabetes into a statement that will guide our solution.

Predict if a person will develop diabetes.

Need statement to direct and validate work. A question that will drive the data we gather, how we mold the data, how we interpret the solution and the criteria we use to validate the solution.

We need a statement to define the end goal, starting point and how to achieve the goal.

Solution statement goals:

- Define scope (including data sources)
- Define target performance
- Define context for usage
- Define how the solution will be created

Based on the first statement we can see if there are some *assumptions* that can help with the definition of the scope of the solution.

Scope and Data sources:

- Understand the features in data
- Identify critical features
- Focus on “at risk” population
- Select data sources:
    - Pima Indian Diabetes study is a good data source

The review of the data on the american diabetes website shows that diabetes has several factors that can help guide us:

- age (older people are more likely to get diabetes),
- race (african american, american indians, asian americans are more likely to have diabetes).

There are several data sources for diabetes, but one stands out as the data source for our purposes. It is the Pima Indian Diabetes study available at the UCI machine learning repository. This is a study of diabetes in women and indian population located in arizona. The study was conducted in the 90s and is a classic data set. When we incorporate the pima data set, the statement changes too:

Using ***Pima Indian Diabetes data****, predict which people will develop diabetes.*

Important aspect is *accuracy*.

Performance Targets

What prediction accuracy should we reasonably expect?

Binary result: True or False (Coin Flip = 50% accuracy, obviously we want to be more accurate than that).

Historically disease prediction is notoriously bad, which is due to genetic difference.
70% Accuracy is common target.

This changes the statement to:

*Using **Pima Indian Diabetes data**, predict with **70% or greater accuracy** which people will develop diabetes.*

Context

Disease prediction
Does it mean we are absolutely sure about the prediction?
Medical research
What are the common practices in the medical research?
There are many variables in each individual and many of these are unknown to how they affect the probability of developing a disease.
Unknown variations between people
So medical research will predict **how likely** is it for someone to develop the disease.
Likelihood is used.

*Using **Pima Indian Diabetes data**, predict with **70% or greater accuracy** which people are **likely** to develop diabetes.*

Solution creation

For our case we will use Machine learning workflow:
- Process the Pima Indian Diabetes data
- Transform data as required

*Use the **Machine learning workflow** to **process** and **transform Pima Indian Diabetes data** to create a prediction model. This model must predict which people are **likely** to develop **diabetes** with **70% or greater accuracy**.*

Preparing the data

- Find the data we need
- Load into python and inspect and clean the data
- Explore data: structure, make any necessary alterations
- Mold the data into **Tidy** data which works best with machine learning algorithms

We will do all this in python and jupyter notebook.

Tidy data - it is normal form 3, but it is a bit more pragmatic than normal forms and focuses in presenting the data in a clean readable format. Data is tidy when it was produced by a well designed view. Getting the data into a tidy form can take a little time (50 - 80% of a ML project is spent getting, cleaning, and organizing data):

- Each variable is a column
- Each unique observation is a row
- Each type of observation unit is a table.

Data Rule #1: Closer the data to what you are predicting, the better.
Data Rule #2: Data will never be in the format you need.

In Python we will use **Pandas DataFrames** to make this reformatting easier: a column for each feature and a class, and a row containing the observations of these features and a class for each patient.

Feature != column:
Feature is something used to determine a result, a **column** is a physical structure that stores a value of the feature or a result.

We will need to refer to the definition of the columns from time to time. To ensure our definitions are available, we will add a feature definition table to our notebook:

Columns to eliminate:

- Not used
- No values
- Duplicates - or provide the same info in a different format, we can visually inspect columns to see if they really are the same, but visual inspection is error prone and does not deal with the critical issue of **correlated columns**.

Correlated columns state the same information in a different way, such as ID and a text value for the ID, these don't affect the result, they can introduce bias and confuse algorithms because some algorithms treat every column as being independent and just as important.

For example: we predict the price of a house with area in (sq ft) and (sq m), both being important.

Pandas makes it easy.

Molding the data:

Check **data types**, make sure they **all are numbers**: convert true to 1 and false to 0 with map function of panda.

Data Rule #3: Accurately predicting rare events is difficult.

Means we need to **check** how many cases of diabetes we have in db vs the non diabetes cases we have, e.g. the **ration true/false.**

Data Rule #4: Track how you manipulate data.

When you manipulate data it is very easy to change the meaning of the data; this can be done intentionally, but it can also be done unintentionally.
By keeping track you can always reproduce the data and you can see what went wrong.
Keeping track of changes on data is done automatically in Jupyter Notebook; use Git as well.

So, what we did so far with the data:

- Use pandas to read the data
- Identify correlated data using visualizations
- Clean data by removing the column correlated
- Molded data by changing data types of columns into what we needed
- We checked the ratio of true and false to ensure we have a good ratio for prediction
- We discussed data rules that guided our work

Selecting the algorithm

Role of the algorithm

It is the engine that drives the entire process.

We will use training data containing the results we want to predict and the features values we are using to predict that result. The training function *fit()* (in scikit-learn) executes the *algorithm* and processes the *training data;* the data is analyzed with respect to the mathematical *model* associated with the algorithm. The algorithm uses the results of the analysis to adjust internal parameters to produce a model that has been trained to best fit the features in the training data and produce the associated class result. This best fit is defined by evaluating a function specific to a particular algorithm. The fit *parameters* are stored and the model is now said to be *trained*.

Later the model is called via the prediction function *predict()* (in scikit-learn). When this function is called, real data is passed to the trained model using only the features in the data, the model uses its parameters values set in training and evaluates the data's features and predict class result diabetes or not for this new data.

Perform algorithm selection

Over 50 algorithms (in scikit-learn).

Compare algorithms based on factors. Decide which factors are important. Each data scientist has his own decision factors that he uses to compare algorithms. With time you will also have your own list of important factors to base your decision on.

Factors:

1. *Learning type* supported

Looking at our solution statement we see that we aim for a prediction model, therefore we should look for algorithms in **supervised machine learning** that includes the prediction algorithms.

Now the number is reduced to 28.

2. *Result* type the algorithm predicts

It can either be **regression** or **classification**.
Regression - continuous set of values, like in the determining price of the house example, we used the features of the house, e.g. size, number of bedrooms, etc., we put these features into an equation and produced the price. Any **change** in the feature resulted in a **direct change** in the price.
Classification - discrete set of values, such as small, medium, large; 1-100, 101-200, 201-300; true and false. **Changes** in the feature values **may or may not** change the classification!

Diabetes is true or false which is a **binary** outcome. Since it is binary we have to choose algorithms that are for classification problems and eliminate the ones that are not, in particular choose the **binary classification algorithms**.

Now the number is reduced to 20.

It did not reduce the number that much because most of the algorithms support both regression and classification.

3. *Complexity* of the algorithm

Eliminate algorithms that “ensemble” algorithms in order to keep it simple.
These are special algorithms that combine **multiple** other algorithms under **one interface**.
These are more often used and we need to tune the model to **increase performance** and we are still trying to do our initial training. These can be difficult to debug.

Now the number is reduced to 14.

4. Whether it is *basic or enhanced*

Enhanced algorithms are variations of **basic** algorithms. They have been improved for better performance, have additional functionality and are more complex.

Basic algorithms are less complex and easier to understand.

Discuss 3 candidate algorithms

1. Naive Bayes

- Based on likelihood and probability

It is based on Bayes theorem that calculates the probability of diabetes by looking at the likelihood of diabetes based on previous data combined with the probability of nearby feature values. In other words, how often does a person having higher blood pressure correlate to diabetes;

- Every feature has the same weight

it makes the naive assumption that all the features that we pass are independent of each other and have equally impact result, e.g. blood pressure is as important as BMI, which is as important as number of pregnancies.

- Smaller amount of data needed to be trained

Because it allows for fast convergence due to same weight on features.

2. Logistic Regression

- Confusing name

In statistics *regression* means continuous values, but logistic regression returns a binary result.

- Relationships between features are weighted

It measures the relationship between features and weights their impact on the result. The result value is mapped against a curve with 2 values, 1 and 0, which is equivalent to diabetes or no diabetes.

3. Decision tree

- Binary tree structure
- Node contains decision

With each node making a decision based upon the values of the feature; at each node the value makes us go down one path or another;

- Requires enough data to determine nodes and splits

A lot of data is needed to find the value which defines taking one path or another. Tools are available to produce the picture of the tree. It is easy to visualize how the trained model works.

Select one initial algorithm

Naive Bayes Algorithm

- Simple - easy to understand and result type yes or no;
- numerical characteristics useful for an initial alg.
- Fast, up to 100X faster, training time is important.
- Stable to data changes.

Training the Model

Split data into:
70% training data - produce the model with the algorithm
30% testing data - evaluate the model against unseen real-world examples

Minimize the set of features in order to get faster training

Use scikit-learn to train and evaluate the model and access algorithms, e.g.:

- Split data (train_test_split from cross_validation)
- Pre-process data - missing data or hidden null values in plain sight is a common problem, if few data then deleting is ok, but if it is too much data then replacing with other values is the option, also called Imputing
- Feature selection
- Model training
- Model tuning for better performance
- etc.

Imputing (scikit-learn Imputer class)- replacing with:

- Median, mean, or other statistic
- Expert knowledge derived from the remaining features in the row as determined by an expert medical researcher

Evaluate the Model

Evaluate with test set:

See how well it predicts the training set and then how well it predicts the testing set.

But do not trust these numbers: check:

Confusion Matrix - compares the predicted vs actual results for diabetes; columns are the predicted values: left column is predicted True and the right column is predicted False; the rows are the actual values: top row is actual true and the bottom row is the actual false. FP - false

positive, actual False diabetes, but predicted to True diabetes. TN - true negative, actual False diabetes and predicted False diabetes. All correct predictions are located in the diagonal of the table, so it is easy to visually inspect the table for prediction errors, as they will be represented by values outside the diagonal.
Perfect Classifier would have only numbers on the columns and on the outside - only zero:
[[80 0]
[0 151]]

Actual vs Pred	Predicted True	Predicted False
Actual True	52 (TP)	28 (FP)
Actual False	33 (FN)	118 (TN)

Classification report - based on Confusion Matrix values.
We want to focus on the probability of a True result, meaning that the patient has the disease, which is the **Recall** for class 1 diabetes, known as true positive rate (TPR) or sensitivity, or hit rate, it is “how well the model is predicting diabetes when the result is actual diabetes”.
Mathwise, Recall = TP/(TP+FN) = TP/PredictedTrue, and it is 0.65, but we need this number to be >=70%.
Precision is known as the positive predictive value PPV = TP/(TP+FP) = TP/ActualTrue, is how often the patient actually had diabetes when the model said they would. We need to increase this number, meaning fewer FPs.

Performance Improvement

- Adjust current algorithm - hyper params let us tune the model’s performance, but Naive Bayes doesn’t have them
- Get more data or improve the data we have, but we are already using all the data of the Pima set and we’ve already processed it a little bit
- Improve how we train the data
- Select newer algorithm that works better with our data - more complex alg. can make everything easier

4. Random forest algorithm

Ensemble algorithm, based on Decision Trees, creates multiple trees with random subsets of the training data, the results of these trees are averaged to avoid overfitting.

The prediction with the training data generates .99 accuracy, whereas with the test data .71 accuracy. The difference is big, which is a sign of **overfitting**. The model learned the training data too well.

2 primary techniques to **handle overfitting**:

- Special **tuning** parameters called **hyperparameters**, using these we can define how the alg. learns and operates. The hyperparameters that impacts overfitting goes by different names but the general term is **regularization**. Setting the value of the regularization hyperparameter allows the developer to control how much the alg. focuses on precisely fitting every corner case of the training data. E.g. lambda in a function, different settings for lambda define how much values produced in the trained model are dampened via the regularization term. This dampening decreases the accuracy of the model on trained data, but potentially increasing the accuracy of the model on the test data. Each alg reacts diff to the data and the values of the hyperparameters. Some experimentation is required to get the best values for each hyperparameter with the specific set of data. E.g. y = x1+w2x2^3+w3x3^8-f(W)/lambda; the effective regularization in this particular example equation is larger when the value of regularization (lambda) is smaller. The same relation is common but it is not universal, therefore read the doc on your alg. to understand how the regularization works in your alg.
- **Cross-validation** - use multiple subsets of the training data during the training process.

These techniques are not mutually exclusive; both can be used at the same time. With these techniques we can come up with a compromise between the accuracy on training data and accuracy with testing in real world data. This is often called the **bias** or **variance trade-off**. This is needed to be considered in almost all ML algorithms. We need to sacrifice some perfection in training for better overall performance with test in real world data.

Re-evaluate the options for further performance improvement: adjusting the algorithm by tuning the hyperparameter that might influence overfitting, which is oob_score=True and it was already set (confused now), getting more data would be a solution but we are already using all the data from the Pima study, we could do cross-validation for improving the training but it wouldn't have much impact with Random Forests because it is doing cross-validation by building multiple trees and comparing them, and finally we could switch algorithms to a simpler one, logistic regression.

The recall is still not right with the Logistic Regression model. We can try different values for the C regularization parameter by running a loop. We found that the optimal C value is 1.4 where the Recall is 0.61, which is still not good.

Our data had more Non diabetes (65%) than diabetes (35%), and this might be an issue. It means we have unbalanced classes. This imbalance can decrease performance in an algorithm. Logistic regression has a hyperparameter that compensates class imbalance. It is a equivalent to balancing a scale and the result is a shift in the predicted class boundary. So we should run the same loop but with the class_weight=balanced. We find that the Recall is 0.738 for the C=0.3. Now the accuracy of prediction on test data looks good.

We only tested this on only 1 set of test data and by looping through the values, we sort of cheated by setting the regularization value based on the test data. The check the of the accuracy with the test data affects the results, that’s we are suddenly tuning the model to the test data. It would be great if we could use test data only for the final test and have some other data for tuning. We could do this by:

- **Splitting** the data into **3 data sets**: training 50%, validation 25%, testing 25%, where validation is used to tune-validate the model and the testing data is only used for the final test. The issue is in how to choose the validation set when we don’t have enough data and the mitigation of overtraining.
- We can use **k-fold cross-validation** instead. For each fold determine the best hyperparameter value, next set the model hyperparameter value to average best. In order to use this built-in mechanism from the scikit-learn, use the algorithm with the extension CV and specify the number of folds, or how the regularization hyperparameter is determined. However, CVs take longer to run.

Running the LogisticRegressionCV

Setting Cs to an integer value defines the number of values it will try trying to find the best value for C for each fold.

More classic datasets at archive.ics.uci.edu/ml/