Applied Machine Learning with Python (Uof Michigan Coursera):

Sean al-Baroudi

July 21, 2019

Chapter 1: Machine Learning Landscape:[1]

What is Machine Learning, Why Use it?

- Define it as giving computers the ability to learn without explicitly programming them.
- Any problem that cannot be solved by listing a finite list of programming rules is open to solution by machine learning.
- ML is most useful for:
  - Complex Problems where there is no good programming solution.
  - Complex Problems where a list of rules is insufficent for solution.
  - Fluctuating Environments
  - Human Learning: Giving insights to complex systems.

Types of Machine Learning Systems:

- Algorithm catagories include: Supervised, Unsupervised and Semi-supervised
- Reinforcement learning
- Instance vs Model Based learning.

Supervised Learning:

- Data: both features and the targets (labels) are provided (y,X)
- Functional Structure: $\hat{y} = f(X)$ and $\epsilon \propto |\hat{y} - y|$
- The provided labels can be class names, or numerical values (generally).
- Training: You give many (X,y) pairs, and learn based on some metric of ϵ.
- Examples: All forms of Regression, Support Vector Machines, Decision Trees and Forests, Neural Networks.

Unsupervised Learning:

- Data: Only feature data is provided (X).
- Functional Structure: $f(X)$.
- Generally, these methods use various techniques to pick out structure, or form representations of the data.
- there is no training set, or gradient descent with error
- Types of Unsupervised Learning:
  1. **Clustering:** Forms groups naturally, using numerical metrics. k-Means, K Nearest-Neighbours, Expectation Maximization.
  2. **Visualization and Dimensional Reduction:** Seeing underlying structure in detail, by boiling it down to some basis representation (Linear Algebra Tricks). Principle Component Analysis, Singular Value Decomposition, t-SNE
  3. **Association Rule Learning:** Aprior, Eclat (???)

Semi-Supervised Learning:

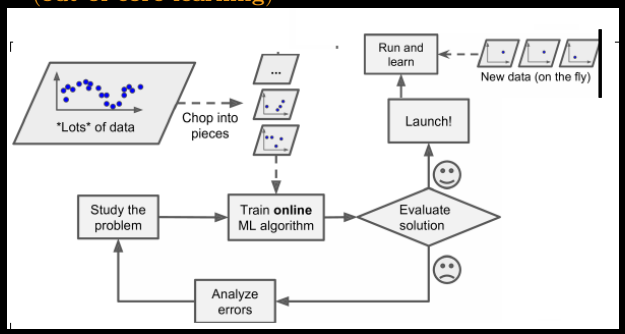
- This involves algorithms that combine elements of unsupervised and supervised learning.
- For example: The first stage might involve learning unsupervised representations, and then the second stage could involve labelling such representations for detection later.
- Data: X, then y later.
- Model: f(X), y = g(f(X))...

Reinforcement Learning:

- Here, our model is represented as an agent with actions that can be chosen (by policy rules).
- Its goal is to maximize positive reward, and minimize punishment.
- Not the focus of this document!

Batch and Online Learning:

- **Batch Learning:** System is trained using all available data. - not suitable for large data sets, or limited computing power.
  - if a new model is needed, one has to retrain the model all over again.
  + simple in concept; online learning not suitable for some problems.
  + for smaller problems and data sets that don't change much.
- **Online Learning:** Learning is done in batches (single cases, or mini-batches).
- Note: "Online" should not be taken literally. It can just involve batching without being linked to real time data. A better term is incremental learning.
  + low computational and memory resources, learning steps done very quickly.
  + learning can be set up to improve from new data in the real world.
  - with enough anomolous or bad data, models can learn and end up performing worse.
  + suitable for continuous flow applications (such as time series or streaming data).
  + can easily be unrolled if poor learning occurs.
  + useful for any problem that is so huge it can't fit in memory (**out-of-core learning**)



Instance vs Model Based Learning:

- The **goal** of ML is to perform well on data that has not been seen.
- **Instance-Based Learning:** Here, we just encode all of examples and memorize them. We then perform a measure of similarity, and compare new examples to instances we know about. **Clustering and Classification can be done with this simple method.**
- **Model-Based Learning:** We make a parameterized model to represent the data. Applying the model to new data allows us to make predictions.
- To evaluate different models, we need performance measure.
- **Utility Function:** Measures how well your model is doing.
- **Cost Function:** Measures how bad your model is doing. Often formulated in terms of model deviation between targets and predictions.
- **Basic Steps in Training a Model:**
  1. Study the data, engineer features.
  2. Select a model
  3. Train model on the training data. Evaluate a variety of models using test data.
  4. The best model is used to make predictions on new cases and data (perform inference).

Main Challenges of Machine Learning Systems:

Problems with Data:

- **Insufficient Training Data:** Not enough data to learn the structure of the problem.

- **Non-Representative Training Data:** In particular, influences via outliers, or missing data of a particular group. Example: With regression, extreme missing values can completely shift your model and mess up extrapolation.
- **Poor-Quality Data:** If the data is noisy, signal to noise is so low that you can't detect the underlying patterns properly. Strategies to correct:
  - Imputation or Substitution for holes in the data.
  - Reaquisition of the data.
  - Filtering/noise separation (if possible).
  - Outlier detection, and removal.
- **Irrelivent Features:** You either select or derive new features. This requires quite a lot of insight for complex data.
-

Overfitting and Underfitting:

- It is **assumed** that for structured data (at least for those that measure a phenomenon), that there is some underlying relation, and that there is signal and noise for every dimension of the data.
- I also assume that the data set is not just randomly generated, for this section.
- Overfitting occurs when a model does well on the training data, but does not generalize well. It also means that the model is detecting patterns in the noise of the data, instead of the underlying relations themselves.
- Another way to think about overfitting is that our model is too complex for the underlying patterns being modeled.
- Solutions to Overfitting:
  - Reduce the complexity of your model.
  - Gather more training data
  - Noise filtering/reduction.
- **Regularization:** The process of constraining your model (reducing parameters), in order to reduce the risk of overfitting. This is quantified by a hyperparameter, often.
- **Hyperparameter:** A parameter of the learning algorithm, not the model itself. Think of it as a **meta-parameter**.
- **Underfitting** occurs when the model does not learn the training data very well. This often means the model is too simple.
- Solutions to Underfitting:
  - Selecting a more powerful model, with more parameters.
  - Feeding better features to the learning algorithm
  - Reducing constraints, or regularization hyperparameters.
-

Testing and Validation:

- After we have learned on some training data, we want to evaluate how well the model has generalized. You do this by trying it out on new cases.
- We get new cases by holding out a section of the data set - this is our **test set**.
- **No Free Lunch Theorem:** We don't know (a priori) which model (in terms of parameter complexity) will perform the best. We iteratively train different models, and then compare them on training and test set errors. We then use metrics to decide which model is "best".
- Practical Corollary: There is no best algorithm or model. Data is typically very complex - thats why you are using ML in the first place (if it was easy, you could just do it by eye/hand).
- Note: Measuring the generalization error on the test set is **not sufficient** for good model performance on unseen data (outside our data set). There are many reasons for this.
- To reduce the risk of poor real world performance, we can also introduce another partition of the data: the Validation Set. You replace the Test Set with the Validation Set, and use this to select the best model out of all the models. As a final test, you use the Test Set to check the generalization error before using the model in the real world.
- Sadly, Train+Test+Valiation comparisons is **not sufficent** for good model performance in the real world either. It can do better than just a Train+Test.
- Problems with Data Partitioning: The performance of our model is depedent on the particular partition of data we select. We also have to account for this error.
- You might also have limited data, and you can't spare more data for a validation set.
- **Cross-Validation:** We repeatedly sample a random subset of data for the training set, and use the rest for validation. Each model is tested multiple times on a different partition of data.
- You use Cross-Validation to investigate the best model (in the space of models you defined). You then select this model, and use a Train-Test partition to get the best possible parameters for this model.
-

Applied Machine Learning with Python:

Week1:

Introduction:

- Machine learning is the study of computer programs that learn from examples, and learn without having to be explicitly programmed.
- This is a course on Applied Machine Learning. It is not a course on going into the proofs of the algorithms (do this yourself).
- Learning how and when to use the algorithms is largely separate from understanding how they work (proof). Don't confuse these two.

Key Concepts in Machine Learning:

- **Unsupervised Learning:** finding useful structure in data with no labels available. You focus on different properties, and group/cluster these data points together. You can also derive properties by applying functions and operations (feature generation).
- A feature is just a property of an entity.
-
- **Basic Machine Learning Workflow:**
  1. Representation: Choose a feature representation, and your algorithm to apply.
  2. Evaluation: Define criterion that allows you to decide which algorithms are better or worse (relative to each other).
  3. Optimization: Once an algorithm has been selected, you find an optimal set of parameters.
  4. **[See workflow image in previous section]**
- There are literally hundreds of features you can choose; selecting or deriving the right features is not trivial.

An Example ML Problem:

- We wish to build a simple recognition classifier.
- Data: A set of fruits was measured, and the data compiled into a table. This data is associated with name labels. We wish to input the features, and output a name accurately.
- Data: (FruitFeatures, FruitName)
- In our table, rows are complete data points (data, label), each feature and label is represented in its own column.
- We have a special range value called "color score". It ranges from 0 to 1. With 0 = Fuschia, 0.2=Blue, 0.5=Green, 1=red. It maps the VL continuum onto a real number line, to further aide in classification for our example.
- We partition our data into test and training set. This is to stop our algorithm from just memorizing the data - which would result in poor results for new unseen data.
- Once the data is split, you examine the training data for the following issues:
  1. Missing Data
  2. Errors or Noise in the Data
  3. Non-Tidy columns
  4. Mixed Column data (data of different units).
  5. ...do you even need to use ML to solve this problem?
  6. ...do we need additional data (derived, or data joins). Example: links to images with GPS coordinates in the meta-data.
- Next, we visualize the data to see clustering in pair-wise scatter diagrams. If we see good clustering, this suggests that classification is possible. (Why)?
-

k-Nearest Neighbours:

- This is a Clustering Algorithm, that consists of the following parts:
- With this algorithm, we try to find the k closets points to our input point, and classify it based on what is locally around it.

- Clusters of points form classes.
- The entire training set is memorized by the algorithm. It just tries to find the closest points and predicts class from this.
- **Consequences and Implicaitons:**
- The boundaries between two classes are necessarily equidistant between those classes. This constraint dictates how the visualizations are drawn.
- Low Value of K: - more fragmented areas
  - more sensetive to noise
- High Values of K: +More Cohesive class areas.
  + More robust against noise, outliers.
  - More misclassification mistakes for training data: you will frequently see points form other classes, embedded in the wrong class.
  - less detail in class domains
- k is often odd, so that we can easily break ties.
  1. Distance Metric (2-norm usually)
  2. k: number of nearest neighbours
  3. Optional weighting function for k-nearest points. This can be an arbitrary function, or based on distance.
  4. Method to Aggregate Classes: Such as Majority Vote.

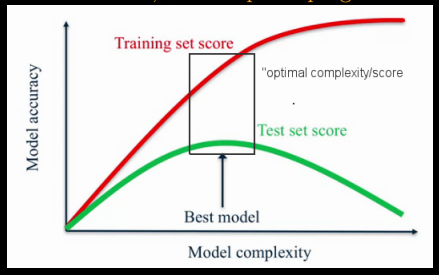
Week 2: Supervised Learning Part 1:

Introduction:

- **Model:** A mathematical description that describes a relationship between input and output variables.
- For inputs and outputs, we have the following names, depending on the field:
  - **Statistics:** I: Independent (Random) Variables, O: Dependent Variables (function of random variables).
  - **ML:** I: Features, O: Targets
  -

Overfitting and Underfitting:

- **Generalization:** An algorithms ability to give accurate predictions for new unseen data.
- Underlying Assumptions about test set:
  - future unseen data will have same properties and training data.
  - Note: This often doesn't occur; test and future data has additional features or complexities that weren't captured in the train data. Hence it performs worse.
  - For 99.9 percent of models, accuracy on train must be better than accuracy on test.
- **Overfitting:** When a model is too complex for the data it was given, and generalizes poorly. Intuitions:
  - The model starts to fit spurrous details and noise, parameters not constrained enough.
  - The model captures local variations more than the global trend, it is too sensetive.
- **Underfitting:** Model is too simple, and may even perform poorly on the training data. You will see high errors in both the test and training sets.
- In both cases, poor generalization can occur.
- When we seek out our best model, we keep stepping up the complexity of our model (underfitting), until we start noticing overfitting. Then we step back. An idealized curve of test and training scores is below, for a stepwise progression



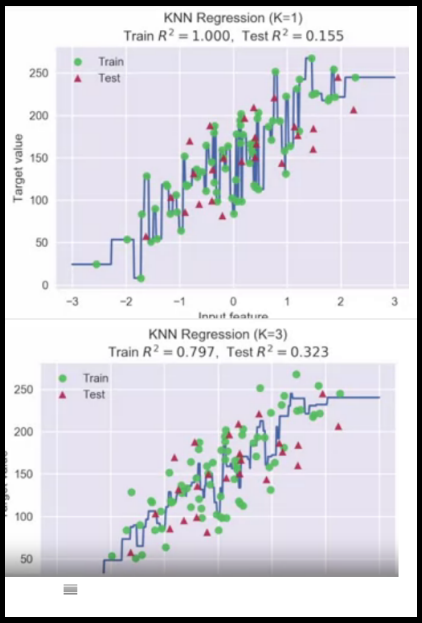
- **Curse of Dimensionality:** For hyper-dimensional data sets, data typically lives in corners in the space, and the distance between points is large for any pair of points. This makes supervised learning difficult.
- In general, for a space R^n, if we want to interspace p points along each axis (to make a cartesian grid) we need p^n. For p = 5, and n = 10, we need 9765625 points. For datasets with thousands of dimensions, its no wonder points end up in corners!

kNN: Classification:

- Much the same stuff as last week.
- k=1 gives perfect training predictions: we memorize the table of data so this must be so. Sharp, complicated boundaries result.
- as k increases, we get better test predictions, but worse training predictions.
- Boundary complexity inversely proportional to model complexity, in general.

kNN: Regression:

- With kNN Regression, we have our feature data X and now continuous target labels (y).
- **Idea:** Memorize a table of training points, for a given input point, find the closest points to it, and do a majority vote on the associated target labels.
- Some Observations about this algorithm:
  - Information needed to describe the "target lines (triangles)" goes down as k goes up.
  - we see the target lines are between the points more, as k increases.
  - it visually appears as a kind of straight line regression between local clusters of points (descriptively, not literally).



  - Again, for k=1 train set performance is perfect
- Error Score: R-squared: This is a metric to tell us how well our regression fits our data. **Some definitions:**

$$SS_{tot} = \sum (y_i - \bar{y})^2$$

$$SS_{res} = \sum (y_i - f(x_i))^2$$

$$R^2 = 1 - \frac{SS_{tot}}{SS_{res}}$$

- R-squared ranges between 0 and 1. A score of 1 means we fit perfectly.
- What is the purpose of SStot? Notice that: SStot $\geq$ SSreq, and that it normalizes our score between [0,1] for any regression. If we did not have it, our Rsquared value would range between [0,∞] for the set of all regressions; it stops scale interpretation problems, for every regression.
- kNN in general has fewer assumptions about the underlying phenomenon (compared to Regression).
-

Linear Regression: Least Squares:

- Also known as Ordinary Least Squares (no special objective function).

- Here, we try to fit a linear function that minimizes some objective measure (MSE, RSS). We have a weight (w) and a bias (b) to find.
- Solution: Calculate Directly (solve equations), or a gradient descent.
- Assumptions: This algorithm **assumes** that the output is a function of the inputs. Normally distributed residuals and error.
- RSS: Residual Sum of Squares:

$$RSS(w,b) = \sum_{i=1}^{N}(y_i - (wx_i + b))^2$$

- Mean Squared Error: Used ot estimate the the bias of a statistic, but can also be interpreted like RSS:

$$\frac{1}{n}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2$$

  Here our "predictor/statistic" is the linear model itself.
-
-

Ridge and Lasso Regression:

- These variants of regression focus on modifying the Objective function, to get better results in certain conditions.
- They both involve introducing a penalty term on the weights, which forces gradient descent to penalize large weights.
- **Regularization:** The process of stopping weights from getting too large. This is done to reduce overfitting.
- **Ridge Regression:** Invoves a 2-Norm penalty on the weights. Our objective function becomes:

$$RSS + \alpha \sum w_i^2$$

  Where α is a parameter we can tune. Effect: Tends to cause weights to stay small.
- **Lasso Regression:** Involves a 1-norm penalty on the weights:

$$RSS + \alpha \sum |w_i|$$

  Effect: Tends to cause small weights to zero out, focus on the main factors in regression.
- **Use cases for Lasso and Ridge:** They are typically used when you have many features (and weights).
  - Ridge: use when weights tend to medium large size (with OLS), and you want to cut them down.
  - Lasso: When you observe lots of small randomized weights, and you want to zero them out.

Feature Normalization:

- Some ML algorithms need features to be on the same scale. Feature normalization transforms features to remove the extremes.
- Examples of Algorithms that need this: SVM, NN, kNN.
- MinMax Scaling: For each feature, transform using the following formula:

$$MM(x_i) = \frac{(x_i - Min(X))}{(Max(X) - Min(X))}$$

  Where X is the entire set of features.
- Quartile Scaling:

$$QM(x_i) = \frac{(x_i - Q1(X))}{(Q3(X) - Q1(X))}$$

Polynomial Features with Linear Regression:

- In addition to our linear features, we have every combination of non-linear feature (think: Binomial Theorem).
- This blows up quickly (number of non-linear features), and increases the power of the model dramatically. It should be used sparingly.
- Note: The model produced is non-linear. But the framework to derive the optimal weights is still a linear one (we just use OLS framework with some modifications).
- In Scikit-learn, the non-linear inputs are actually implemented by computing binomial terms with the columns. These columns are appended to the data frame, and fed into the Linear model as if they are linear features (the Linear Model doesn't know any better). It has the same effect, without needed a special non-linear class.

Logistic Regression:

- This is linear regression that is passed through a sigmoid function. This has the effect of restricting the output to a [0,1] interval.
- Use Cases: Classification between two sets. Implements a Linear Decision Boundary.
- Model:

$$y(x) = \sigma(wx + b) = \frac{1}{1 + exp(wx - b)}$$

  This is equivalent to a sigmoid neuron in a NN.
- Regularization and Feature Normalization is an option.

Decision Trees:

- Learns a series of IF THEN rules, in the form of a tree.
- Tree of Classifier Questions: The nodes encode a series of quesions about object features, which discriminate between the objects.
- Goal: Find a series of questions that (i) has the best accuracy and (ii) fewest number of steps/levels.
- Each node in our tree structure can have the following fields:
  1. Decision Rule
  2. Number of Samples (total of value list)
  3. Value List: Number of Samples per class
  4. Name of the Majority class.
- Node Terminology: - **Mixed Node:** One that consideres instances of more than one class
- Pure Node: Only considers instances on one class.
- Leaf nodes terminate, and do not have a decision rule.
- Adding Levels vs Overfitting: With DTs, we can keep adding nodes until all leafs are pure. However we might have so many leafs that we end up overfitting (almost every point is its own class). DTs are **prone to overfitting.**
- Overfitting can be stopped by a variety of options, including **pruning and limiting max depth.**
- Decision trees also have an importance score, which indicates which features contribute to classification, relative to others. This is called **Feature Importance**.

Linear Classifiers: SVMs:

- An SVM is a linear classifier, that is fed into a sign() function. Instead of the input being a number, it becomes a class.
- Model:

$$f(x, w, b) = sign(wx + b) = \{1, -1\}$$

  This implements binary classification. sign(wx+b $>$ 0) $\rightarrow$ 1 and sign(wx+b $<$ 0) $\rightarrow$ -1.
- Classifier Margin: the space between our line and the closest point.
- Objective to find "Best Classifier": The one that has the greatest margin is best.
- **Margin HyperParameter: C:**
  - Smaller Values of C: give better test performance, and tend to produce larger margins.
  - Larger Values of C: fit training data better, and focus on classifiying individual points more. Tends to produce smaller margins.
- **Pros and Cons:**
  + simple to train
  + predicts fast
  + good scaling relative to dataset size (memory)
  + works with sparse data
  + relatively easy to interpret results.
  - for lower dimensional data, other methods out perform (such as non-linear regression).
  - fails or poor performance for non-linearly separable data.

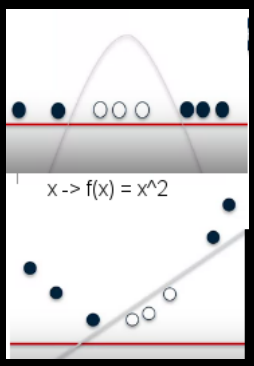
Multi-class Classification:

- For SVMS: we just make a series of binary classifiers. One for each class

Kernel SVMs:

- Kernel SVMS involve what is called the "Kernel Trick". This is used when the data is not linearly separable.

- If the data cannot be easily separated with hyperplanes, then we use a Kernel Function to shift the data in the following ways:
  1. Transformation of Coordinates
  2. Go up from dimension N to dimension N+K
  3. Transformation of Data itself.
- **Key Idea:** Data not linearly separable in a given space. So map to another space and check if its separable there.



- **Why it works:** Recall that in High-D spaces, data typically lives in corners. By transforming to an N+K Dim space, the data is likely to get cornered. We know that it takes massive amounts of uniform data to fill this large space - which the data set is unlikely to fulfill, so we have a good shot at cornered data, and separability.
- Types of Kernels:
  - **Radial Basis Kernel:**

$$K(x, x') = exp[-\gamma||x - x'||]$$

    Now $\gamma = \frac{1}{2\sigma}$ is a scaling parameter, and $||x - x'||$ is the distance between any two points.
    Note, that this function has a Gaussian form.
  - Behaviour of γ: this controls the effect that a point has on other points in its local area. Let $||x - x'|| = D$, Then:

$$\text{As } \gamma \to 0, exp[-\gamma D] \to 0 \text{ more slowly as D} \to \infty.$$

$$\text{As } \gamma \to \infty, exp[-\gamma D] \to 0 \text{ more quickly as D} \to \infty.$$

    So, large gamma → small decision region, small gamma → large decision region.
    The effect of scaling gamma can also be thought of as a "Normal" distribution about the point, with its variance scaling.
  - **Polynomial Kernel:** This kernel measures similarity of features. Less similar features "shoot away" from each other.

$$K(x, y) = (x^t y + C)^d$$

    Where C is a constant, and d is a power. It can be seen that small differences compound to large differences (relative to the original space), when d is a large number.
- **Hyperparameter Tuning:** C and γ : Recall that C controls the margin size of our lines. Using a Kernel with γ has a non-independent effect on C, so they are typically tuned together (grid search pattern).
- C only has a margin effect **when gamma is small.**
- SVMs in general are sensetive to multi-scale data. **You need to normalize!**
- **Pros and Cons:**
  + Performs well on a range of datasets
  + Versatile: different kernels can be used, for many different data types
  + Tricks work for Low and High Dim data
  - inefficient for large datasets
  - needs careful normalization of input data and parameter tuning.
  - does not provide direct probability estimates **(platt scaling?)**
  - Interpretations of why a prediciton made, difficult.

Cross Validation:

- Random Test-Training Splits can give suprisingly varied results. The wrong selection of train/test points can give unusually good/bad results. We wish to avoid this variance in the measurement of model performance.
- More Generally: models can be *inherently sensetive* to being trained differing subsets of the data
- Cross Validation is done to minimize this.
- **N-fold Validation:** Data is partitioned into N splits, each split being 1/Nth of the total data set. N models are trained. The test set varies as one of the N splits of the data, for every model trained. We calculate a performance score for each of the N models, and then compare them at the end.
- Leave One Out Validation: When one data point is used for the test set! useful for **small** data sets.
-

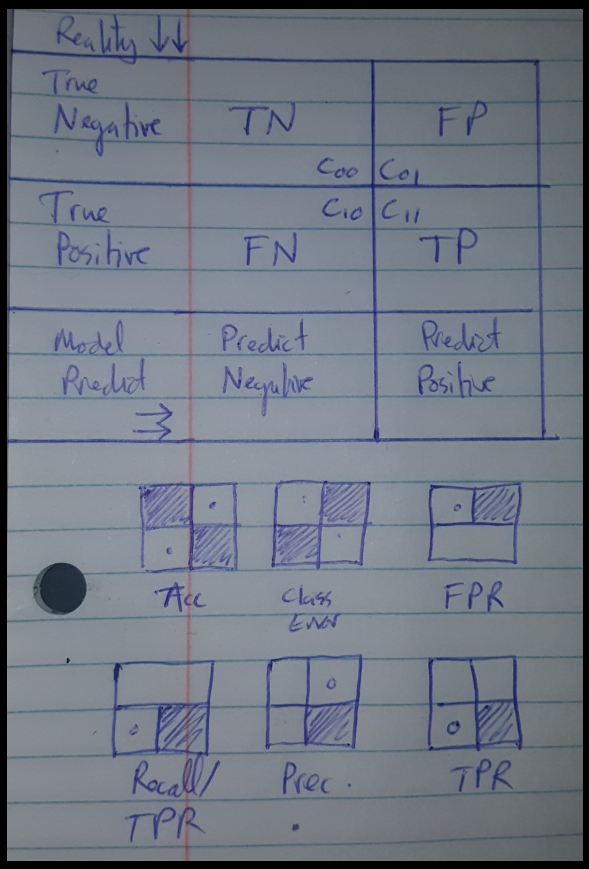
Week 3: Evaluation Methods:

Model Evaluation and Selection:

- Accuracy: Number of Correctly predicted targets / Number of Instances.
- Accuracy only gives us a partial picture. There are many different evaluation metrics we can use.
- For a classifier, it produces predictions that either match the class of a given instance (true), or not (false). This gives us *True Positive, True Negative, False Positive, False Negative* to consider.
- **Point of Reference:** For TP,FP,TN,FN, these terms are conditional on what ***the "positive" class*** is defined to be. This must be specified beforehand!
- $\{True/False\}\{Positive/Negative\}$: The first term identifes if our prediction on an input matches the input's class. The second term indicates what class we have actually predicted.
- Evaluation metric is often depenent on **practical application goals**: Survival rates, risk of unknowns, number of clicks, etc.
- **Against Accuracy: Imbalanced Classes:** Suppose you have 999 members in a negative class, and 1 member in a positive class. The information needed to explain this set is really low. To predict with 99.9 percent accuracy, just produce a function that pumps out "Negative Class" as an answer.
- Accuracy score needs to be measured relative to what a null/base classifier can do, and class distribution. Up until this point, seeing 90 percent has always been "good". For badly balanced classes, this is no longer true.
- **Dummy Baseline:** An algorithm that ignores that data, and returns a simple answer according to a rule.
- **Dummy Classifier:** A simple classifier baseline for evaluation. Return types include *most frequent, stratified (represents class distribution), uniform, or constant.*
- **Dummy Regressor:** A continuous function that returns a simple statistic, such as *mean, median or some constant*
- **Note:** RSS uses the mean dummy regressor to calculate the RRtot sum!
- Reasons for a null-level performance of classifier:
  1. Ineffective or erroneous features
  2. Poor kernel, or hyperparameter selection
  3. large class imbalances!

Confusion Matrix and Basic Evaluation Metrics:

- **Confusion Matrix:** A 2x2 contingency table, representing two conditions (predicted, and actual), across two classes (binary).



- The main diagonal of the CM indicates predictions that match reality. The off diagonals indicate mismatch (this is where the "confusion" stems).
- **Accuracy:** Number of inputs we classified correctly / all inputs considered

$$\frac{TP+TN}{TP+TN+FN+FP}$$

- **Classification Error:** Number of inputs we incorrectly classified / all inputs considered

$$1 - Acc \ \ or \ \ \frac{FP+FN}{TP+TN+FN+FP}$$

- **Recall:** Number of points we correctly predicted as positive / all positive points.

$$\frac{TP}{P} = \frac{TP}{TP+FN}$$

Other Names: True Positive Rate (TPR)
- **Precision:** Number of points we correctly predicted as positive / all points we prediced as positive.

$$\frac{TP}{TP+FP}$$

Other Names: Sensetivity
- **Specificity:** Number of points we correctly predicted negative / all negative points.

$$\frac{TN}{N} = \frac{TN}{TN+FN}$$

Other Names: True Negative Rate
- **F1 Score:** Harmonic mean of Precision and Recall.

$$\frac{2PR}{P+R} = \frac{2TP}{2TP+FN+FP}$$

- **Generalized F Score:** Beta can be tuned to give a biased mean between Precision and Recall:

$$F_\beta = (1+\beta^2)\frac{PR}{\beta^2(P+R)}$$

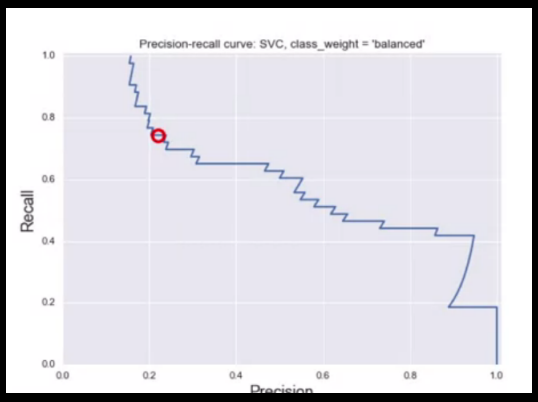
- Tasks can be optimized for particular kinds of errors, as they have real consequences in the world:
  1. **Recall Oriented Tasks:** Consequences of false negatives are high! Examples: Medical Diagnosis: Missing a tumour is more important that falsely identifying one (judgement: patient's life more important than comfort).
  2. **Precision Oriented Tasks:** The consequences of false negatives are high. Example: Customer Online Search: giving a customer a result that isn't relevent is worse than not giving them something relevent (judgement: stopping customer dissapointment is more important than giving them choices).
- **Fine Tuning the Confusion Matrix and Error Metrics:**
- This is done by shifting decision boundaries for classifiers. This will cause the counts in metrics and the Confusion Matrix to shift. You can shift decision boundaries to optimize for the metrics that you care about.
- Precision-Recall Tradeoff: An effect commonly seen when boundaries are tweeked. This **does not mean you cannot have very high P and R together**, they are just often coupled negatively.

Classifier Decision Functions:

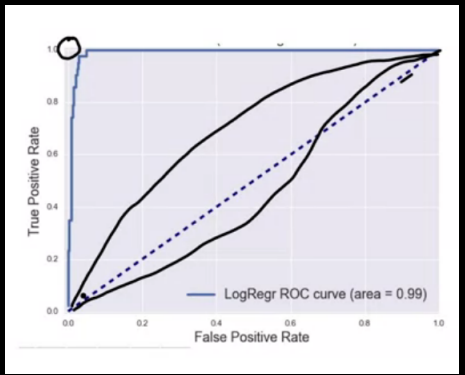
- For many classifiers (such as LR, SVC, etc), you can get access to intermediate classifier outputs. Depending on the type of model, there are two functions that are provided:
  - Decision Function: This gives access to the **classifier hyperplane** (Example: SVC).
  - Predict Proba: For classifiers with probabilistic results, this gives the probability that points are in a base class or not (example: Logistic Regression).
  - We can use the decision functions to shift boundaries, and plot Metric Curves, such as Precision-Recall and ROC.
- **Important Note:** There is no way to bias or shift a classifiers output directly (such as setting a property or calling a function. In Scikit-learn, you have to get one of the two decision functions, and manually apply thresholding.
- Both Decision functions assume a **baseline/bias of zero when operating**.

Precision Recall (PR), and Receiver Operating Characteristic (ROC) Curves:

- Precision Recall Curve: A plot of Precision and Recall Metrics in a 2D diagram. The curve is generated by taking a decision function and varying its thresholding. For every threshold, a pair of (P,R) points is generated. You typically see large jumps because small changes to the decision boundary can affect the TP,TN,FP,FN counts dramatically.



- **Ideal point:** top right corner of the diagram.
- Reciever Operating Characteristic: This plots the TPR and FPR on a 2D diagram. The dotted line represents a classifier that just produces random output (as a baseline).

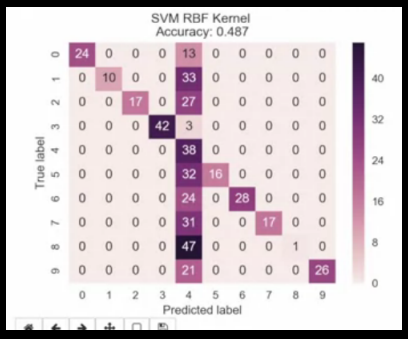


- **Ideal point:** top left corner of the diagram.

- **Area Under Curve:** A metric for evaluating model performance in an ROC diagram. The more area under a particular curve, the better the classifier. For the baseline, the area is 0.5. The limit is 1.

Multi-Class Evaluation:

- here, we have a set of binary outcomes, across more than one class.
- evaluation metrics are no longer marginal fractions, they are averages across classes.
- support (number of cases per class) becomes more of an issue.
- Example: MNIST forms a 10x10 matrix. Each class represents a number (or not).



- We can apply heat maps to our bins. This allows us to pick out problems more easily. We can see above that the digit 4 has serious issues in classification.
- Macro Averaging: we calculate a given metric per group, and then calculate a final equal weighted average across all the groups.
- Micro Averaging: We don't take groups, we calculate the metric across all the classes.
- **When to use Micro vs Macro Averaging:**
  - if the class distributions are roughly equal, micro and macro averaging should be around the same
  - Claim: microA << macroA: then large classes may have poor metric performance.
  - Claim: macroA << microA: then small classes may have poor metric performance.

Regression Evaluaiton Metrics

- With regression, we output a contiuous value, not a discrete label. Differences between our output and target on the $\mathbb{R}$ are what matter here.
- The standard metric used to evaluate Regression is R^2. There are others:
  1. Mean Absolute Error: L1 norm. What else?
  $$\sum |f(x) - \hat{y}|$$
  2. Median Absolute Error: Robust to Outliers.
  $$\sum (f(x) - median(y))^2$$
  3. Mean Squared Error: Will blow up with outliers.
  $$\sum (f(x) - \hat{y})^2$$
- Dummy Regressor: This is used to output a summary statistic, like median, mode, constant, etc.

Model Selection: Optimizing Classifiers for Different Evaluation Metrics:

- The different metrics learned are used to select models.
- Depending on the metric you choose, the "best" set of hyperparameters will change.
- GridSearchCV() class will search an array of values, for a given metric for you. It encapsulates the model, you can invoke the model through the GridSearchCV object.
- 4 Types of Training with Data:
  1. **Test/Train on the Same Data:** - assume it will overfit.
  + useful for a baseline sanity check: low metric on train set indicates implementation problem.
  2. **Single Test/Train Split:**
  - only one metric.
  - no sub-sample variance information: depending on your split, metric will change.
  3. K-fold Cross Validation:
  + more resistance to metric variation, and overfitting (does not guarentee you will not overfit!).
  - more computationally intensive ~ Use in conjunction with GridSearchCV.
  4. Training, Validation and Test: Here, we have two stages of hold out set: validation and test. Do Test/Validation split to train a series of models. Select From the series of models using the final test set in the end.
  - still does not guarentee no overfitting: very subtle forms can occur.

Week 4:Supervised Learning: Part 2:

Naive Bayes Classifiers:

- A simple probabilistic classifier. Input features, output probability of a given class. In general, given a point $\vec{x}$, we are trying out output:
$$P(C_k|\vec{x})$$
- **Features:**
~ assumes features are conditionally independent. The full joint probability for x and all classes C is typically very complicated. NB makes independence assupmtion to greatly simplify the joint probability, and its calculations.
- conditional independence practically, all features of an instance are independent of the others, given the class. This means:
$$P(C, x_1, ...x_n) = \Pi P(C, x_1)...P(C, x_1)$$
+ because of simplifying assumption, very simple and fast.
~ Can be used as a non-trivial baseline for more complicated models.
- confidence and probability estimates for NB classifiers are unreliable, typically **(??)**
- Ideal Case: High dimensional dataset, where features are mostly independent. NB is competitive with other methods.
- Worst Case: Smaller Datasets with highly correlated features. NB performs very poorly.
-
- **Types of Naive Bayes:**
  1. Binomial: structure?
  2. Multinomial: structure
  3. Gaussian: Assumes that data is generated from a Gaussian Distribution with (ν,σ). There is a gaussian distribution that sits atop each class area in space. Points with the lowest SD score away from the mean of a Gaussian, are classified accordingly.
- Bin and Multi are often used for text classification.
- Claim: If Variances differ between classes: the decision boundary is parabolic.
- Claim: If variances between classes are identical, the decision boundary is linear.
- **Claim about Reality (?):** For very high dimensional datasets, lots of features have to be independent. If everything was correlated, and there was a lot of causal links, there would be insane complexity and feedback.

Random Forests:

- An ensemble of decision trees.
- DTs typically overfit different sections of the data.
- We can combine and evolve trees in such as a way as to get better test/train performance, compared to individual models.
- Can be used for classification, or regression.
- Methods for Evolving Trees:
  1. Bootstrap sampling: sample n of N training rows, with replacement, when training the tree.
  2. Subset of Features: pick a subset (not all) features, and just split trees on this subset.
- **Prediction Algorithm for Random Forests:**
- First, you get a prediction for every tree. Then:
- For Regression: take the mean of all predictions.

- For Classification: each tree gives probability for each class. Predictions averaged across class probabilities. Choose the class with highest average probability.
- **Pros and Cons:**
  + excellent performance on many problems.
  + doesn't need feature normalization, scale invariant.
  + easily parallizable.
  - Difficult to interpret results.
  - computationally intensive for large datasets.

Gradient Boosted Decision Trees:

- Similar to Random Forests; an ensemble of trees that can be used for Regression or Classification.
- **Growth Rule:** Each successive tree tries to correct for the mistakes of the previous trees.
- Typically, GBDT uses lots of shallow trees ("weak learners"), makes fewer mistakes as trees are successively added.
- Parameters to tune:
  - Learning Rate: This affects how strongly our newest tree tries to correct for the mistakes of the previous tree. Parameter size is directly proportional to strength.
  - n estimators: how many trees we wish to grow.
- **Pros and Cons:**
  + some of the best off-the-shelf accuracy
  + prediction from the model requires modest CPU and memory usage.
  + no need to normalize features; scale invariant.
  + can deal with any type of feature, or a mixture, quite well.
  - difficult to interpret models
  - sensetive to parameter settings; these need to be tuned carefully.
  - potentially computationally intensive to train.
  - performs very poorly with high dimensional, sparse data. Or with textural analysis tasks.

Neural Networks:

- This module focuses on Multilayer Perceptrons.
- Neural Network consist of layers of connected neuron units - which are activation functions that take hyperplane inputs. The hyperplanes consist of a weight - input dot product, with a bias term.
- For every layer, there is an input, a weight matrix, a set of activation functions that accept a weighted input. The activation units themselves will pump out a layer of outputs.
- NN can be configured to output for Classification, Regression or as intermediate results for other nets.
- Activation Function Types: Sigmoid, Linear Rectifier, Tanh, sign, etc.
- Important Parameters:
  1. hidden layer sizes, alpha, activation.
  2. alpha: Regularization Parameter, just like with other sklearn models.
  3. activation function: choose from one of above.
- NN are extremely powerful models with the potential of many layers, weights and the ability to capture highly non-linear and complex behaviour. This makes them harder to train and design, however.
- NN are sensetive to large weight overfitting, and multi-scaled featured. Normalization and Regularization are often needed.
- **Pros and Cons:**
  + Extremely powerful, exceeds the potential of all other models seen.
  - large complexity, hard to design and tune.
  - needs significant computer power to find a proper model.
  - sensetive to features, and has trouble with mixed type features.

Data Leakage

- This is a subtle and multi-faceted problem - one of the most serious in data science.
- Definition: When the data you use to train contains information about what you are trying to predict.
- **Most Basic Example:** Training an SVC with the yTarget as a column in the xTrain training set. The classifier would just zero out all weights except for the yTarget, and get perfect accuracy. But for prediction for future data points, we won't have yTarget. This is useless, and cheating.
- More generally, we can introduce features that "give-away" information about the target. This results in near perfect accuracy, and bad test performance.
- *More generally, any feature that is highly predictive of the target, but not available during the prediction phase is a give-away, or has been leaked.*
- **Categories of Data Leakage:**
  - **Leakage in Training Data:**
  - Performing preprocessing using parameters or results, derived from analysing the entire dataset.
  - Time Series Datasets: Using data from the future, to help predict future events.
  - Special information in Marker or NA data: this could give the state of a system.
  - **Leakage in Features:**
  - Removing illegitimate features, without removing other features that entail informaiton about the target (example: remove patient surgery for medical condition, but patient ID indicates they are a special type of patient).
- **Note:** External Data Joins are a major source of Data Leakage!
- **Detecting Data Leakage:**
  - **Before Building the Model:**
  - Exploratory Data Analysis for data anomolies or suprises.
  - Are any features very highly correlated with the targets?
  - **After Training the Model:**
  - Surprising behaviour of the fitted model.
  - Results too good to be true (compared to other model baselines).
  - Are there features with very high weights, or information gain?
  - **During Deployment:**
  - very poor generalization, test results.
- **Minimizing Data Leakage:**
  - **Perform Data X within each cross fold separately, not for the entire dataset before.**
  - Scaling and normalization must be done inside the fold.
  - For any parameters calculated when processing the data, they must be used for the test fold as well.
  - **For Time Series Data, give a time cutoff.**
  - **Before any work with a dataset, split of a final validation set.**
  - ...you may not be able to do this (not enough data).
  - use as the final check after the final model has been selected from the evaluation process.
  - confidence and probability estimates for NB classifiers are unreliable, typically **(??)**

Residual Questions:

- Why does 1-Norm Lasso cause small weights to go to zero, in gradient descent?
- Why does regularization of weights become less important, as your number of data points increase?
- For Multiclass classification, what happens when we predict positive for more than one class? How do we choose uniquely?
- What is influence in the KBF? what is the object being influenced?
- Why does scaling on the entire dataset before hand cause leakage (test leaks into train)?
- `http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html` to avoid data leakage.
- Data Visualizaitons Playground
- Understanding the baseline curve of a AUC plot
- Derivations: Decision Trees, Random Forests,

References

[1] Geron, Aurenlin, Hands on Machine Learning with Scikit-learn and Tensorflow, Chapter 1, O'Reilly Press.

[2] Applied Machine Learning with Python, University of Michigan - Coursera Course.