**Lecture 3 - NumPy**

Import numpy as np, data = np.array(…), all elements must be of the same type – data.shape, data.dtype, data.dim, np.zeros, np.ones, arrange is like range but returns a numpy array, eye / identity both return an identity array. Binary operators between two arrays of the same shape are done element by element, An array and scalar applies the scalar to every element. Comparing two same shape arrays creates a Boolean array. The .copy() makes a copy of the array. Array indexes are views. Arr[x , y] – x and y are applying to different dimensions. Arr[3, :2] says to take the third element in the first dimension and then take 0-2 values in the second dimension. If Names has a list of names that is the same number as the first dimension of data, can say data[name == “Bob”] to get all of Bob. Name == “Bob” creates a binary mask. This always creates a new array.

Transpose function, arr.T is the transpose of arr. Np.where(cond, Xarr, Yarr) – if true, from Xarr, else Yarr. Sum, mean, std, etc.

**Lecture 4 – Pandas**

Pd.series ([1,2,3,4]) – series with numeric 0 – n-1 as index. Pd.series([1,2,3,4], index=[“a”,”b”,”c”,”d”]) series with character index. Similar to a dictionary, a = pd.Series(aDict) will convert the dictionary with the key as index.

Dataframe has both a row and column index. A dictionary of equal length lists can be used to create a dataframe. Can use Columns parameter in pd.dataframe. Assigning or mentioning a column that does not exist will create the column with NaN. New columns cannot be create with the dot notation. A.newcol = 7 will not work but a[“newcol] = 7 will. Del df[“acol” will delete acol. Indexing is a view on the underlying dataframe. Use copy to create a new one. Can set name of index and columns using df.index.name = “row name” and df.columns.name = “col name”. Indexes are immutable. Can use reindex to reorder the index values and add new ones. Method can be used with reindex to fill in values – forward, backward, etc. Can use reindex on columns with the columns keyword. Drop can remove rows or columns. Use index for rows and Columns for columns. Loc and iloc are preferred. Loc is label based and iLoc is interger based. Pandas slices include the last element. Df[“a”] referes to a column named a in the dataframe df. Oh, and if you want two columns, you need two braces. So, df[[“a”,”b”]].

Df[df[“b”] > 5] gets all rows in df where column b is greater than 5. Df.loc[“Colorado”] gets the row with index label of Colorado. To get multiples, we need the double brackets. Df.loc[[“Colorado”, “Ohio”]]. Df.loc[ row, col] so, we can have df.loc[“Ohio”, “a”] to get column a for row Ohio. If sets, use the doulbel brackets. Df.loc[[“Ohio”,”New York”],[“a”,”b”]] gets columns a and b for rows Ohio and New York. Same for iLoc but you need to use integer references. Df.sort\_index() sorts rows by index. Df.sort\_index(axis=”columns”) sorts columns. Add , ascending= False to sort Descending. Use sort\_values to sort by values. .describe() provides summary statistics.

**Lecture 6 – What is Machine Learning?**

*Supervised learning\** involves somehow modeling the relationship between measured features of data and some label associated with the data; once this model is determined, it can be used to apply labels to new, unknown data.

This is further subdivided into *\*classification\** tasks and *\*regression\** tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities.

*\*Unsupervised learning\** involves modeling the features of a dataset without reference to any label, and is often described as "letting the dataset speak for itself."

These models include tasks such as *\*clustering\** and *\*dimensionality reduction.\**

Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data.

Scikit-Learn, Seaborn

import seaborn as sns; sns.set() / sns.pairplot(iris, hue='species', height=1.5); 🡪 Generates plots for each combination of features

Scikit Learn principles - - *\*Consistency\**: *\*Inspection\**: *\*Limited object hierarchy\**: *\*Composition\* \*Sensible defaults\**

Chouse model, set hyperparametrs, arrange data, fit(), for supervised, predict(), for unsupervised transform() then predict()

from sklearn.linear\_model import LinearRegression / model = LinearRegression(fit\_intercept=True) / X = x[:, np.newaxis] / model.fit(X, y) / Xfit = xfit[:, np.newaxis] / yfit = model.predict(Xfit)

from sklearn.model\_selection import train\_test\_split / Xtrain, Xtest, ytrain, ytest = train\_test\_split(X\_iris, y\_iris, random\_state = 142)

predict on training set, test on test set

*if our estimator is underperforming, how should we move forward? /* Use a more complicated/more flexible model / Use a less complicated/less flexible model Gather more training samples / Gather more data to add features to each sample

High Bias 🡪 underfit the data, High Variance 🡪 Overfit the data

For some intermediate value, the validation curve has a maximum. This level of complexity indicates a suitable trade-off between bias and variance.

Validation curve is about model complexity; learning curve is about size of training data

**Lecture 7 – Feature Engineering**

One-Hot Encoding – a 1/0 column for each possible value

Improve a model not by changing the model, transform the inputs

from sklearn.impute import SimpleImputer / imp = SimpleImputer(strategy='mean')

Why do you use a pipeline? Allows you to combine multiple steps.

Gaisian Naïve Bayes requires no covariance between dimensions, is generally quadratic

Multinomial Naïve Bayes – from a multinomial distribution

Good: They are extremely fast for both training and prediction

They provide straightforward probabilistic prediction / They are often very easily interpretable / They have very few (if any) tunable parameters

Useful: When the naive assumptions actually match the data (very rare in practice) / For very well-separated categories, when model complexity is less important / For very high-dimensional data, when model complexity is less important

Convert input to polynomial or gaussian – replicates x’s

There is a question on Regularization --> review it, what do Ridge and Lasso do and what are the parameters – alpha

Ridge regression, L2 (Tikhonov) regularization – penalizes sum of squares (alpha 🡪 0 standard linear, to infinity all suppressed), Lasso, L1 () regularization – penalizes sum of absolute values, favors sparse models

Support Vector Machines – Classification and regression, maximize the margin, support vectors- on the margin, insensitive to distant points, kernels 🡪 all combinations but Kernal Trick means we do not have to. RBF in Scikit Learn, Parameter C tuns the margins – low C, soft, some points in the margin, large C hard margin

Good: Their dependence on relatively few support vectors means that they are very compact models, and take up very little memory. / Once the model is trained, the prediction phase is very fast. / Because they are affected only by points near the margin, they work well with high-dimensional data—even data with more dimensions than samples, which is a challenging regime for other algorithms. / Their integration with kernel methods makes them very versatile, able to adapt to many types of data.

Disadvantages: The scaling with the number of samples $N$ is $\mathcal{O}[N^3]$ at worst, or $\mathcal{O}[N^2]$ for efficient implementations. For large numbers of training samples, this computational cost can be prohibitive. / The results are strongly dependent on a suitable choice for the softening parameter $C$. This must be carefully chosen via cross-validation, which can be expensive as datasets grow in size. / The results do not have a direct probabilistic interpretation. This can be estimated via an internal cross-validation (see the ``probability`` parameter of ``SVC``), but this extra estimation is costly.

Random Forests, decisions trees are good but can overfit, ensemble method called bagging 🡪 Random Forests, need to specify number of trees, Random Forest Regression

**Lecture 8 – PCA and Clustering - Unsupervised**

PCA – Principle Component Analysis-unsupervised dimensionality reduction, the length of the vector is an indication of how "important" that axis is in describing the distribution, affine transformation, the least important principal axis or axes is removed, leaving only the component(s) of the data with the highest variance. Basis function converts the high dimension data to lower dimension, PCA is the coefficients of this basis function, select number of components with cumulative *\*explained variance ratio\** as a function of the number of components- look for the first N components, PCA can filter out noise

K-Means – Clustering, The "cluster center" is the arithmetic mean of all the points belonging to the cluster. / Each point is closer to its own cluster center than to other cluster centers. Expectation–Maximization

1. Guess some cluster centers

2. Repeat until converged

1. *\*E-Step\**: assign points to the nearest cluster center

2. *\*M-Step\**: set the cluster centers to the mean

May not produce a globally optimal solution, parameters include number of times to run starting guess and number of clusters, Silhouette Analysis to find number of clusters, spectral clustering for looking for non linear boundaries, k means can be slow for large samples

**Lecture 9 – Time Series analysis in Python**

*Time series\** is a series of data points indexed (or listed or graphed) in time order.

MAPE – Mean Absolute Percentage Error, moving average, smooths the data, can use different window sizes, can weight recent observations higher,

Exponential smoothing, Exponentially reduce weight for older values, Double Exponential Smoothing –

Triple Exponential Smoothing – Holt Winters – seasonality

Time Series Cross Validation needs to keep time periods separate, so cannot be random

The idea is rather simple -- we train our model on a small segment of the time series from the beginning until some $t$, make predictions for the next $t+n$ steps, and calculate an error. Then, we expand our training sample to $t+n$ value, make predictions from $t+n$ until $t+2\*n$, and continue moving our test segment of the time series until we hit the last available observation. As a result, we have as many folds as $n$ will fit between the initial training sample and the last observation.

truncated Newton conjugate gradient. – parameters can range from 0 to 1

Stationariyt 🡪 mean and variance do not change

This is the main idea behind the [Dickey-Fuller test](https://en.wikipedia.org/wiki/Dickey%E2%80%93Fuller\_test) for stationarity of time series (testing the presence of a unit root). If we can get a stationary series from a non-stationary series using the first difference, we call those series integrated of order 1. The null hypothesis of the test is that the time series is non-stationary, which was rejected on the first three plots and finally accepted on the last one. We have to say that the first difference is not always enough to get a stationary series as the process might be integrated of order d, d > 1 (and have multiple unit roots). In such cases, the augmented Dickey-Fuller test is used, which checks multiple lags at once.

We can fight non-stationarity using different approaches: various order differences, trend and seasonality removal, smoothing, and transformations like Box-Cox or logarithmic.

Shifting the series $n$ steps back, we get a feature column where the current value of time series is aligned with its value at time $t-n$. If we make a 1 lag shift and train a model on that feature, the model will be able to forecast 1 step ahead from having observed the current state of the series. Increasing the lag, say, up to 6, will allow the model to make predictions 6 steps ahead; however it will use data observed 6 steps back. If something fundamentally changes the series during that unobserved period, the model will not catch these changes and will return forecasts with a large error. Therefore, during the initial lag selection, one has to find a balance between the optimal prediction quality and the length of the forecasting horizon.

As we already know, not all features are equally healthy -- some may lead to overfitting while others should be removed. Besides manual inspection, we can apply regularization. Two of the most popular regression models with regularization are Ridge and Lasso regressions. They both add some more constrains to our loss function.

In the case of Ridge regression, those constraints are the sum of squares of the coefficients multiplied by the regularization coefficient. The bigger the coefficient a feature has, the bigger our loss will be. Hence, we will try to optimize the model while keeping the coefficients fairly low.

As a result of this $L2$ regularization, we will have higher bias and lower variance, so the model will generalize better (at least that's what we hope will happen).

The second regression model, Lasso regression, adds to the loss function, not squares, but absolute values of the coefficients. As a result, during the optimization process, coefficients of unimportant features may become zeroes, which allows for automated feature selection. This regularization type is called $L1$.

But, this victory is decieving, and it might not be the brightest idea to fit `xgboost` as soon as you get your hands on time series data. Generally, tree-based models handle trends in data poorly when compared with linear models. In that case, you would have to detrend your series first or use some tricks to make the magic happen. Ideally, you can make the series stationary and then use XGBoost. For example, you can forecast trend separately with a linear model and then add predictions from `xgboost` to get a final forecast.

Prophet – Facebook, addresses Seasonality, Trends, Outliers, scales to lots of analysts, lots of problems, automation – y(t) = g(t) + s(t) + h(t) + epsilon. G – is trend, s is seasonality, and h is holiday