EDA — Python

When to use GridSearchCV vs RandomSearchCV?
Latter is mostly better cuz faster since checks on a finite space of hyperparam.

The only way to find the best possible hyperparameters for your dataset is by trial and error, which is the main concept behind **hyperparameter optimization**.

Now, let's define the hyperparameter space to implement random search. This parameter space can have a bigger range of values than the one we built for grid search, since random search does not try out every single combination of hyperparameters.

Run the following lines of code to run random search on the model: (Note that we have specified n_iter=500, which means that the random search will run 500 times before choosing the best model.

Since each DT takes a different set of training data as input, the deviations in the original training dataset do not impact the final result obtained from the aggregation of DTs. Therefore, bagging as a concept reduces variance without changing the bias of the complete ensemble.

Removing features having a correlation(pearson/spearmann) = 0.6 would probably be unnecessary. Consider to select features with 'variable importance' instead.
@*which models are more sensitive to correlated features?* - Classifiers/ regressors with low or no regularization e.g. LDA or MLR.

duplicateRows = df[df.duplicated(['team', 'points'])]

#view duplicate rows
print(duplicateRows)

EDA: Generally, how do u drop a col from a panda df?
Remove two columns 'C' and 'D'
df.drop(['C', 'D'], axis=1)

df.drop(df.iloc[:, 1:3], inplace=True, axis=1)
What does the axis arg do?
Specifies u're ref to columns (axis=0 is X or rows)
Remove all columns between column index 1 to 3

How to create an attribute of list of flavors and then print ∀ flavors of an instance (of ice_Cream_stand)?
Under __init__:
  self.flavors = []
  def show_flavors(self):
      print("....")

instance_name.flavors = ['Choco', 'Vanilla', 'strawberry']
instance_name.show_flavors()

How to import a class?
From mod_name import Class_name as CN.

model=GaussianNB() this creates a model from this class. You can now apply functions on this.
To see if what ur importing is a class or a function, use help function.
model.fit(X_train, y_train) will train the model on these data.

Can also use drop function. **Learn**

Py: With X and y matrices, how do we form a df with both these?
Use Concat function.
joint_df = pd.concat([X,y],axis=1)

joint_df = pd.concat([X,y],axis=1)
What does axis = 1 mean?
Axis = 1 means has columns.
Axis 0 as rows.

With scatterplot of y and x axis of both our X features we can see how they are scattered and which species they point to.

Sklearn is a big boi package which has a library called model_selection. You import function traint_test_split.

tts(X, y, test_size=0.2, random_state=9)

Random_State is a seed value.
If u sow the same seed u get the same flower, the same random set. It is for reproducibility.

Sklearn also has a lib called

There's conditional prob In this cuz it is about if you have these 4 lengths then what's the probability that this belongs in

Before creating the object, the para u will pass will be hyperpara.

All classifiers have at least the 2 methods of fit and predict.

y_test vs y_test.values have a display diff, the latter shows in form of array and former in just 1 col.

model.fit(X_train, y_train)
Train model.
The .fit shows that this is a function that is being called.

sklearn.metrics lib measures the dist bw pred value and actual value. Has many functions in it.

Accuracy_score is one such function in it. U pass pred and then actual values to it of y.

Now with the model trained and evaluated, you can pass two random points or values of petals to see what flower it can be.

He is always available in office

Testing 1st point, p1. Z = 0 + (1.0)(1) + 0.8(2) = -0.6, so falls in otherwise category in the Φ(z) s.t. Φ(-0.6) = 0. Wrongly classified.

P1 is first vector of train data.

In iteration you will update weights by which row you are working in.

Class Perceptron:

    def __init__(hyper paras):

    def fit(trains alg on data):

    def predict:

    def decision region

Perceptron is a binary classifier

We will discuss the math behind the alg but not going to be tested in exam.

Exam will be all about application, not about math. On data available on Sklearn.

Google CoLab can return codes from Python input.

Jupyter: The eta = 1, this is the learning rate. It is also the rotation factor. Same eta.

Size = 1+X.shape[1] here +1 is to account for bias. shape[1] is the column since .shape returns rows x col.
Y comes in two dim. All df do. .flatten() turns to 1D.
Y_pred has no i cuz we are looking at current row.
While loop has if error == true then it repeats.
Error = any(Y-y_true) evaluates to True if there's a difference. That's what any function does.

Class can create any object, say model.
And then model.function for using any function.
We used .w as an attribute cuz we wanted to check weights later by doing model.w.
\pd.option.display.max_rows sets rows how many u want, not the default 10.

Classifier works only for 0 and 1, so code your categorical values. Use replace function.
When one iteration of run is problematic, then reset your code. At start, use %reset. Previous values will be erased.

Default perceptron iterations are 1000.
U have to allow some error in your code. U can't say stop iff error = 0. Allow 70%. This isn't a benchmark tho. It can be optimized along the way. U can set it too if you want.
When u don't accept any kind of error, u are overfitting the data.

Sklearn.datasets is a lib w many datasets.

U start w a guess of x0 and cal the slope there. We know if slope is dec we're going towards the minima. These values are called successive approximation. Eta here is the step towards the minima. U keep finding the slope at each x point, and to get the next x point, closer to optima/minima, u just add the slope (since it is negative): x1 = x0 + (-slope).

If slope is positive, x1 = x0 - slope
But if we take a full slope step towards the minima, we may overshoot. So we take a fraction of a step, that's our parameter now, eta, η
And since our slope isn't 1D but 2D, since two var involved, we have a gradient instead of slope, ∇E.
X = X - η∇E

∇E = [∂E/∂w0 ∂E/∂w1] = gradient

Another optimizer is Stochastic descent. It is also called solver. This is a hyperparameter in the alg
Steepest descent isn't an alg, it is an optimizer. Also called batch descent.

Logistic Regression

We want to get the correct target now for each row given features and values. Same as perceptron, but we find the probability of the labels. We'll predict both class (like Perceptron) but also the probability.

Same threshold function we have, but there is an intermediate function, a probability function, that assigns what's the probability of the z.
Basically, you have any z value but this function maps the z value from 0 to 1, a probability.
It is Φ(z) = 1 / (1+e^-z). ∃ called Sigmoid Function.
Chain: Net Input function (z) —> Probability function: Φ(z) —> Threshold function (and you can set your thresholds)

Logistic regression has Y either 1 or -1. Nothing in bw. No 0 either.
Finally, y^ = {1 if Φ(z) > 0.6, -1 o.w.}, the threshold function.

We will not use probability but rather odds. However how do we connect z to Φ(z)? The pr can be only bw 0 and 1. But z can be more than that. So we use odds to let values go beyond 1. Odds = P/1-P
But again, what about negative z values? Odds can't be negative. So we apply log. log(p/1-p) = z. Now we have the logistic function. Specifically, it is called logit function.

ln(p/1-p) = z. How do we get this in the form of Φ(z) = p?
P/1-p = e^z. —> e^z - pe^z = p —> p = e^z / (1+e^z) = 1 / (1+e^-x)
BTW, p = Pr(Y=1 | wn, xn)

Sklearn.linear_model.LinearRegression
No eta parameter here cuz using old method, fixed. Gives exact weights or values. Eta or learning rate comes iff ∃ iter in alg. Otherwise only 1 round and no learning needed.

Inheritance:

Model = LinearReg() creating object
Model.coefficients gives you all the weights
This class, LR, has all these functions fit and predict.
Anything starting w small is function but w capital is a class.

LR class has default att and values. U don't need to pass anything in init. But when you are fine tuning it you have to change stuff.

You can have regression
modle.fit(X, y) doesn't return anything cuz it isn't a value. It is an object that is being stored. When you return an object you can perform other things on the class.
U can't apply other functions on values. So it doesn't return values. Just like in R.
U can also apply 2 functions in same line, you can't do that with values.

You can return numbers with just self. Self is an objectt

When you return self it doesn't return any value but rather an object that can't be displayed on the terminal. It will just give you gibberish. But you can apply two functions on it at the same time.
If you return a var or value, a, then you can't apply anything else on it.

If you return self, then
You can then use self.attribute to get the value at end.

In all parameters that have an underscore, it means they were trained in the alg. Not hyperparameters but parameters.
Model = LinearReg(eta=9)
model.eta to get the default value of eta.
Model.fit() is an object
To get the values or ßs of the model, model.coefficients_
We normally return objects from class, not values.

Logistic Reg

X1	X2	Y

Every row is a sample

Combining them gives you P(Y | x''w) = Φ(z)^y (1-Φ(z))^(1-y)
So when y=1, we have 1st and vice versa.

This is our Likelihood function basically.
We gotta max this.
L(**w) =** Product sum Φ(zi)^y (1-Φ(zi))^(1-y)
Take log to bring down exponents and turn product sum to ∑.
log[L(**w**)] = ∑ [yi • logΦ(zi) (1-Φ(zi)) • log(1-y)]
This is called the log loss function.
The bold rep vector of weights

To min this function, we use the gradient descent method. We need to choose such **w** or weights that z is min so that log loss is min.

Steepest descent isn't a model, just a solver.
Data that has one or two points mixed or across the DB is still called linearly separable data.
W new = W old - eta•gradient F(w)

For this alg, you need data that is perfectly linearly separable.

EDA: What's our λ in Logistic Reg for classification?
penalty parameter with the weight constraint: min 1/2•||w||^2.

EDA: What's each term in f(W) = F(w) - λf(w)?
F(w) is the old loss log function that must be min
The capital W rep the new function of loss with penalty function, f(w)

However, this alg is gonna try to overfit. It will keep

The constraint is gonna be modulus or mag of the weights. Keep the weights low to avoid overfitting.
||w|| = (∑w^2)^1/2
Math in simplicity uses 1/2||w||^2.
We gotta min this. This is our constraint.
But with that we have a penalty parameter, lambda Λ.
We gotta set this from the start,
f(W) = F(w) - σf(w). The capital W rep the new function of loss with penalty function. This constraint was imposed to account for the outlier.

Transformers and Pipelines

Just like estimators have two methods, fit and predict.
Fitting for scalar methods is calculating the min and max, it will calc these two (if min/max scalar) and cal mean and s.d. (if standard scalar). Fit cal the parameters req to transform, just like fit in estimators predict parameters to use for reg. Transform method then actually does the work with the parameters found in fit.

Another scalar called Robust scalar.

Le = LabelEncoder()
Is creating an object based on label encoder class (preprocessing)
Y = le.fit_trasnform(y) performs both fit and transform methods.
le.classes_ calls the classes or parameters in the model. The _ signals it isn't a hyperpara.

Under Trasnformers, every class has both a fit and transform method.

le.transform(['M', 'B'])

Stratification or stratify ensures the Y dist in both test and train sets are same.
IT BASICALLY RANDOMIZES or selects unbiased sample from y df.

Cross Validation

‘Lbfgs’ is the Quasi Newton’s method.
Newton’s Method:

To recall which fold got the best score, just set k to 7 and assign the train and test to this sample’s.

Learn how to call num in strings, pipe method and whatnot.

How to make a grid in bumpy?

For anything related to splitting, just choose model_sleection lib.

U can just cross valid w its own class in a one-liner.
Can pass linear reg in estimator or the whole pipeline.

Support Vector Machines

Dis = ax1+bx2+c / a^2+b^2
= w′X / ||w||
This norm we used is called L2 norm.
Max norm is L1? Idk. **GOOGLE.**
These norms are basically penalties hence hyperpara.

The idea is to maximize the two hyperplanes. The distance bw the data points.

Dis = ax1+bx2+c / a^2+b^2 = w′X / ||w||
How does this relate to SVM?
Find a w s.t. the distance is max.

EDA: Can combine two constraints into one constraint: $y(w'x+b) \geq 1$

The constraints are that the hyperplane should be def s.t. $w \cdot xi+b \geq +1$ when yi = +1
$w \cdot x1+b \leq -1$ when yo = -1.

U MAX total d bw H1 and H2: $2/||w||$
To max this, u can min $1/2 \cdot ||w||^2$.

Lagrange multiplier is what we'll use to min this, for this optimization problem.

Use the sklearn.svm lib to use linearSVC (for linear only), svc (for both cases) classes.

sklearn.inspection lib import **DecisionBoundaryDisplay** class to make DCs for both linear and nonlinear, but features must be 2 only.

ANKI:

Flatten the tuple or array to make it 1dim array, 1 row.

Zip basically makes a cross product or cross tuple of matching stuff, here models. Titles, and graphs.

Models = clf.fit(X,y) for clf in models

For clf,title, ax in zip(models, titles, sub.flatten()):
So clf matches w models, title with titles, and axis with subplots (rmb that sub has axis)

Bw min max and standard i.e. normalize vs standardize. WHEN to do which?

Sms.pairplots is a function to obs your data. If itslooks normal then standardize it. But if it doesn't look normal then do normalization.

Your classifier in SVM is based on f(x) = sqn(w′x +w0) (sign of) and if >0 then y=+1 and vice versa.

If you encforce hard margins then you try to classify accurately, perfectly, and also max dist bw supports. But that might mean no convergence. Just not possible.

The line below is w′x+w0 = -1
Soft margin means the condition of 1 will be relaxed.
Yi(w′x + w0) ≥ 1
We subract a number from it, say eta, ηi.
The I subscript means it is diff for each sample.
The bigger the zi, the bigger the misclassification allowed. So model will be more general, preferring variance > bias.

Penalty method vs Lagrange:
Min f(x) + cg(x) with g(x) being the penalty function, the c isn't your Lagrange multiplier, but rather penalty parameter which is a hyperpara and not optimized along the way.

Yi is the label of your samples
Xi is not a label, it is a vector.
W is also a vector, it has as many components as xi does. But not just a function of Xi but also yi and ai.
ai is Lagrange multiplier.

How is xi a vector?
Why is ai with I?
f(x) = syn(∑yiai(x•xi)+w0) dot product of x and xi is basically multiplying the coordinates of x point in question with all other x points coordinates (vector of xi)

This is what will decide the entire formula.
We use a kernel for this dot product: k(x,x′) = x•x′
These kernel functions can vary, can be linear or nonlinear. If we project the data in 3D space, and then apply these kernel functions, we have linearly separable data.
We don;′t need to know actualy form of function to project.
Now u are working with 3 features.
Using Φ(x) u will map. So instead of x you will have Φ(x) in f(x), with 3 features.
Now, W = ∑aiyi (Φ(x)•Φ(xi) + w0.
We need a kernel function for this dot product.

The diff kernels you have for this, a kernael for x and xi. X′ = xi.

1. Linear kernel: k(x, xi) = xTxi — if linearly sep
2. Poly kernel: k(x, x′) = (rxT•x′ + c)^d, try diff powers of d and diff values of scaler r.
3. Gaussian = exp(-r||x-x′||)
4. Hyperbolic = tanh(rxT•x′ + c)

These are called kernels cuz we can cal the dot products without knowing the function. Predicting the product can be this and that.

Your suppotrt vectors decide what happens.
Sklearn already cal the ai before for all so it doesn′t multiply x and xi for all samples, just for the critical ones or the ones lying on support vectors. Ai will be 0 for all points except for support vectors.

Change the kernels if alg isn′t able to sep the data linearly.

support_ is a n att to get the support vectors after fit.

SVR for regression.

GridSearch can help you optimize your C parameter in penalty function of SVM.

DECISION TREES

Info Gain (IG) = Entropy(T) - ∑ Entropy of each split after the feature.
The latter is called expected entropy.

Or more accurately, IG = Entropy(T) - ∑si/T • Entropy(Si)
Si is basically split i. This can be two, e.g. income is H or L. But it can also be 5. H, Very H, M, L, V Low.

The tree ends at overcast sub feature of outlook cuz it is always +, the Y=+ always or P(Y=1 | Outlook = overcast) = 1. So that's our leaf node.

Every fit takes 2DIM ARRAY and test takes 1D.

U can also pass diff estimators in grid search in the form a dict.

Squared error is being used as a criterion, same as s.d for DTregression. You can also optimize criterion in grid search actually. Anything goes.

In sklearn check description of data, maybe it is already standardized or normalized.
X= Data.data to have a df of features
Y = Data.target

Dif bw class and function.

The decision features have to be 2 for plots.

DECISION TREES (2)

Decision criteria based on entropy or gini.

Info gain highest is selected.

Initially you will get same entropy for all Xn.
Entropy is cal of Y, entropy of vegetation = 2/7 log(2/7) - 2/7 LOG(2/7) - 3/7 log(3/7) = 1.5567.
ENTROPY IS CAL FOR EACH VAR, BOTH Y AND X.

Wouldn't the results of info gain be higher if small n? Cuz only one possibility for many attributes/classes. Nodes will quickly become pure nodes.
Only one sample here in leaf node so overfitting. The further down you go the more overfit and useless. So you cut branches. But then it won't classify then. Will give error on those cut branches.
To cal noise of data just find variance. If noisy data, then depth is issue. If not,

then let it go to the ground.
In DT you are gonna notice that you don't need to use all features. If you just know elevation is low, that's enough. No need for attributes. That's what dim reduction is about.

For cont Y, instead of entropy you have variance. I.G = Var(p) - ∑Nc/Nd•Var(C)
You put the cont X var in ascending order. So you can see labels of Y against them. And then you see which pairs or which two are such that label changes. You take the average of those two cont values where table is changing as the threshold. Multiple options. So u cal gini or entropy of each and split on that threshold.

Random Forest basically takes the mode of the results/calssificication given by multiple DTs. The multiple DTs are trained on diff data. These diff sets of data are generated by same pop data but using bootstrapping, so sampling with replacement but same n size (multiple records may repeat).

If u wanna use stochastic gradient but with regression then use the SGregression class, pass the alg/method to this.

Isn't hyperpara tuning also ensemble learning in that you do parallel learning with many diff forms of same algorithm (diff hyperparams) and see which works best?

Try bagging or boosting or bootstrapping or voting.

The 3rd main method is called Boosting. Two types: Adaptive Boosting and Gradient Boosting.

Clf._class_._name_ returns the name of classifier.

In log reg, recalll that your Φ(x) or y^ had boundaries, say if z≥0.5 then yes.
You can change the 0.5 threshold of classifying bw 0 and 1.
So to improve recall and precision, say if you wanted higher recall than precision, you can play with the threshold in the ROC curve. See the ROC curve and decide accordingly.

In clf, u can use decision_function attribute to get the score/threshold the clf is using.

If all classifiers have diff activation functions, then obviously the scores will be quite varying for all. The z value. So obviously the diff thresholds u define will be diff for each clf whose scores are diff.
The idea is that you cal the scores of each sample in a clf. Based on the scores of each sample, u decide the threshold.

Deep Thought answers in the only language it knows, it says “42” giving life an equivalent meaning to that of a variable or wildcard. In essence, Deep Thought is saying that the meaning of life is whatever you want it to be. 42 = life is what you make of it

HET: What are Marshall’s main 5 contributions to Econ?

1. Wants < Activities (D < S) and ∀ interconnected
2. Partial Eq and Ceteris paribus (since ∀ is interconnected)
3. Elasticity of D,
4. CS and PS and
5. Time periods (Market period, SR, LR, and Secular)
6. Economies of Scale and Diseconomies (internal and external)

HET: What's Marshallian Cross?
Just the S D diagram (cross) with partial eq.

HET: What was missing from Marshall's D and S analysis, etc?
Complements and substitutes were missing from his analysis.

HET: What are wants and activities for Marshall?
Wants are demands and activities are S. Supply is what will vary according time, that's his time period concept. Also supports Say's Law, so S>D.

HET: How does Marshal derive the idea of ceteris paribus and partial eq?
He says everything from S and D is so interconnected so gotta hold some things constant if you wanna see things isolated.

Gossen's 1st law is Law of Dim Utility.

HET: What did Marshall say about how the time periods will be decided?
By how much FC, VC there are etc, so based on costs and which FOPs can be varied.

HET: Marshall: The MC curve becomes the Supply curve when (1) MC is positive slope (2) after it intersects AC curve, all points thereafter.

Marshall contributions:

1. CS + PS
2. EOD
3. Quasi Rent
4. Partial Eq
5. Taxes and Welfare
6. Time period (Supply)
7. Distribution
8. Wants < Activities. Wants are demands and activities are S. Why are Activities more imp? Supply is what will vary according time, that's his time period concept. Also supports Say's Law. But the most new idea of his is that these are all INTERCONNECTED.

SE and IE concepts derive the idea of giffen, normal and inferior goods.

He talks about not just micro but also macro when he delves into welfare and how all these policies will affect the welfare of people and poverty.
He was considered the daddy of Econ cuz considers that his theories will ∂.
He is the first one to start the rational choice theory.

Marshall's Use of prices to measure U dep on (1) an additive U function that ignores Sub and Complementary RSs. (2) The IE from small p changes is negligible i.e. MUm is constant.
He says the Marshallian cross, the S and D diagram, shows the price is being decided at the margin, the last person. So there must be some CS being earned by all consumers before.

By looking at the FCs and VCs he came up with the conclusion that the firm actually min losses by operating as long as TR>TVC cuz that will help you cover FCs.

HET: How did Marshall's Quasi rent concept differ from that of rent?
A subset of rent that's tentative.

Marshall theory of distribution.
When we have our distribution we should accommodate people who don't contribute anything.

Clustering or K-means drawbacks?

1. V sensitive to outliers
2. V sensitive to scaling.
3. Needs convex data (can be grouped nicely)

EDA: How to deal w the issue of poor random initialization of centroids in normal K-means?
Use k-mean++.

How does k-means++ defeat random centroids being too close or poor intitialization?
it will pick the random centroids v far away from each other. Centroid C1 and C2 will be far away from each other. These centroids won't be just arbitrary coordinates.

How would k-means++ go about randomly initializing centroids?

1. First centroid will be x point/obs, randomly.
2. The C2 will be the point/obs farthest from C1, based on distance, euclidean.
3. The C3 will be obtained by cal the distance of all obs from C1 and C2

and choosing the obs which is the farthest away from both (by summing the dist)

Hierarchical clustering is what?
Agglomerative clustering and Divisive (top-down) to deal w convergence point issue.

How does Agglomerative clustering work?
Bottom-up Approach: Considers every obs as a cluster. And then builds up from that, clustering the closest two points at each stage.

Say i1. We cal the dist of i1 from all i2, i3, i4, and i5. The distance = ∑(f1 - fi) i.e. all the features of the obs.
Basically find the distance matrix. We see which obs are closest to each and merge them obs, say 5 and 3.
Then u make a dendrogram. It shows all the mergers.
So now u have a new combined point, 35. And again sketch the distance matrix. All entries stay the same but 35 will have diff entries.
How do u cal the distance from 35? U have multiple options, all of them are under the hyperparam of Agglomerative clustering by the name of linkage. Can be single linkage, complete, or average.

EDA: Where do the hyperparams of Agg/Divisive clustering come into play?
If grouped 3 and 5, 35, how to cal the distance to other obs?
The hyperparam of Linkage has the diff methods: can be single linkage, complete, or average.

EDA: Divisive Clustering is the rev process of Agglomerative Clustering.

EDA: What's the purpose of a dendrogram?
The dendrogram will show all the clusters being made, all the mergers.

Divisive Clustering works how, generally?
It will start w one big cluster and then converge when each obs is its own cluster. Same distance matrix and all.

K-means clustering works best for what? Data?
when you have convex data/obs. So the linear combination of any two obs is within the region.

EDA: If data is not convex, then do what clustering? Cluster at all?
should use DBSCAN clustering. Doesn't assume spherical shape.

EDA: A dendrogram is built on linkage/dissimilarity matrix.

EDA: What's complete vs single linkage vs avg linkage?
Complete: Based on max distance bw Xi and C1 (cluster formed with obs)

EDA: Prob with Agg Clustering is u must choose a breaking point. Where do u cut the dendrogram, horizontally?
Based on the Euclidean distance, which is based on num of classes—must infer. For iris we have 3 classes, so Euc = 3.5 which gives us 3 clusters, one for each class.

Learn how to interpret silhoutte index or plot.

If the number of Xi are same, or same numb of diff flowers, then same thickness of clusters in Silhouette. If the bars are almost rectangle, each bar rep a cluster, then u have tight, good clustering.

The problem with traditional clustering is that it will clusterize EVERY obs, so even outliers. So u want sth that has fortitude against outliers and noise.
DBSCAN - Density-based Clustering Application with Noise.
It will cal the density of each cluster, or how dense are some points in a region, if not dense enough then it will drop the clusters w few values.

K-means is based on shape, assumes spherical.; DBSCAN is not. It is based on density.

DBSCAN has 3 elements:

1. **Core point (similar to centroids but not quite)**
2. **Border point (∀ core point, ∃ are border points)**
3. **Outliers**

The hyperparams are epsilon (eps or e) and minpts.
The former decides how much distance, or radius, u will see around a point to check how many obs lie in that eps radius circle.
Latter decides how many minpts must a point have to become a core point. If an obs x5 has 3 points in its radius of eps=0.5 radius, and if minpts=5, then x5 becomes border point, not a core point. And any point not

No need fit_predict here. No labels, so no predicting stuff.

Math has to be guided. It can be guided through your intuition or exp.

Dimensionality reduction can be done through PCA or SBS (seq backward selection)

We are looking for directions where variance is max. We do a covariance matrix and then find variance where it is max.
If you want to find the direction, u get the eigen values of this matrix since ∀ matrix can be decomposed into eigen vectors.
Eigen value is the mag or weight of your eigenvector.
We will have 4 indep directions in the diagonal of the matrix.
The vectors/directions are going to be orthogonal. The principal dir of a matrix are its eigen vectors. U can produce those via matrix.

The dot product of a with b will get you a in direction of vector b.

Matrix of Eigenvector = W = the directions of the orthogonal vectors.
Your data has matrix X, with features as cols.
W has many directions, cols = features initially.
So you have X•W=X.
The W can't all be selected, must choose only the highest eigen values directions so that the transformed matrix of features, reduced one, X, is of lesser dim.

Thus, there's a diff bw compression and selection. Data reduction or PCA is unsupervised. We don't give a shit about the classes or labels. But in feature selection we do need the labels.

When we transform, it projects the data on those directions/projections in PCA's fit (on train data).
For PCA, u get the cumulative lambdas or eigenvalues. U plot a step function of the lambdas. The higher the step the more the variance. The y axis has examples variance ratio: Λi / ∑Λi

LDA AND FACTORS EXTRACTION

PCA can be used for both supervised and unsupervised learning
LDA is only for supervised. PCA doesn't consider labels or classes when u reduce dimesnions. SO it merges classes. But LDA, supervised learning, considers classes.
LDA seperates the classes but keeps the variance of the classes to min.
We need to find such direction s.t. when u project obs on that line/direction your data becomes 1 dim. A dir that min the within class variance but max the

exogenous class variance. To avoid dim red s.t. the points within classes go too wide, we put a constraint that reduces the var within classes.
Projection means just take perpendicular from an obs to the projection line.
The issue with the oval data points show that if you project down on the x axis line then your μ are actually far away, but the points of class 2 are overlapping with the obs of class 1 when projected.

The line's eq is y=w'x. All points will have coordinates given by y.
The w is what we need to find. It is w' = [w0, w1]' in case of 2D.
U want to max the distance bw the means so J(w) = |μ = μ2| = |w'μ1 - w'μ2| = |w'(μ1-μ2)|
We will define two measures for optimization/metrics of projection quality: within class scatter and bw class scatter.
For each class, we define the within-class scatter (same as var) or sum of squarese diff bw the projected samples and their class mean: si^2 = ∑(y-μI)^2

Our overall score for a w is J(w) = |μ1-μ2|^2 / (s1^2 + s2^2)
Max the top—bw class cluster. Min the bottom: within class scatter.

When u diff a function w.r.t a vector u get a vector
When u diff a vector w.r.t. a vector u get a matrix.

Si = ∑(x-μi)(x-μi)'
Is this the formula for variance?
May not seem like it, but this the variance for each class. Does not have sample size, n, in den, cuz we assume obs are uniformly distributed in each class.
Sw = Si+Si
This quantity is nothing but covar.
Vector mag |x|^2 can also be written as dot product: x•x. Or as x'x.
Sb is the scatter bw the classes.
So simpllifying stuff gives us J(w) = w'Sb•w / w'Sw•w.
Vector calculus: x'Ax rep the quadratic form. X1^2 + X2^2 can also be exp this way.
D/dx
D/dx(x'Ax) = Ax'+Ax' = 2Ax'
The w we were finding is an eigenvector of the within class and bw class scatters.
Simplifying down u get Sw^(-1)Sb•w = Λw.
We have inverse Sw cuz we want a direction of w that min the within classes variance.
If we had Sb•w only

Λ is just the eigenvalue of the eigenvector, which is w.
The number of w, or eigenvectors, u get dep on the number of the classes that are in the data.
IF 3 classes then 3 eigenvectors.
Which eigen vector u choose again dep on the explained variatnce ratio formula

from PCA. Use that.

In Numpy, u take transpose of matrix when multiplying with anohter matrix.
M•N′.

First u should calculate how many eigenvectors are giving u useful information.
U should do the eigenvecotes first and then run LDA or PCA.
For both LDA and PCA u must normalize your data. V sensitive to outliers.

DEEP LEARNING

Keras is the package for DL.
Conda install tensor flow
Pip install tensorflow

The w0 or intercept is called bias in Neutral network (NN). It is the shift factor.

Activation function can be based on sigmoid or whatever. It is not the same as the threshold function.
Loss function is needed to optimize the results/model, a metric is needed.
L(w) = ∑[y-y^(Φ(z)]^2.
U can switch diff number of neurons u want in your NN, your activation functions, threshold functions, loss functions (which has error and penalty functions both, so two in one), your updating function or eta value.
Wnew = Wold - etaΔ•L(w).
The eta is the learning parameter; doesn't have to be rotational factor. That's only in perceptron or logistic.

Input layer are those
Hidden layer: connecting those input nodes to a neuron or more than one.
Output layer.
Each input is connected to a neuron. But what's the weight given to each? X1 connected to N1 is w11, to N2 is w12 and so on, how many every neurons there are.
The reason there are multiple layers of neurons is because we want to capture non-linearity. Non linear features can be captured. We therefore have many weights, not just one. First weight, w1(1,1), is the weight for connecting X1 to 1st neuron and then x2(1,1) is for connecting z1(1) to z2(1) and further. U pass all the signals to 1st nearon, but then u also apply activation function to that neuron so that the signal passed to the next neuron in chain is designed accordingly. Once all the wights are applied and all neurons have received the signals, the final neurons send their signals (after activation function) and that gets aggregated. If meets threshold function then fires or not.

Convolution NN or CNN and Artificial Neural Networks (ANN) and lastly

Recurrent NN (RNN). The diff is how is one layer is connected to another and the connections bw layers. Sometimes signals are sent back to to previous layer, that's RNN. For feedback.
What we did is Feed forward propagation NN.

There are 3 main packages in Keras.

What's a Tensor?
Tensor is a generalization of matrix.

EDA: What do diff orders of tensors rep?
Scaler is tensor of order 0. e.g. 1.
A tensor of order 1 is a 1D array or [2, 4, 5]'.
A tensor of order 2 is a 3D matrix. How do u even do that? U stack matrices on top of each other.

Keras has 3 main APIs: Models, Layers, and Core Models.
Models: how the neurons and the diff layers are connected. Are they only forward, sequential, backward, both, etc.
Model has 2 classes: functional (more flexibility and can move around) and sequential (go 1 by 1)
Layers: are the layers dense or not? Number of neurons in the diff layers. In layers u also decide CNN, RNN, ANN, etc.
Core Models: here u specify what will be your loss function, activation function, optimizer, regularizer.

MNIST DATA:
If u have 28,28 in the dim of the image data, that means 28x28 pixels or even better it means 28 columns or 28 diff features. So each should be treated as a feature.

A dense layer is one that connects an input with several neurons and vice versa. Hardcore bijection.
So z1, the neuron, is = w11•x1 + w12x2 +...
Then u apply an activation function on z1 itself to get z1' and so on.

Types of Gradient Descents: Batch GD, Stochastic GD, and Mini Batch GD.

Mini batch: takes small batches of data, a few obs, and u go through all the data and form batches and train until all obs are covered. That's 1 epoch. Then u run more iterations or epoch.
If hyperparam or number of samples n mini batch are set 1 then that's just Stochastic SD, and if u do it n = n (actual number of obs) then that's just batch SD.

Stochastic SD: u take just one obs and find loss function and then other obs and so on until all obs are covered. That's 1 epoch. U keep doing epochs until your loss function criterion is satisfied. The criterion can be avg errors or avg loss of all obs, or it can be that the max(loss) < some predefined criterion.

!pip install tensorflow on Jupyter.

Datasets from Keras isn't in df or array format (Sklearn type) but rather in tuple format.

The data already comes in the form of train and test, and it has 4 partitions

The data is in greyscale so every pixel has just 2 values, not like RGB/color where it has 3 values.
Thus you see all those 0 values in the array for they represent the pixels. The pixels can be from 0-255.
Every image is 28x28 and every image has just 1 value. If it had 3, like in RGB, then u would have

It is a 3D tensor. So img[0] is the first image. It is stacked. One on top of another. Stacks, bro.
To show the image, use imshow from matplotlib.
Models can be sequential or functional.
NN.add(layers.Dense(512, activation='relu', input_shape(28*28,)))
Layers is the class, you are making an object within the argument of add. NN itself is an object.
512 is the num of inputs/nodes.
Dense is the type of layer we selected. Every node will be connected to another.
The
The shape arg defines what's the shape or dim of your output. We try not to work with matrices cuz computationally expensive.
input_shape(28*28,) does this.
Now we have defined the first layer.
THe last layer, assuming we are having just 2 in total, will be the output layer.
So the nodes of the last layer dep on the type of classification problem u have. If binary, then just 2 nodes. Since problem is multiclassication where there are 10 diff outputs, from 0-9, we will need 10 nodes in the last layer:
NN.add(layers.Dense(10, activation ='softmax'))
The activation function must be decided on every stage or layer definition.

Next stage is definitng your optimizers. Yr loss function and steepest descent.
NN.compile is the function that will compile your loss function, your optimizer, and your layers. It is your pipeline function from Sklearn.

NN.compile(optimizer = ‘rmsprop’, loss = ‘categorical_crossentropy’, metrics = [‘accuracy’]).
Scaling is v important in NN data. U use one-hot encoding.
How do u do so for 9 classes? U form tuples, first one has [1, 0, 0….] so all classes are 0 except 0. Similarly for [0, 1, 0, 0, ….] for rep label 1.

train_label = to_categorical(train_label).
Same for test_label.

The loss functions are multiple but for classification u will bed using one from the group of probabilistic loss functions, obviously not regression functions or hinge loss. Why not hinge loss?

We also need to reshape the images to this size. train_img((60000, 28*28))
Every image has max value 255. In grayscale the range of values are from 0 to 255.
So for scaling u just train_img = train_img/255.

NN.fit(train_img, train_label, epochs=5, batch_size=128).

Yi = wi•x.
We use activation functions on these Yi to bring non-linearity in our NNs. Otherwise they would be just simple linear reg.
Φ(y1 not subscript) = Z1
And then u apply w2 to z1, the 2nd layer weight, to get Y2. The 2nd layer neuron output, and then u again apply activation function to it to get z2 = Φ(y2), and then again apply 3rd link/layer weight to get Z3.

Activation functions can vary between layers, but not within layers, depending on the problem.
Mostly activation functions at the last layer are sigmoid functions cuz of binary nature of problem.

Definition of linear function: f(x+y) = f(x) + f(y)
E.g.

Relu alg is Rectified Linear Unit.
In all hidden layers we use Relu, cuz it is a non-linear function so brings complexity but it’s the most simple sort of non-linear.
Use MSE for the loss function.
How many layers u want and how many nodes u want are hyperparams of NNs.

U take nodes as 8, 16, 24, cuz that’s the ram of your computers. Not required

though. Can change to any. Just a convention.

In **binary** you use relu in hidden layers.
Output layer use sigmoid function
Categorical cross entropy for loss function.

Multiclassification: again relu in hidden layers, in output layer use soft max, and use multi cross entropy as loss function.

U have two labels so u need cross entropy. U have predicted labels and actual labels. So u can cal the entropy of both and try to min them.

Cross Entropy = -∑ yi log(f(zi)

NN.add for adding layer has argument input_shape = (28*28,) to specify what sort of argument it will pick and accept. It will accept 1D data, the images turned from matrix to a vector. Tensor bro, 1D.

We always do one-hot encoding of our output in NN for classification. In all classes, regardless of number of classes. No binary classification. It is req in one-hot encoding.

Epochhs are like features in a sense. If you inc too much then it might overfit. U expect loss function to dec as epochs inc until a turning point is hit. Then loss function starts inc as u inc epochs.
This because the output and loss/feedback of the first output to the 2nd epoch and it gets better. It gets fed. Through steepest descent. Xi+1 = Xi-ΛΔF(X0)
Batch size doesn't have to multiple of total num of obs.

LOSS FUNCTION AND VALIDATION

We have to run loss function on both the train and validation set.
The more data u have the more layers u need to add. The more xomplexity there is in the data since more n.
U can change the num of nodes, layers, and epochs and cal the loss function on each combo and see how the change is happening and the optimal combo of these hyperparams.

How is the coding of text data going to happen for each obs?
Based on freq of the letters in the data. The most common letter will be 1 and next 2 and so on. SO u can codify the data from letters to numbers.

Tokenization is converting text data to numbers and vice versa.

Before u can pass these tokens or num converted to model or layers, u need to

one-hot encode them. To do so , u need to make a vector of 10000 columns for each obs/number set. So 60 would have a vector in which all columns are 0 apart from 60. Cuz we have 1-10,000 values so 10,000 diff columns.
We will convert the whole data into these sets of vectors.

For binary classification, use need just 2 outputs at end, 0 or 1, so u use activation function sigmoid.
For multiclassificaton, use softmax.
For regression, nothing. U don't need t code the output. Plain is good.

Partial x train is train set after validation set taken out.

Taking batch size is for the sake of efficiency.
1 epoch is basically all the data observations going through.
Within the epoch u can define how many obs will go together in a batch, that's batch size.
Changing epochs and batch size does not change how much data is processed. All data will be processed regardless.
If u make batch size 1 then 1 obs goes each time so 15,000 iterations in 1 epoch.

Stochastic gradient is where u pass just one obs at a time.

In CNN we may have some params that Arne't being trained.

Even here, the metrics are the same: loss, accuracy and even recall and precision if u wanna find out.

This is the ANN model.