**Theoretical**

Linear regression and model selection:

1) Lin reg is a closed soln, finds the optimal fit by minimising MSE.

2) Inc complexity = lower trg error as it has more freedom to fit to trg points, but after a certain complexity, it startts fitting to noise and test error starts to increase = overfitting.

3) More complexity = memorises noise, unable to predict unseen test data, inaccurate

---Choosing best method---

a) CV

i) Split into train and test data (randomly)

ii) Fit to train and find MSE on test, lowest MSE = best model

iii) + Cheap - Wastes data (test) that could have been used to train the model better

b) LOOCV

i) Split into k-1 points, last point is test

II) Fit to k-1 points, test on the last point, find the error

iii) repeat until every point has been used as a test set, find the mean error on all these points, that is the MSE

iv) + uses all data to train the points but its expensive

c ) K fold

i) Divide into k datasets, use k - 1 to train and last set to test. Find MSE error on that test dataset

ii) Repeat for k times to get all the errors, find the mean of all the errors.

d) Nested k fold

i) for all the k - 1 training datasets, further split them into train and validation, train on inner train, find best model on validation then repor the error on the test set outside.

ii) Find the mean of MSEs on test

-----------------------------------------------------------------------------------------------------------------------------------------------------------

3) Model selection and classification;

a) How to select diff models (subset selection, feature reduction or regularisation)

b) Classification: Not like regression, instead of predicting values (cont) you are predicting a certain class (based on majority vote)

Classification:

for feature xi,x2...xn, you have a prediction(class) that is either +1 or -1.

Weight vector has bias in it (b0,b1,b2,b3...) and feature vector is (1,x1,x2...xn)

Dot product is the classification (the hyperplane). Use eqn of the hyperplane to get the prediction (+1) or -1

Hard margin = No misc, low bias high variance as it changes alot when new datapoint is given, still has a margin but small

Soft margin =misc, high bias low variance as it allows for misc, lower overfitting to training, allows for points to be wrong, penalty in the form of budget is given. Penalty is when point is on the wrong side of margin, introduced in the form of slack variables that are (+)ve if they are misc

Support vec = points on the margin borders or within the margin, affect the shape of the hyperplane.

----------------------------------------------------------------------------------------------------------------------------------------------------------------

4)SVM

If points are not linearly separable, add another feature to transform them to a higher dimension, then plot a line through it..Eg changing feature 1 from x to x^2 (works great if the points u wanna separate are 'stuck' or 'sandwiched' by other points. If these points are 'stuck' in 2D, add a 3D feature then...from x to x^3.

Classification for non-binary points--linear separation not as easy now..use either 1v1 or 1vall

1v1 : if u have a b c d features, 1v1 is a vs b, a vs c, a vs d and so on, total number of linear classifiers = nC2, n = number of features

At prediction: classifier asks which side of the hyperplane is the new point on, takes the label with the majority vote

1vall: if u have a b c d, 1 v all is a vs bcd, b vs acd and so on, total number of classifiers = n = number of features given

At prediction: Classifier asks which side of the hyperplane is it on, maximises the distance on +1 side of any hyperplane if it falls into any region (eg if its in the region which has both a and b in it, it will choose the side where the distance from the hyperplane is the most, so it checks distance of the point from the line of a vs bcd and b vs acd and whichever line is the furthest from the point is the answer (or in the case where it cant be predicted by any point (no man's land), it will choose the closest hyperplane it is to and it will predict that region)

---------------------------------------------------------------------------------------------------------------------------------------------------

5) Dec trees

Non linear methods also used in classification and regression (until now we saw the use of SVM and linear regression/polynomial for both respectively)

Easy to interpret and are also in 2 forms, regressor trees and classification trees

Regressor trees:

Uses recursive binary splitting, checks the region which has the lowest SSE and plots the line there. Its a greedy algo because it only looks at the next best step, reaches local optimum and not global (in this case, there is no way to reach global optimum, greedy algo works best!)

Predicted point = mean of all the points in the region

If a new point is given, based on its feature, it will be put into a region and then based on the mean of region, it will be given a value (prediction)

ERROR TESTING: Uses SSE, similar to MSE but no need to \*1/n, its just the sum of squared errors.

PROBLEM: Might split into too many regions (max\_depth remember! More max depth = more distance from root node to leaf = more splits = overfitting)

Classification trees:

Doesnt predict value, predicts class. Can be non binary too but cannot be continuous (thats regressor yeah)

ERROR TESTING: O/1 Loss, Gini, Entropy

0/1: Just counting the number of misclassified points, if asked for average 0/1 loss, find (number of misc points in that region/number of points in that region)/(number of points in that region/total no. of points), basically weighted average. NOT GOOD FOR GROWING TREES AS NOT SENSITIVE ENOUGH

Gini: G = sum up pmk\*(1-pmk). Assign the regions their numbers, so m = 1 to n and for each region (each m), find the proportion of class k in that region and apply formula. Once u have the total Gini for that region, multiply by number of points in tat region if finding the total Gini or multiply by the weighted average of all points if finding the average gini

Entropy: Similar to gini, find m, for that m apply formula but now its -Sum of pmk(ln(pmk)), as usual if its total entropy multiply each m value by the no. of points in that region or if asked to find average, multiply by weighted average of the points it tnat region.

--------------------------------------------------------------------------------------------------------------------------------------------------------

6) Ensemble

Fancy name for combining diff models into 1. Why? bec as discussed earlier dec trees are bad at predicting (overfitting), but combining diff dec trees (ensemble) reduces overfitting and variance.

Bagging: Split data into similarly sized datasets and randomly sample with replacement from the original dataset and put them into new datasets/bags/bootstraps. Then, train model on these bags, find the average prediction on the test set.

Find average of all these predictions (for bagged regressor its ave value and for bagged classifier its majority vote)

Increasing B or number of datasets does NOT lead to overfitting, because u are just training from the same points in the train dataset again and again. Increasing B INCREASES ACCURACY

Increasing max\_depth DOES lead to overfitting, more depth = more splits = more classificaitons = more chances of points being in their own region

Random Forests: Optimised verison of bagging, almost the same but when putting points into bags, it also randomply samples sqrt(p) features (so it samples from all rows (points) but only samples from certain features (columns)). This reduces presence of strong feautres, root nodes might be different, greater decorrelation among bagged trees = more bias = less var = less overfitting = more accurate prediction on unseen data

ERROR TESTING: Uses OOB, Out of error on points not used to train (test set), similar to MSE

---------------------------------------------------------------------------------------------------------------------------------------------------------

7) Proba predictions

Truth telling: y = what u report, p = your true belief about an event occuring, x = whether event occurs or not

If its proper, y = p to maximise ur expected value, if not proper, y =/= p. Proper is quadratic and logarithmic, non proper is linear

Lin is not proper bec when u plot the graph of p\*y + (1-p)\*(1-y), the graph is upwards sloping, highest expected value = y = 1 but that may not be what your p is

Quad and loga are proper because when you find the expected value, u can then find the max value by doing dy/dx = 0, and u will see that y = p for both cases

Expected value (lin) = p\*y + (1-p)\*(1-y)

Exp value (Quad) = p1\*(1-(y-x1)^2) + (1-p1)(1-(y-x2)^2), at max value y = p. Remember that x may not always be 1 and 0, it could also take other values. The first term is for value1 of x or x1 and the next term is for x2, another value for x.

Exp value (loga) = p\*ln(y) + (1-p)ln(1-y), same as quad

Problem with loga rule : Your expected reward always < 0, high weight on very extreme values if u get wrong, if u guess 0 and 1 comes out, your 'reward' = -infinity, same for the opposite where u guess 1 and 0 comes out

Now, if u know the actual probability of an event occuring, p is replaced by theta, everything is still the same.

Expected value if u somehow manage to guess right is when y = theta, your expected value is theta^2 + 1 - theta. Hence, the loss you get arises when you fail to choose theta, which is : Expected value when you choose theta (perf scenario) - Exp value when you report y = (theta - y)^2. So you can also use this loss functuon to see which forecaster is better

DIFFERENT SCORIGN SYSTEMS rate differently, by loss u may say A is better than B but loga rule might produce opposite results.

More than 2 possible choices but only 1 right answer?

Score for Quad = y(j) - sum of (y(i))^2 - 0.5, y(j) is the probability of the actual correct answer, yi are all the probabilities u assigned, squared

Score for log = ln(y(j))

Real valued outcomes: Continuous?, beta = upper bound, alpha = lower, y = value u predict, x = actual value

Median = (beta - alpha) + abs(y-x)

Mean = (beta - alpha)\*\*2 + (y-x)\*\*2

------------------------------------------------------------------------------------------------------------------------------------------------------

8) Value of info: How much am i willing to pay for more info?

First, find the expected value under normal circumstances, using quad rule/loga rule whatever's specified. That is your benchmark

Next, if u have a new piece of info, draw a tree diagram to illustrate the probability of getting diff scenarios. Find the expected value in each 'branch' and then multiply by weighted average of that branch to get new expected value

Money I have to pay = New exp value - old exp value

New expected score > expected score of prior, prior graph = theta^2 + 1 - theta, the new expected score sia atreaight line on top of the prior graph

Positive affine trf = if u linearly scale the expected score of a proper scoring rule, the new exp score is also scaled by the same amount

Aggregation = combining diff predictions into 1, more accurate bec if more people believe smth to be true, the P goes up

Extremising : Quantitative version of aggregation, pushing values to 1 or 0 depending on if they are above or below 0.5

------------------------------------------------------------------------------------------------------------------------------------------------------

9) Bayesian:

Bayes rule to update ur own belief, combination rule to find aggregated probability. Then compare the mean of 2 updated beliefs with the aggregated probability (optimised) and decide where to push value towards (1 or 0)

2 forecasters :

Winner takes all: We said earlier quad = proper bec u are incentivised to say the truth, y = p and so on. But if 2 people say the truth, y = p for both of them so the chances of selecting a winner = 0.5. Hence, one person misreports so that he/she can maximise the score. (You want to max your score or minimise the score of ur competitor). So if u believe that P(x=1) = 0.8 and the other guy also believes p = 0.8, u purposely misreport y = 0.9 because u are now closer to 1 than ur competitor (u choose 0.9 he/she chooses 0.8), ur expected score is higher >>> called advantageous misreporting

>=2 forecasters: You use the formula for finding the probability of selecting the best forecaster, leads to truthful outcomes

---------------------------------------------------------------------------------------------------------------------------------------------------------

1o) SP Algo

Info report : what did people actually report, if 3/10 ppl voted option A, 2/10 B, 3/10 C and 2/10 D, the info report vector = .3,.2,.3,.2

Prediction report : info report of people \* prediction respectively

SP Algo : If so many people were predicted to choose A (Prediction report) but a lesser number of people chose it, A is wrong. But if few people were predicted to choose B and in actual fact many people chose B, B is the answer. So compare the surprise by doing info report (actual distribution of people choosing) - predicted report to see the biggest difference = winner.

CODE

MODELS (Lin, Polynomial, DecTreeReg, DecTreeClass, Bagged\_tree, RF)

GENERAL STEPS FOR MODELS:

1) Load data, split into X and Y and reshape them into 2D arrays (-1,1)

2) Use traintestsplit to split X and Y into training and testing data

3) Define your model (either make\_pipeline or DecisionTreeRegressor/Classifier etc)

Make\_pipeline(PolynomialFeatures(degree=3), LinearRegression ())

4) Fit your model to training set

5) Define MSE

6) Define X\_seq by using np.linspace

7) plot the scatterplot, plot the prediction on x\_seq and find mse on xtest ytest

data = pd.read\_csv('xyz.csv')

x = np.array(data['feature1'])

y = np.array(data['feature2'])

x = seq.reshape(-1,1)

y = seq.reshape(-1,1)

X\_seq = np.linspace(0,100,10000)

def MSE(polyreg, X, Y):

return ((y - model.predict(X))\*\*2).mean()

###model = make\_pipeline(PolyFeatures(deg = x), LinReg())

###tree\_reg = DecTreeReg(min\_split\_samples = x)

###tree\_classifier = DecisionTreeClassifier(max\_leaf\_nodes=3, criterion="entropy",max\_depth=2) ###Max depth is the length from root node to leaf

###Increasing max depth leads to overfitting

###tree\_classifier.fit(X,y)

###tree.plot\_tree(tree\_classifier)

###tree.plot\_tree(tree\_classifier, max\_depth=3)

###bagged\_regression\_tree = BaggingRegressor(base\_estimator=DecisionTreeRegressor(max\_leaf\_nodes=max\_leaf), n\_estimators = 500)

###rf = RandomForestRegressor(n\_estimators=500, oob\_score=True, max\_features="sqrt")

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y, test\_size = n, r.s. = 5)

model.fit(X\_train, Y\_train)

print('The mse is:', MSE(model, X\_test, Y\_test)

plt.scatter(X,Y, color = 'blue')

plt.plot(X\_seq, model.predict(X\_seq), color = 'black')

plt.show()

-------------------------------------------------------------------------------------------------------------------------------------------

SVC

GENERAL STEPS FOR SVC:

1) Create clusters of X

2) Combine x define y

3) Create SVC object

4) Fit to X,Y and plot model plt(SVC,X,Y)

1) Create 2 clusters of x

mean1 = [1,1]

cov1 = [[0,1],[1,0]]

x\_clust1 = np.random.multivariate\_normal(mean1,cov1,rs)

repeat for cluster 2

2) Combine x and define y

X = np.append(xcluster1, xcluster2, axis = 0)

y = np.repeat([1,-1],10), so xcluster1 which is column 0, gets label 1 and xcluster2 gets label -1

3) create svc object

svc1 = SVC

-------------------------------------------------------------------------------------------------------------------------------------------------

Probabilistic Predictions:

1) Process the data by changing all the yes votes of the binary questions to a probability, (%/100)

2)Those that have no data, fill them in with the value of 50 (or 0.5)

3)Define the outcome variable, which is a vector with 1s and 0s that represent the correct outcome of 1 or 0

4) Define the functions for QSR and LSR.

def qsr(x,y)

return (1-(y-x)^2)

def lsr(x,y):

if x == 1:

return np.log(y)

elif x == 0:

return np.log(1-y)

-------------------------------------------------------------------------------------------------------------------------------------------

tree\_classifier = DecisionTreeClassifier(max\_leaf\_nodes=3, criterion="entropy",max\_depth=2)

tree\_classifier.fit(X,y)

tree.plot\_tree(tree\_classifier)

#tree.plot\_tree(tree\_classifier, max\_depth=3)