Machine Learning

**Lecture 0: Why Machine Learning?**

* ML is a type of AI that allows software applications to become more accurate by predicting future outcomes using some dataset
* Use case: reccomnedation engines, fraud dectection, spam filtering, malware threat detection etc
* Why is it important?
  + Gives idea of user trends in customer behaviour and business operation patterns
  + Supports development of new products
* 4 basic approaches:
  + Supervised learning
  + Unsupervised learning
  + Semi-supervised learning
  + Reinforment learning
* Supervised learning
  + Data scientists supply algorithms with labeled training data
  + Input and output of algorithm is specified
  + Ex. regression modeling, binary classification, ensembling
* Unsupervised learning
  + Do not require data to be labeled
  + Machine sifts through unlabeled data to look for patterns
  + Ex. deep learning, clustering
* Reinforcment learning

**Lecture 1: Linear Regression Single Variable**

* Linear regression model uses equation: y=mx+b
* Python code:
  + Create data frame using csv file *(df = pd.read\_csv(‘test\_csv.csv’))*
  + Import linear model from sklearn *(from sklearn import linear\_model)*
  + Create linear model (reg = linear\_model.LinearRegression())
  + Create line of best fit (reg.fit(df[[‘area’]], df[.price]))
  + Predict future values (reg.predict(3300))

**Lecture 2: Linear Regression Multiple Variables**

* 1 dependent variable is determined by many (2 or more) independent variables
* Ex.
  + Price = m1 \* area + m2 \* bedrooms + m3 \* age + b
* Independent variables are also called features
* m1, m2, m3 are also called coefficients
* Data preprocessing: Handling NA values
  + Figure out what you want to change NA values too
  + Ex. If one of the bedroom number values is NA, one approach is to find the median of all the bedroom values and sub it into all the NA values
  + In python use: (df.column.fillna(0))
* Create line of best fit
  + Use sklearn function *reg.fit*
  + reg.fit => (independent variables, dependent variables)
  + Python code: *reg.fit(df[[‘rooms, age, area’]], df.price)*
* Look at coeffiencents and intercept
  + Python code: *reg.coef\_ , reg.intercept\_*
* Predict futre values
  + Python code: *reg.predict[[2000, 3, 2]]*

**Lecture 3: Gradient Descent and Cost function**

* Gradient Descent is an algorithm that finds the best line fit for given training data set
* **A picture containing text, clock, gauge

  Description automatically generated**MSE (cost function) is the difference between the actual output value – predicted output value, squared divided by n
* Learning Rate: Is the adjustment factor of how much each paramter will move by to eventually get to the optimal value
* SSR (loss function)
  1. **Text

     Description automatically generated**Summation of the (difference between the actual output value – predicted output value) squared
* Gradient Descent steps (one variable – specifically the y-intercept for linear relation):
  1. Find a metric to evaulte how well a line fits the data (Loss function)
     + In this example, let’s use Sum of Squared residuals as Loss function
  2. Take Derivative of the loss function with respect to that one variable
     + In this example, partial derivative of y-intercept
  3. Pick a random value for the intercept
     + In this example, you can start at y-intercept = 0
  4. Calculate derivative with that chosen random value (fancy word for slope)
  5. Plug that slope into step size calculation
     + Step size = slope \* learning rate
  6. Calculate new intercept
     + New intercept = old intercept – step size
  7. Plug new intercept into derivative an repeat everything until step size is close to zero
* Note: Gradient descent is very sensitve to the learning rate
  1. Certain learning rates will not find the minimum loss function
* Note: Sum of the squared residuals is just one type of loss function
  1. Gradient descent works the same way, with any type of loss function
* Gradient Descent (General method for multiple parameters)
  1. Take the derivative of the loss function for each parameter in it
     + This means finding the partial derivative with respect to each feature
     + ML lingo: ‘*Take the gradient of the loss function*’
  2. Pick random values for the parameters
  3. Plug the parameter values into the derivatives
  4. Calculate the step sizes:
     + Step size = slope \* learning rate
  5. Calculate the new parameters:
     + New Parameter = old parameter – step size
  6. Repeat step 3 until step size is very small or you reach maximum number of steps
* Problem: when there are millions of data points, algorithm can take a long time
* Solution: Use Stochastic Gradient Descent
  1. Uses a randomly selected subset of the data at every step rather than full dataset
  2. This reduces the time to calculate the derivatives of the loss function

**Lecture 4: Save and load model using Joblib and Pickle**

* Pickle module allows us to serialize python object into a file
* Python code:
  + *import pickle*
  + put model into a file: *with open(‘model\_pickel’,’wb’) as f: pickle.dump(model,f)*
  + retrieve model: *with open(‘model\_pickel’,’rb’) as f: mp = pickle.load(f)*
* You can also use Joblib which is more efficient on objects that have large numpy arrays
* Python code:
  + *import joblib*
  + put model into file: *joblib.dump(model, ‘model\_joblib’)*
  + Load model: mj = *joblib.load(‘model\_joblib’)*

**Lecture 5: Dummy variables & One Hot Encoding**

* Categorical variables
  + Nominal
    - These variables don’t have any order between them
    - Ex. {"Green”,”Red”,”Blue”}
  + Ordinal
    - Categories have a numberical ordering between them
    - Ex. {“High”,”Medium”,”Low”}
* One Hot Encoding
  + Problem: Need to assign numbers to nominal variables
  + We don’t want ordering between the numbers, so it isn’t as simple as giving the numbers 0 -> n
  + Table

    Description automatically generatedSolution: Assign each value to a binary number (called dummy variables)
* Python code
  + Create dummy columns for the ‘town’ variable: *pd.get\_dummies(df.town)*
  + Merge original data frame and dummy columns: pd.concat([df, dummies], axis=’columns’)
  + Need to drop one of the dummy variable columns because we don’t want the data to have a collinear relationship
  + *final = merged.drop([‘town’,west windsor], axis=’columns’)*
* Find the score of a model you created
  + Provide the input and output points
  + *model.score(X,y)*

**Lecture 6: Training and Testing data**

* Split dataset into training and testing data
* Python code:
  + Import splitting method for testing and training data:
    - *From sklearn.model\_selection import train\_test\_split*
  + Divided training and testing data by percentages (random sampling):
    - X\_train, X\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.2 )
  + Random\_state parameter keeps the sampling the same every time
  + Do linear regression on training data
  + Predict on testing data

**Lecture 7: Logistic Regression (Binary Classification)**

* Linear Regression
  + Predicting Home prices, weather, stock price
  + Predicted values are continuous
* Classification
  + Is the email spam? , Will the customer buy life insurance? Which party is a person going to vote for?
  + Predicted values are categorical
* Classification types
  + Binary classification
    - Will customer buy life insurance? (Yes or NO)
  + Multiclass classification
    - Which party will you vote for (Democrats, Liberals, Conservatives etc)
* Chart, diagram

  Description automatically generatedSigmoid function
  + Converts input into range from 0 to 1
  + Apply sigmoid function on the input (y = mx + b)
* Python code
  + Import Logistic Regression: *from sklearn.linear\_model import LogisticRegression*
  + Create model: *model = LogisticRegression()*
  + *model.predict\_proba(X\_test)*

**Lecture 7: Logistic Regression (Multiclass Classification)**

Diagram

Description automatically generated**Lecture 8: Decision tree**

* How do you select the ordering of the features in the decision tree?
  + Use the approach that gives you high information gain
  + Gini impurity
    - When the sample has a bit of impurity (the division of classification leaves some overlap)
* Python code
  + Change words to numbers
    - *from sklearn.preprocessing import LabelEncoder*
    - Change word to numbers: *le\_company = LabelEncoder()*
    - Add columns to inputs data frame
    - *inputs['company\_n'] = le\_company.fit\_transform(inputs['company'])*
  + Decision tree
    - *from sklearn import tree*
    - *model = tree.DecisionTreeClassifier()*

**Lecture 9: Support Vector Machine**

* Tries to maximize the margin between classifications
* Chart, scatter chart

  Description automatically generatedSupport vector machine draws a hyper plane in n-dimensions space such that maximizes margin between classification groups
* Chart, scatter chart

  Description automatically generatedGamma
* Regularization
  + High Regularization is fitting the line exactly to the data points
  + In sklearn, the paramter ‘c’ means regularizationChart, scatter chart

    Description automatically generated
* Kernel
  + Creating a transformation on existing features
* Python code
  + *from sklearn.svm import SVC*
  + *model = SVC()* (parameter C can be set ex. *model = SVC(C=10)*)

**Lecture 10: Random forest algorithm**

* Random forest analogy
  + A student want to choose what is major should be in University
  + He consults various people like his cousins, teachers , degree students etc
  + He asks them questions like: course fee, job oppurtunities, course life
  + He decides to take the course suggested by the most people (majority vote)
* Steps for algorithm:
  + Subset of data points are collected for constucting decision tree
  + Individula trees are constructed for each decision tree
  + Each tree will generate an output
  + Final output is considered based on majority voting or averaging
* Python code
  + Creating random forest algorithms
    - *from sklearn.ensemble import RandomForestClassifier*
    - *model = RandomForestClassifier()*
    - *model = RandomForestClassifier(n\_estimators=30)*
    - *n\_estimators=30* => creates 30 random trees

**Lecture 11: Cross validation**

* Cross validation allows you to evaluate a particular models performance
* Ways to train your model
  + Use all the available data for training and testing on same dataset
    - Ex. Train a kid for Math test with 100 math questions. On the test, you ask him those same questions and see how well he does.
    - This is not a good way of measuring his skills because he has already seen those questions before
  + Split available datasets into training and test sets
    - We have been using this “*train\_test\_splilt*” method for all of the models so far
    - Problem: What if the training data does not see the entire picture and misses out on a major aspect of the data?
  + K fold cross validation
    - Chart

      Description automatically generatedDivide 100 samples into folds (5 folds each with 20 samples)
* Python code
  + Implement kfolds
    - from sklearn.model\_selection import StratifiedKFold
    - folds = StratifiedKFold(n\_splits=3)
  + Automatically get a score that compares different ML models
    - *from sklearn.model\_selection import cross\_val\_score*
    - *cross\_val\_score(RandomForestClassifier(), digits.data, digits.target)*

**Lecture 13: Naive Bayes Classifier algorithm**

* Conditional probability:
  + Probability of B given A occurred
  + P(B | A) = P(A ∩ B)/P(A)
  + P(A | B) = P(A)\* P(B | A)P(B)
* Naïve assumption that features such as male, class, age , cabin are independent of each other in Titanic scenario
* Naïve bayes is used in weather predication, face detection etc
* Python code
  + Naïve Bayes
    - *from sklearn.naive\_bayes import GaussianNB*
    - *model = GaussianNB()*

**Lecture 14: Naive Bayes Classifier algorithm Part 2**

* Sklearn CountVectorizer
  + Covert groups of words to numbers by assingning values to unique attributes
* Naïve Bayes classifiers
  + Bernoulli Naïve Bayes
    - Assumes that all features are binary. 0s can represent “word does not occur in doccument”. 1s can represent “word occurs in document”
  + Multinominal Naïve Bayes
    - Used when we have discrete data. In text learning, we have the count of each word to predict the class or label
    - Ex. Movie rating from 1 to 5
  + Gaussian Naïve Bayes
    - When features are continous and based on assumptoin of normal distribution.
* Python code
  + *from sklearn.naive\_bayes import MultinomialNB*
  + *from sklearn.pipeline import Pipeline*
  + *from sklearn.feature\_extraction.text import CountVectorizer*
  + *clf = Pipeline([*
  + *('vectorizer', CountVectorizer()),*
  + *('nb', MultinomialNB())*
  + *])*
  + *clf.fit(X\_train, y\_train)*

**Lecture 16: L1 and L2 Regularization**

* **Text

  Description automatically generated**Problem is you don’t want to overfit your data as it will not be a good generalization of future trends
* The red is basically a penalty for making the feature values too high. It prevents equations from overfitting the data.
* You can fine tune the lambda value
* Text, letter

  Description automatically generatedPython code
  + Printing unique values in each column
    - *df.nunique()*
  + One Hot Encoding as some of the columns are in words not numbers
    - *df = pd.get\_dummies(df,drop\_first=True)*
    - Dropping the first column avoids the dummy variable trap
  + L1 regularization
    - *from sklearn import linear\_model*
    - *lasso\_reg = linear\_model.Lasso(alpha=50, max\_iter=1000, tol=0.1)*
    - *lasso\_reg.fit(X\_train,y\_train)*
  + L2 regularization
    - *from sklearn.linear\_model import Ridge*
    - *ridge\_reg = Ridge(alpha=50, max\_iter=1000, tol=0.1)*
    - *ridge\_reg.fit(X\_train,y\_train)*

**Lecture 18: K nearest neighbours classification**

* Goal: To classify a new datapoint into a specific category
* Graphical user interface

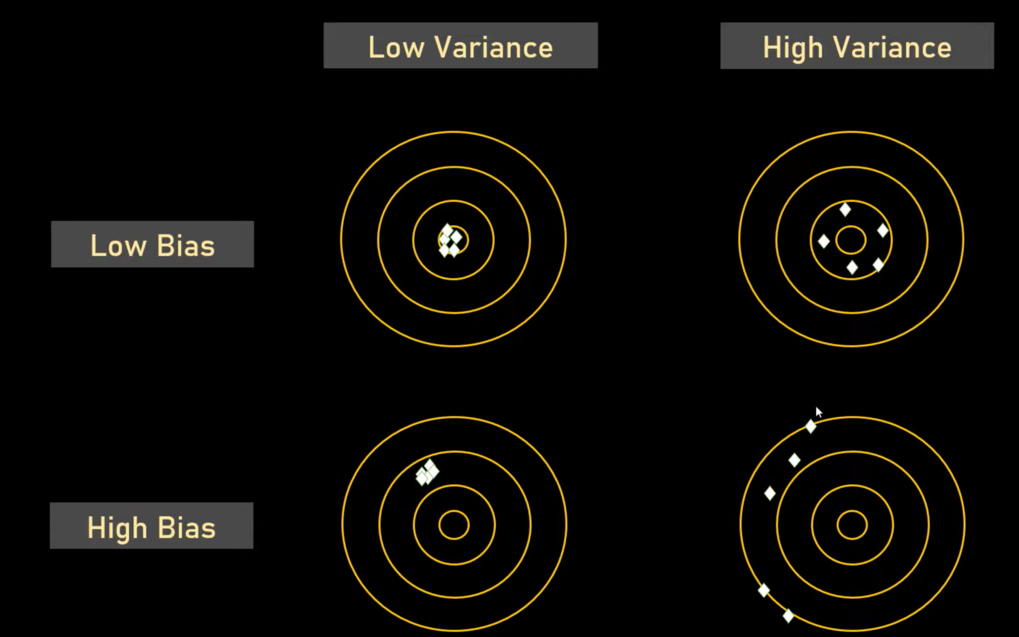
  Description automatically generated with medium confidenceSolution: Find the k-nearest data points and put it in the category with the most amount of nearest data points
* Fine tuning the ‘k’ is the crux of the algorithm
* Python code
  + Implement knn algorithm
    - *from sklearn.neighbors import KNeighborsClassifier*
    - *knn = KNeighborsClassifier(n\_neighbors=3)*
  + Confusion matrix (tells you for which classes the prediction was correct)
    - *from sklearn.metrics import confusion\_matrix*
    - *y\_pred = knn.predict(X\_test)*
    - *confusion\_matrix(y\_test, y\_pred)*
  + Instead of confusion matrix, you can also use classification report
    - *from sklearn.metrics import classification\_report*
    - *print(classification\_report(y\_test, y\_pred))*

**Lecture 19: Principal Component Analysis**

* PCA is a dimensionality reduction tool that can help reduce dimensions in large datasets
* It’s a process of figuring out the most important features or principal components that has the largest impact on the target variable
* Simple solution: Get rid of non important features
  + Faster training
  + *A picture containing graphical user interface

    Description automatically generated*Data visualization becomes easier
* Things to keep in mind before using PCA
  + Scale the features (same units) before using PCA
  + Accuracy of the model could drop
    - Ex. Let’s say you have 100 features and all 100 are integral to the target variable, accuracy will significantly drop if you reduce the features to 5
* Solve Dimensionality curse problem!!
  + Datasets with too many inputs (makes dataset hard to visualize)
* Python code
  + Scaling features before applying PCA
    - *from sklearn.preprocessing import StandardScaler*
    - *scaler = StandardScaler()*
    - *X\_scaled = scaler.fit\_transform(X)*
    - *X\_scaled*
  + PCA
    - *from sklearn.decomposition import PCA*
    - *pca = PCA(0.95)*
    - *X\_pca = pca.fit\_transform(X)*
    - Calculates new columns

**Lecture 20: Bias vs Variance**

* Bias
  + Measurement of how accurately a model can capture a pattern in the training dataset
* Overfit model almost traces each data point in the training dataset
  + It leads to a training dataset error of 0
  + Problem: Test dataset error can be very high
  + Leads to high variance (the test dataset error depends on what random samples are chosen for the training dataset)
* Underfit model
  + very simplistic model
  + does not truly capture the pattern in the training dataset
  + Leads to low variance (the test dataset error does not vary that much)
  + High bias
* Balanced fit
  + Low bias, low variance
  + Training dataset error and testing dataset error is low
* ****\*Remember you pick your training dataset at random\*

**Lecture 21: Ensemble Learning**