hw3

November 13, 2018

```
In [73]: import pandas as pd
         import numpy as np
         data = pd.read_csv("spambase/spambase.data")
```

1 Problem 1

1.1 Part A

```
In [288]: from sklearn.model_selection import train_test_split
          train, test = train_test_split(data, test_size=0.25)
          X = train.drop(train.columns[-1], axis=1)
          y = train[train.columns[-1]]
          y_true = test[test.columns[-1]]
          X_test = test.drop(train.columns[-1], axis=1)
```

```
In [289]: from sklearn.ensemble import RandomForestClassifier
          from sklearn.metrics import confusion_matrix
          model = RandomForestClassifier()
          model.fit(X,y)
```

```
Out[289]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                      oob_score=False, random_state=None, verbose=0,
                      warm_start=False)
```

```
In [315]: def print_stats(model):
              print("Statistics on Testing")
              y_pred = model.predict(X_test)
              cm = confusion_matrix(y_true, y_pred)
              tn, fp, fn, tp = cm.ravel()
              print("True Negative:", tn, "False Positives:", fp, "False Negatives:", fn, "Tru
              accuracy = model.score(X_test, y_true)
              error = 1 - accuracy
              print("Accuracy:", accuracy, "Error:", error)
              precision = tp / (tp + fp)
```

```
    recall = tp / (tp + fn)
    f1 = 2 * (precision * recall) / (precision + recall)
    print("Precision:", precision, "Recall:", recall, "F1:", f1)
    print("Statistics on Training")
    y_pred = model.predict(X)
    cm = confusion_matrix(y, y_pred)
    tn, fp, fn, tp = cm.ravel()
    print("True Negative:", tn, "False Positives:", fp, "False Negatives:", fn, "Tru
    accuracy = model.score(X, y)
    error = 1 - accuracy
    print("Accuracy:", accuracy, "Error:", error)
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    f1 = 2 * (precision * recall) / (precision + recall)
    print("Precision:", precision, "Recall:", recall, "F1:", f1)
```

```
In [316]: print_stats(model)
```

```
Statistics on Testing
True Negative: 667 False Positives: 18 False Negatives: 49 True Positives: 416
Accuracy: 0.9417391304347826 Error: 0.058260869565217366
Precision: 0.9585253456221198 Recall: 0.8946236559139785 F1: 0.9254727474972192
Statistics on Training
True Negative: 2099 False Positives: 4 False Negatives: 5 True Positives: 1342
Accuracy: 0.9973913043478261 Error: 0.00260869565217392
Precision: 0.9970282317979198 Recall: 0.9962880475129918 F1: 0.9966580022279985
```

1.2 Part B

```
In [406]: from sklearn.utils import resample
          from sklearn.feature_selection import mutual_info_classif
          from sklearn.metrics import accuracy_score
          from math import log
          from math import isnan
          class Node:
              def __init__(self, col, val, isLeaf):
                  self.col = col
                  self.val = val
                  self.left = None
                  self.right = None
                  self.isLeaf = isLeaf

              def predict(self, x):
                  if self.isLeaf:
                      return self.val
                  lOrR = x[str(self.col)] >= self.val
                  if lOrR:
```

```
            return self.right.predict(x)
        else:
            return self.left.predict(x)
def entropy(ps):
    s = 0
    for p in ps:
        if p <= 0.000001:
            continue
        s += p * log(p,2)
    return -1 * s
def information_gain(data, col, val):
    part1 = data.loc[data[col] >= val]
    if len(part1) == 0:
        p = 0
    else:
        c = np.sum(part1.iloc[:,-1])
        p = c/len(part1)
    rightEntropy = entropy([p, 1-p])
    part2 = data.loc[data[col] < val]
    if len(part2) == 0:
        p == 0
    else:
        c = np.sum(part2.iloc[:,-1])
        p = c/len(part2)
    leftEntropy = entropy([p, 1-p])
    cond = len(part1)/len(data) * rightEntropy + len(part2)/len(data) * leftEntropy
    cWhole = np.sum(data.iloc[:,-1])
    pWhole = cWhole/len(data)
    return entropy([pWhole, 1-pWhole]) - cond
class RF:
    def __init__(self, trees, m, samples=1000):
        self.trees = trees
        self.m = m
        self.samples = 1000
    def train(self, train):
        trees = self.trees
        m = self.m
        t = []
        def build_tree(data, layers):
            xs = data.drop(train.columns[-1], axis=1)
            ys = data[data.columns[-1]]
            test = np.sum(ys)
            if test == len(ys) or test == 0:
                return Node(None, 0 if test == 0 else 1, True)
            if layers == 1:
                return Node(None, 1 if test > len(ys)/2 else 0, True)
            selected = xs.sample(m, axis=1).join(ys)
            bestIg = -1
```

```
                    bestCol = None
                    for col in selected.columns[:-1]:
                        sep = selected[col].unique()
                        if len(sep) > 5:
                            _, sep = np.histogram(sep, bins=5)
                        for val in sep:
                            ig = information_gain(selected, col, val)
                            if ig > bestIg:
                                bestIg = ig
                                bestCol = Node(col, val, False)
                    left = data[data[bestCol.col] < bestCol.val]
                    right = data[data[bestCol.col] >= bestCol.val]
                    bestCol.left = build_tree(left, layers - 1)
                    bestCol.right = build_tree(right, layers - 1)
                    return bestCol
                for i in range(trees):
                    s = resample(train, n_samples=self.samples)
                    t.append(build_tree(s, 10))
                self.t = t
            def predict(self, x_test):
                def pr(x):
                    s = 0
                    for i in self.t:
                        s += i.predict(x)
                    return 1 if s >= (len(self.t)/2) else 0
                return x_test.apply(pr, axis=1)
            def score(self, x, y_true):
                y_pred = self.predict(x)
                return accuracy_score(y_true, y_pred)
```

1.3 Part C

```
In [400]: m = len(X.columns)
          myModel = RF(5, m)
          myModel.train(train)
```

```
Tree 1 is done
Tree 2 is done
Tree 3 is done
Tree 4 is done
Tree 5 is done
```

```
In [401]: print_stats(myModel)
```

```
Statistics on Testing
True Negative: 670 False Positives: 15 False Negatives: 94 True Positives: 371
Accuracy: 0.9052173913043479 Error: 0.09478260869565214
Precision: 0.961139896373057 Recall: 0.7978494623655914 F1: 0.8719153936545242
```

```
Statistics on Training
True Negative: 2046 False Positives: 57 False Negatives: 210 True Positives: 1137
Accuracy: 0.922608695652174 Error: 0.07739130434782604
Precision: 0.9522613065326633 Recall: 0.844097995545657 F1: 0.8949232585596221
```

```
In [398]: d = len(X.columns)
          myModel = RF(5, d//2, 500)
          myModel.train(train)
```

```
Tree 1 is done
Tree 2 is done
Tree 3 is done
Tree 4 is done
Tree 5 is done
```

```
In [399]: print_stats(myModel)
```

```
Statistics on Testing
True Negative: 659 False Positives: 26 False Negatives: 89 True Positives: 376
Accuracy: 0.9 Error: 0.09999999999999998
Precision: 0.9353233830845771 Recall: 0.8086021505376344 F1: 0.8673587081891581
Statistics on Training
True Negative: 2046 False Positives: 57 False Negatives: 214 True Positives: 1133
Accuracy: 0.9214492753623188 Error: 0.0785507246376812
Precision: 0.9521008403361344 Recall: 0.8411284335560505 F1: 0.8931809223492313
```

```
In [396]: from math import sqrt
          d = len(X.columns)
          myModel = RF(5, int(sqrt(d)))
          myModel.train(train)
```

```
Tree 1 is done
Tree 2 is done
Tree 3 is done
Tree 4 is done
Tree 5 is done
```

```
In [397]: print_stats(myModel)
```

```
Statistics on Testing
True Negative: 661 False Positives: 24 False Negatives: 98 True Positives: 367
Accuracy: 0.8939130434782608 Error: 0.10608695652173916
Precision: 0.9386189258312021 Recall: 0.789247311827957 F1: 0.8574766355140188
Statistics on Training
True Negative: 2034 False Positives: 69 False Negatives: 237 True Positives: 1110
Accuracy: 0.9113043478260869 Error: 0.08869565217391306
Precision: 0.9414758269720102 Recall: 0.8240534521158129 F1: 0.8788598574821853
```

1.4 Part D

```
In [402]: from math import sqrt
          d = len(X.columns)
          m = int(sqrt(d))
          myModel = RF(10, m)
          myModel.train(train)
          y_pred = myModel.predict(X_test)
          model = RandomForestClassifier(n_estimators=10)
          model.fit(X,y)
          print("\nStats of Mine\n")
          print_stats(myModel)
          print("Stats of Package\n")
          print_stats(model)
```

```
Tree 1 is done
Tree 2 is done
Tree 3 is done
Tree 4 is done
Tree 5 is done
Tree 6 is done
Tree 7 is done
Tree 8 is done
Tree 9 is done
Tree 10 is done

Stats of Mine

Statistics on Testing
True Negative: 666 False Positives: 19 False Negatives: 82 True Positives: 383
Accuracy: 0.9121739130434783 Error: 0.08782608695652172
Precision: 0.9527363184079602 Recall: 0.8236559139784946 F1: 0.8835063437139561
Statistics on Training
True Negative: 2043 False Positives: 60 False Negatives: 187 True Positives: 1160
Accuracy: 0.9284057971014493 Error: 0.07159420289855067
Precision: 0.9508196721311475 Recall: 0.8611729769858946 F1: 0.9037787300350604
Stats of Package

Statistics on Testing
True Negative: 667 False Positives: 18 False Negatives: 46 True Positives: 419
Accuracy: 0.9443478260869566 Error: 0.055652173913043446
Precision: 0.9588100686498856 Recall: 0.9010752688172043 F1: 0.9290465631929047
Statistics on Training
True Negative: 2097 False Positives: 6 False Negatives: 8 True Positives: 1339
Accuracy: 0.9959420289855072 Error: 0.004057971014492789
Precision: 0.995539033457249 Recall: 0.994060876020787 F1: 0.9947994056463595
```

```
In [404]: d = len(X.columns)
```

```
m = int(sqrt(d))
myModel = RF(50, m, 200)
myModel.train(train)
y_pred = myModel.predict(X_test)
model = RandomForestClassifier(n_estimators=50)
model.fit(X,y)
print("\nStats of Mine\n")
print_stats(myModel)
print("Stats of Package\n")
print_stats(model)
```

```
Tree 1 is done
Tree 2 is done
Tree 3 is done
Tree 4 is done
Tree 5 is done
Tree 6 is done
Tree 7 is done
Tree 8 is done
Tree 9 is done
Tree 10 is done
Tree 11 is done
Tree 12 is done
Tree 13 is done
Tree 14 is done
Tree 15 is done
Tree 16 is done
Tree 17 is done
Tree 18 is done
Tree 19 is done
Tree 20 is done
Tree 21 is done
Tree 22 is done
Tree 23 is done
Tree 24 is done
Tree 25 is done
Tree 26 is done
Tree 27 is done
Tree 28 is done
Tree 29 is done
Tree 30 is done
Tree 31 is done
Tree 32 is done
Tree 33 is done
Tree 34 is done
Tree 35 is done
Tree 36 is done
Tree 37 is done
```

```
Tree 38 is done
Tree 39 is done
Tree 40 is done
Tree 41 is done
Tree 42 is done
Tree 43 is done
Tree 44 is done
Tree 45 is done
Tree 46 is done
Tree 47 is done
Tree 48 is done
Tree 49 is done
Tree 50 is done

Stats of Mine

Statistics on Testing
True Negative: 677 False Positives: 8 False Negatives: 94 True Positives: 371
Accuracy: 0.9113043478260869 Error: 0.08869565217391306
Precision: 0.978891820580475 Recall: 0.7978494623655914 F1: 0.8791469194312795
Statistics on Training
True Negative: 2068 False Positives: 35 False Negatives: 218 True Positives: 1129
Accuracy: 0.9266666666666666 Error: 0.07333333333333336
Precision: 0.9699312714776632 Recall: 0.838158871566444 F1: 0.8992433293508563
Stats of Package

Statistics on Testing
True Negative: 669 False Positives: 16 False Negatives: 38 True Positives: 427
Accuracy: 0.9530434782608695 Error: 0.04695652173913045
Precision: 0.963882618510158 Recall: 0.9182795698924732 F1: 0.9405286343612336
Statistics on Training
True Negative: 2101 False Positives: 2 False Negatives: 3 True Positives: 1344
Accuracy: 0.9985507246376811 Error: 0.0014492753623188692
Precision: 0.9985141158989599 Recall: 0.9977728285077951 F1: 0.9981433345711103
```

```
In [407]: d = len(X.columns)
          m = int(sqrt(d))
          myModel = RF(100, m, 100)
          myModel.train(train)
          y_pred = myModel.predict(X_test)
          model = RandomForestClassifier(n_estimators=50)
          model.fit(X,y)
          print("\nStats of Mine\n")
          print_stats(myModel)
          print("Stats of Package\n")
          print_stats(model)
```

```
Stats of Mine

Statistics on Testing
True Negative: 675 False Positives: 10 False Negatives: 140 True Positives: 325
Accuracy: 0.8695652173913043 Error: 0.13043478260869568
Precision: 0.9701492537313433 Recall: 0.6989247311827957 F1: 0.8125
Statistics on Training
True Negative: 2077 False Positives: 26 False Negatives: 372 True Positives: 975
Accuracy: 0.8846376811594203 Error: 0.11536231884057968
Precision: 0.974025974025974 Recall: 0.7238307349665924 F1: 0.8304940374787053
Stats of Package

Statistics on Testing
True Negative: 672 False Positives: 13 False Negatives: 40 True Positives: 425
Accuracy: 0.9539130434782609 Error: 0.04608695652173911
Precision: 0.9703196347031964 Recall: 0.9139784946236559 F1: 0.9413067552602437
Statistics on Training
True Negative: 2102 False Positives: 1 False Negatives: 2 True Positives: 1345
Accuracy: 0.9991304347826087 Error: 0.0008695652173913437
Precision: 0.9992570579494799 Recall: 0.9985152190051967 F1: 0.9988860007426661
```

My implementation is definitely slower than the package's as I had to use smaller bootstrapping size to get results. However, the accuracy is close to that of the package as I can get around 90% while the package delivers 94%.

2 Problem 2

2.1 Part A

```
In [410]: from sklearn.ensemble import AdaBoostClassifier
          model = AdaBoostClassifier()
          model.fit(X,y)

Out[410]: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
                    learning_rate=1.0, n_estimators=50, random_state=None)

In [411]: print_stats(model)

Statistics on Testing
True Negative: 659 False Positives: 26 False Negatives: 44 True Positives: 421
Accuracy: 0.9391304347826087 Error: 0.060869565217391286
Precision: 0.941834451901566 Recall: 0.9053763440860215 F1: 0.9232456140350879
Statistics on Training
True Negative: 2020 False Positives: 83 False Negatives: 91 True Positives: 1256
Accuracy: 0.9495652173913044 Error: 0.050434782608695605
Precision: 0.9380134428678119 Recall: 0.9324424647364514 F1: 0.9352196574832464
```

2.2 Part B

```
In [412]: from sklearn.linear_model import LogisticRegression
          model = AdaBoostClassifier(base_estimator=LogisticRegression(random_state=0))
          model.fit(X,y)

Out[412]: AdaBoostClassifier(algorithm='SAMME.R',
                    base_estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fi
                    intercept_scaling=1, max_iter=100, multi_class='warn',
                    n_jobs=None, penalty='l2', random_state=0, solver='warn',
                    tol=0.0001, verbose=0, warm_start=False),
                    learning_rate=1.0, n_estimators=50, random_state=None)

In [413]: print_stats(model)

Statistics on Testing
True Negative: 651 False Positives: 34 False Negatives: 73 True Positives: 392
Accuracy: 0.9069565217391304 Error: 0.09304347826086956
Precision: 0.92018779342723 Recall: 0.843010752688172 F1: 0.8799102132435466
Statistics on Training
True Negative: 1991 False Positives: 112 False Negatives: 169 True Positives: 1178
Accuracy: 0.9185507246376812 Error: 0.08144927536231883
Precision: 0.9131782945736434 Recall: 0.874536005939124 F1: 0.8934395145999242
```

2.3 Part C

```
In [168]: numClassifiers = [10,50,100]
          import warnings; warnings.simplefilter('ignore')
          for i in numClassifiers:
              model = AdaBoostClassifier(base_estimator=LogisticRegression(random_state=0), n_
              model.fit(X,y)
              accuracy = model.score(X_test, y_true)
              print("Classifiers:",i, "Accuracy:", accuracy, "Error:", error)

Classifiers: 10 Accuracy: 0.9095652173913044 Error: 0.07826086956521738
Classifiers: 50 Accuracy: 0.9217391304347826 Error: 0.07826086956521738
Classifiers: 100 Accuracy: 0.9217391304347826 Error: 0.07826086956521738
```

The accuracy did increase as more estimators were added but it seemed to be to a certain

2.4 Part D

The AdaBoost model gave about 92% accuracy with 100 estimators while the RandomForestModel gave 95% accuracy with 100 random trees. The Random Forest was definitely better as both those scores were on the testing data.

3 Problem 3

3.1 Part A and Part B

```
In [49]: from keras.datasets import mnist
         (X_train, y_train), (X_test, y_test) = mnist.load_data()
         X_train = X_train / 255
         X_test = X_test / 255

In [56]: import keras
         model = keras.Sequential([
             keras.layers.Flatten(input_shape=(28, 28)),
             keras.layers.Dense(128, activation="relu"),
             keras.layers.Dense(10, activation="softmax")
         ])
         model.compile(loss="sparse_categorical_crossentropy",
                       optimizer="sgd", metrics=["accuracy"])
         model.fit(X_train, y_train, epochs=5)

Epoch 1/5
60000/60000 [==============================] - 7s 115us/step - loss: 0.6492 - acc: 0.8381
Epoch 2/5
60000/60000 [==============================] - 7s 112us/step - loss: 0.3363 - acc: 0.9060
Epoch 3/5
60000/60000 [==============================] - 5s 89us/step - loss: 0.2863 - acc: 0.9197
Epoch 4/5
60000/60000 [==============================] - 5s 90us/step - loss: 0.2547 - acc: 0.9283
Epoch 5/5
60000/60000 [==============================] - 6s 92us/step - loss: 0.2317 - acc: 0.9351


Out[56]: <keras.callbacks.History at 0x1251e67f0>

In [57]: test_loss, test_acc = model.evaluate(X_test, y_test)
         print("Accuracy:", test_acc)

10000/10000 [==============================] - 1s 67us/step
Accuracy: 0.9384
```

This fast forward neural network had basically two layers. One layer had 128 nodes with an activation function of relu while the next had 10 nodes with a soft max activation function. The accuracy was high with 93%.

```
In [51]: model = keras.Sequential([
             keras.layers.Flatten(input_shape=(28, 28)),
             keras.layers.Dense(10, activation="sigmoid"),
             keras.layers.Dense(10, activation="sigmoid"),
             keras.layers.Dense(10, activation="sigmoid")
```

```
        ])
        sgd = keras.optimizers.SGD(lr=0.01)
        model.compile(loss="sparse_categorical_crossentropy",
                      optimizer="sgd", metrics=["accuracy"])
        model.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
60000/60000 [==============================] - 4s 72us/step - loss: 2.3023 - acc: 0.1527
Epoch 2/5
60000/60000 [==============================] - 4s 60us/step - loss: 2.2801 - acc: 0.1297
Epoch 3/5
60000/60000 [==============================] - 4s 62us/step - loss: 2.2514 - acc: 0.1565
Epoch 4/5
60000/60000 [==============================] - 5s 81us/step - loss: 2.1920 - acc: 0.2322
Epoch 5/5
60000/60000 [==============================] - 4s 72us/step - loss: 2.0828 - acc: 0.4113
```

```
Out[51]: <keras.callbacks.History at 0x123b30c18>
```

```
In [52]: test_loss, test_acc = model.evaluate(X_test, y_test)
         print("Accuracy:", test_acc)
```

```
10000/10000 [==============================] - 1s 57us/step
Accuracy: 0.4384
```

This fast forward neural network had basically three layers, each layer was the same with 10 nodes and the activation function of sigmoid. Accuracy was low with only 40% on testing data and training.

```
In [7]: model = keras.Sequential([
            keras.layers.Flatten(input_shape=(28, 28)),
            keras.layers.Dense(128, activation="sigmoid"),
            keras.layers.Dense(128, activation="sigmoid"),
            keras.layers.Dense(10, activation="sigmoid")
        ])
        sgd = keras.optimizers.SGD(lr=0.01)
        model.compile(loss="sparse_categorical_crossentropy",
                      optimizer="sgd", metrics=["accuracy"])
        model.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
60000/60000 [==============================] - 6s 99us/step - loss: 2.2861 - acc: 0.1596
Epoch 2/5
60000/60000 [==============================] - 5s 85us/step - loss: 2.1507 - acc: 0.4887
Epoch 3/5
60000/60000 [==============================] - 5s 87us/step - loss: 1.6126 - acc: 0.6832
Epoch 4/5
```

```
60000/60000 [==============================] - 7s 109us/step - loss: 1.0148 - acc: 0.7764
Epoch 5/5
60000/60000 [==============================] - 6s 96us/step - loss: 0.7316 - acc: 0.8271
```

```
Out[7]: <keras.callbacks.History at 0x13bf125f8>
```

```
In [8]: test_loss, test_acc = model.evaluate(X_test, y_test)
        print("Accuracy:", test_acc)
```

```
10000/10000 [==============================] - 1s 50us/step
Accuracy: 0.8463
```

This fast forward neural network had basically three layers. The first layer and second layer had 128 nodes with activation function of sigmoid. The last layer only had 10 nodes with activation function of sigmoid as well.

3.2 Part C and Part D

```
In [58]: from keras.utils import to_categorical
         X_train = X_train.reshape(60000,28,28,1)
         X_test = X_test.reshape(10000,28,28,1)
         y_train = to_categorical(y_train)
         y_test = to_categorical(y_test)
```

```
In [60]: model = keras.Sequential()
         model.add(keras.layers.Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28
         model.add(keras.layers.Conv2D(32, kernel_size=3, activation='relu'))
         model.add(keras.layers.Flatten())
         model.add(keras.layers.Dense(10, activation='softmax'))
         model.compile(loss="categorical_crossentropy",
                       optimizer="sgd", metrics=["accuracy"])
         model.fit(X_train, y_train, epochs=3)
```

```
Epoch 1/3
60000/60000 [==============================] - 171s 3ms/step - loss: 0.4309 - acc: 0.8754
Epoch 2/3
60000/60000 [==============================] - 193s 3ms/step - loss: 0.2487 - acc: 0.9302
Epoch 3/3
60000/60000 [==============================] - 169s 3ms/step - loss: 0.1421 - acc: 0.9600
```

```
Out[60]: <keras.callbacks.History at 0x126aa1978>
```

```
In [61]: test_loss, test_acc = model.evaluate(X_test, y_test)
         print("Accuracy:", test_acc)
```

```
10000/10000 [==============================] - 9s 857us/step
Accuracy: 0.9708
```

This is a 3 layered network with 2 Convolutional layers, one with 64 nodes and the other with 32. The filter size was 3 on each (3x3) with activation function of relu. The last layer was a dense layer with 10 nodes and a softmax activation function.

```
In [67]: model = keras.Sequential()
         model.add(keras.layers.Conv2D(16, kernel_size=6, activation='relu', input_shape=(28,28
         model.add(keras.layers.Conv2D(4, kernel_size=2, strides=(2,1), activation='relu'))
         model.add(keras.layers.Flatten())
         model.add(keras.layers.Dense(10, activation='softmax'))
         model.compile(loss="categorical_crossentropy",
                       optimizer="sgd", metrics=["accuracy"])
         model.fit(X_train, y_train, epochs=3, batch_size=150)

Epoch 1/3
60000/60000 [==============================] - 29s 488us/step - loss: 1.5247 - acc: 0.5225
Epoch 2/3
60000/60000 [==============================] - 28s 473us/step - loss: 0.3724 - acc: 0.8937
Epoch 3/3
60000/60000 [==============================] - 26s 428us/step - loss: 0.2893 - acc: 0.9165


Out[67]: <keras.callbacks.History at 0x129ce1d68>

In [68]: test_loss, test_acc = model.evaluate(X_test, y_test)
         print("Accuracy:", test_acc)

10000/10000 [==============================] - 3s 270us/step
Accuracy: 0.9295
```

This is a 3 layered network with 2 Convolutional layers, one with 16 nodes and the other with 4. The filter size was 6 on on (6x6) and the other with 2 (2x2) with stride of 2 with activation function of relu. The last layer was a dense layer with 10 nodes and a softmax activation function.

```
In [46]: model = keras.Sequential()
         model.add(keras.layers.Conv2D(16, kernel_size=6, activation='relu', input_shape=(28,28
         model.add(keras.layers.MaxPooling2D(pool_size=(3, 3), strides=None))
         model.add(keras.layers.Flatten())
         model.add(keras.layers.Dense(10, activation='softmax'))
         model.compile(loss="categorical_crossentropy",
                       optimizer="adam", metrics=["accuracy"])
         model.fit(X_train, y_train, epochs=3)

Epoch 1/3
60000/60000 [==============================] - 28s 463us/step - loss: 6.8311 - acc: 0.5685
Epoch 2/3
60000/60000 [==============================] - 26s 437us/step - loss: 2.0435 - acc: 0.8645
Epoch 3/3
60000/60000 [==============================] - 26s 435us/step - loss: 1.8580 - acc: 0.8790
```

```
Out[46]: <keras.callbacks.History at 0x1229d4438>

In [47]: test_loss, test_acc = model.evaluate(X_test, y_test)
         print("Accuracy:", test_acc)

10000/10000 [==============================] - 3s 283us/step
Accuracy: 0.8695
```

This is a 3 layered network with 1 Convolutional layers and 1 max pooling layer. The first layer is a 16 node convolutional layer with filter size 6x6. The second layer is a max pooling with size 3x3. The last layer is 10 node layer for the output.

3.3 Part E

The Convulational layer performed better as the model had accuracy of 97% with two layers of convlutional and one for output while fast forward had 94%. These were tested on the test data. However, the Convulational took much longer to train. For each epoch, it took about 30s to completely train over the training set. While the fast forward took only about 5s per epoch. Especially on my machine, this made it possible to train more epochs with the fast forward neural net than it did for the convulational.